Generación de Ambientes para Entrenamiento en Seguridad Informática

Alejandro Blanco, Juan Diego Campo, Lucía Escanellas, Carlos Pintado, and Marcelo Rodríguez

Grupo de Seguridad Informática, Instituto de Computación, Facultad de Ingeniería, Universidad de la República
J. Herrera y Reissig 565, Montevideo, Uruguay
[ablanco,jdcampo,luciae,cpintado,marcelor]@fing.edu.uy
http://www.fing.edu.uy/inco/gsi

Resumen Los ataques informáticos son un tema latente en la actualidad. Para poder contrarrestar los riegos, es necesario contar con profesionales informáticos especializados en seguridad. Dichos profesionales deben de ser capaces de entender cuáles son los riesgos, las amenazas y las vulnerabilidades a las que se están enfrentando. Para ello necesitan aprender tanto como sea posible de los problemas, y poder articular respuestas rápidas ante incidentes de seguridad.
En este contexto, poder contar con ambientes de entrenamiento en los cuales se puedan experimentar con técnicas, herramientas y problemas reales es fundamental.
Sin embargo, si se hace en forma manual, la creación y el mantenimiento de este tipo de ambientes se torna una tarea compleja y que consume mucho tiempo. Esto motiva el desarrollo de una herramienta que asista en la generación de ambientes para entrenamiento en seguridad informática.
En este artículo se describe la arquitectura de una herramienta orientada a solucionar la problemática planteada. Como parte del diseño, se definió un lenguaje de especificación de prácticas de seguridad, que permite la declaración e instanciación de escenarios. Los escenarios, generados a partir de la especificación, emulan de manera fidedigna los ambientes de producción reales y permiten emplear herramientas comúnmente utilizadas en el mercado.

Palabras clave: Laboratorio de Seguridad Informática, Seguridad de la Información, Virtualización.

1. Introducción

El Grupo de Seguridad Informática (GSI) de la Facultad de Ingeniería de la Universidad de la República (Uruguay), imparte cursos de seguridad informática, que utilizan laboratorios basados en la metodología que se describe en "Concepción, Diseño e Implantación de un Laboratorio de Seguridad Informática" [2]. La metodología allí descripta está basada en el uso de virtualización para generar los ambientes de experimentación. Dicha solución fue aplicada con éxito en

varios cursos de grado y postgrado impartidos por el GSI desde el año 2006 a la fecha.

Sin embargo, la creación y el mantenimiento de laboratorios con estas características es una tarea compleja desarrollada en forma manual, y por ende, 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
  Se 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 generalmente 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. En este trabajo se discuten algunos detalles del análisis y diseño de la solución planteada.

El resto de este artículo esta estructurado como se describe a continuación. En la Sección 2 se discute sobre trabajos relacionados. La Sección 3 describe la arquitectura de la herramienta y se especifica el lenguaje para definición de escenarios. La Sección 4 expone los componentes de la herramienta implementada. En la Sección 5 se reseña el enfoque utilizado para el desarrollo de experimentos con la herramienta desarrollada. La Sección 6 concluye y describe trabajo futuro.

2. Trabajo relacionado

Como parte del análisis, se hizo un relevamiento en busca de soluciones que atacaran la problemática de la enseñanza práctica de seguridad informática.

Muchas de las soluciones encontradas implementan los laboratorios de igual manera que el GSI (por ejemplo, los laboratorios de la Universidad de Indiana [5], Drexel [13], Toledo [16] o el Instituto Rochester [12]) y por lo tanto experimentan la misma clase de problemas en lo referente al mantenimiento y administración.

Otras soluciones atacan la problemática mediante la utilización de "técnicas de juego", en donde se plantean desafíos de seguridad como ser: enseñanza de *ethical hacking*, prácticas de ingeniería inversa o cracking (por ejemplo: CyberCIEGE [4], WebGoat [17] o Mod-X [9]). En general este tipo de soluciones

utilizan simuladores que no son modificables por los usuarios, en lugar de aplicaciones reales y vulnerables. Este enfoque tiene la desventaja, si no se ha previsto en la simulación, de no contar con la realimentación proveniente de la interacción con procesos reales. Una ventaja es que se pueden diseñar las prácticas de manera tal que la evaluación se realice de manera automática.

Una iniciativa similar a la planteada en el presente trabajo es el proyecto Tele-Lab [7], desarrollado por el Departamento de Ciencias de la Computación de la Universidad de Trier en Alemania. La solución está orientada al e-Learning, basada en una aplicación Web mediante la cual se accede a un laboratorio implementado con máquinas virtuales. El inconveniente que se le encontró a la solución es que no se proveen mecanismos para que se puedan generar nuevos escenarios en forma sencilla.

3. Un Ambiente para Entrenamiento en Seguridad Informática

El trabajo que aquí se presenta es una continuación de lo expuesto en [2] y por lo tanto contribuye con el objetivo principal de contar con un "ambiente para el desarrollo de actividades de experimentación, investigación y enseñanza de seguridad informática", que además cumpla con una serie de requerimientos ahí descriptos. En ese trabajo, se discuten diferentes opciones de arquitectura para la solución, optando por la utilización de virtualización. Como aporte, se decidió estudiar otros paradigmas de virtualización, diferentes a los analizados, para incorporarlos a la solución. En particular, paravirtualización [11], y productos que la implementan (User Mode Linux [14]).

El objetivo que se comienza a explorar aquí es la concepción de una herramienta que facilite y automatice la creación de nuevos escenarios de experimentación. De manera adicional, se contempla la incorporación de mecanismos para la evaluación de prácticas realizadas sobre dichos escenarios.

3.1. Arquitectura de la Herramienta

La arquitectura que ha sido definida para la herramienta se ilustra en la Figura 1, la cual está inspirada en [7], contemplando las características que facilitan la generación de escenarios, así como también los inconvenientes que tiene la implementación actual de los laboratorios del GSI.

Se optó por una arquitectura en capas, que permite una implementación con modelo cliente servidor que brinda movilidad al laboratorio. La arquitectura consta de cuatro capas: la capa de presentación, la capa lógica, la capa de administración de prácticas y la capa de abstracción de máquinas virtuales.

La capa de presentación tiene dos *frontends*, uno para el estudiante y otro para el docente. Los mismos proveen una interfaz de usuario para el sistema, que puede ser remota, y proveen las operaciones necesarias para cada rol. Por ejemplo, el *frontend* del estudiante le brinda al mismo acceso a la práctica, y el *frontend* docente permite especificar prácticas y evaluarlas.

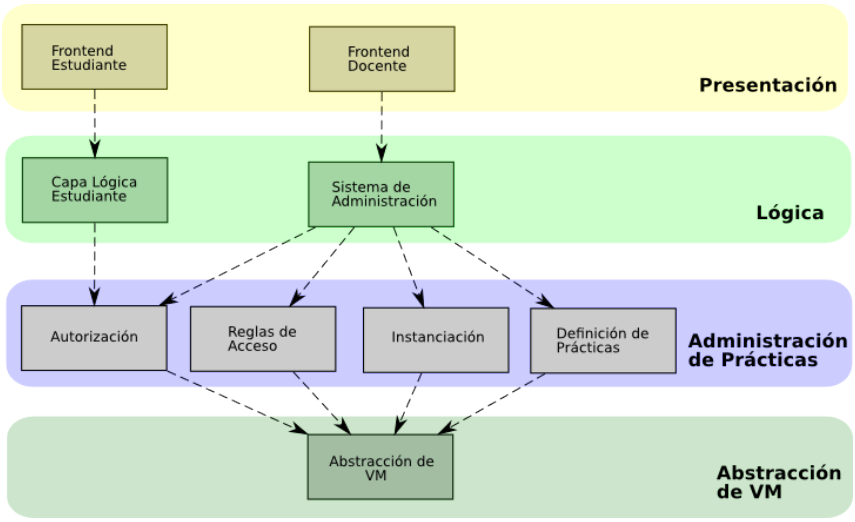

Figura 1. Arquitectura del Framework

La capa lógica comprende la *Capa Lógica del Estudiante* y el *Sistema de Administración*.

La *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.

El *Sistema de Administración* brinda una abstracción para que se pueda manejar el sistema con facilidad. En primer lugar tiene el objetivo de servir de interfaz con el módulo de *Instanciación*, ofreciendo comandos de alto nivel, tales como instanciar o detener una práctica, mapeándolos a comandos de bajo nivel en la tecnología de virtualización utilizada. También tiene como objetivo servir de interfaz con el módulo de *Definición de Prácticas*, ofreciendo una interfaz amigable, que genere la especificación requerida por dicho módulo.

La capa de administración de prácticas contiene módulos encargados de aspectos inherentes al desarrollo de las mismas.

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

El módulo de *Reglas de Acceso* es el encargado de controlar las reglas de firewall y de acceso a las máquinas. Es fundamental para implementar los requisitos de aislamiento que deben tener las prácticas.

El módulo de *Instanciación* se encarga de implementar la instanciación de prácticas. Su función es la de crear, ejecutar y detener las máquinas virtuales.

El módulo de *Definición de Prácticas* es el módulo encargado de implementar la creación de prácticas. 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, entre otros.

La capa de *Abstracción de VM* se encarga del manejo de la tecnología de virtualización utilizada, previendo la posibilidad de emplear más de una tecnología en forma concurrente. Contar con la posibilidad de utilizar más de una tecnología de virtualización en un mismo escenario permite sacar el máximo provecho de cada una de ellas.

La arquitectura aquí planteada es una primera aproximación al diseño general de un *framework* que implemente las funcionalidades necesarias para los laboratorios. En base a esto, se desarrolló una primera versión de la herramienta que permite avanzar en la investigación.

En la siguiente sección se detallan características del módulo de *Definición de Prácticas*, en particular, se destaca la definición de un lenguaje de especificación de prácticas de seguridad.

3.2. Lenguaje de definición de escenarios

Se considera un requerimiento importante que el módulo de *Definición de Prácticas* cuente con la facilidad de poder especificar los escenarios con un nivel de abstracción tal, que oculte detalles de implementación y configuración de bajo nivel.

Se investigaron alternativas para cumplir con este requerimiento, y en el contexto sobre el que se está trabajando, la búsqueda fue infructuosa. No así en el campo de simulación de redes, donde existen productos como Packet Tracer [10] (herramienta propietaria de Cisco [3]), ITguru [8] (herramienta para asistir en cursos de redes) y VNUML [15].

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. Fue pensado para instanciar escenarios de cualquier tipo. En sus comienzos, el mismo fue desarrollado para simular redes IPv6, pero puede ser utilizado para simular escenarios de red genéricos basados en linux. Como contrapartida, no está pensado para generar ambientes contemplando problemas de seguridad y se encuentra ligado, como su nombre lo indica, al producto de virtualización UML.

Utilizar solamente esta herramienta en los laboratorios, si bien facilita aspectos de configuración de redes, tiene como costo asociado tener que migrar a la tecnología de virtualización UML. Esto implica convertir las máquinas utilizadas a la nueva tecnología, teniendo la desventaja de que no es posible utilizar sistemas operativos distintos a los basados en linux.

Por otro lado, disponer de UML tiene como ventaja que la tecnología permite un uso eficiente de los recursos, mediante su técnica de *Copy on write* [1].

[1] *Copy on write* es una técnica que permite utilizar una imagen de disco base para una máquina virtual, en modo sólo lectura, y guardar en un archivo de diferencias los cambios realizados a esa imagen durante la ejecución de la máquina. Cuando varias

En busca de cumplir el requerimiento y ocultar los detalles de implementación y configuración a bajo nivel, se especificó un lenguaje formal. Este lenguaje, permite además independizar la especificación de escenarios del producto y paradigma de virtualización utilizado.

Dadas las características, el lenguaje adquiere una importancia sustancial dentro del la definición del *framework*.

A grandes rasgos, un escenario 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. Para el lenguaje se eligió una sintaxis similar a la definición de estructuras en C. Por ejemplo, la definición de una herramienta de ataque se muestra en la Figura 2.

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

Figura 2. Ejemplo de definición de estructura

Como se puede observar, la definición de una instancia de estructura se realiza declarando el tipo de la estructura (en el ejemplo, *Tool*), un identificador para esta instancia (*ettercap*), una lista de atributos (*name*, *description*, *version*, *running*, etc), 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 (como el atributo *operating_system*), cadena de caracteres (como *name*), número, booleano (*running*) o lista (*exploitable_vulnerabilities*). En la Figura 3, se pueden ver los componentes más importantes del lenguaje, sus atributos y relaciones.

El lenguaje, de aquí en más LISLab, 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.

Los criterios que guiaron el diseño de estas estructuras fueron la necesidad de representar una topología de red con máquinas virtuales con aplicaciones y

máquinas comparten la misma imagen base, guardando sus respectivos archivos de diferencias, el consumo de espacio en disco es mucho menor, debido a que sólo se almacena una vez la mayor parte del contenido del disco.

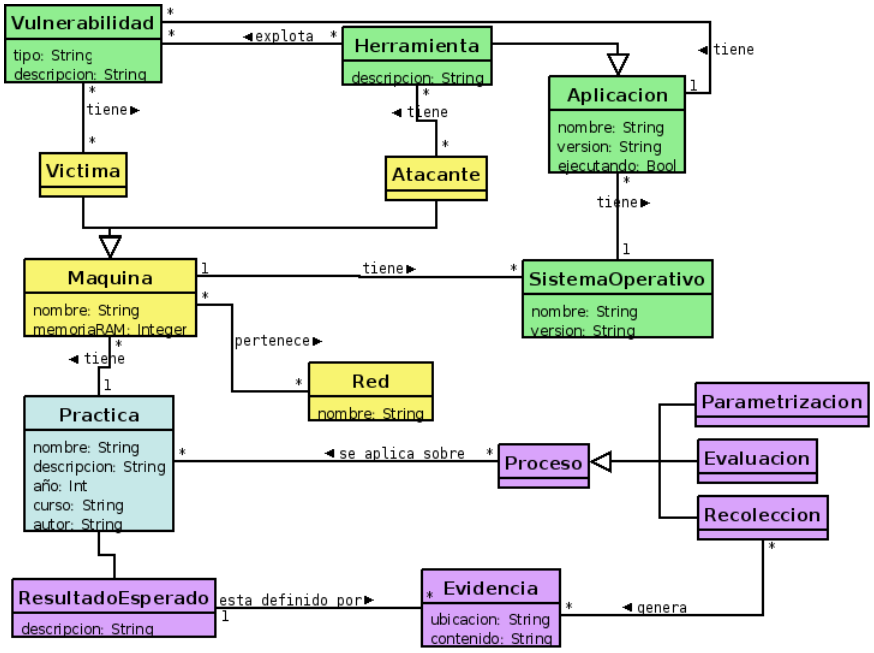

Figura 3. Algunos componentes del lenguaje

vulnerabilidades asociadas, y la necesidad de definir un mecanismo que asista en la evaluación de las prácticas.

Básicamente una especificación de escenario de experimentación en LISLab consiste en una definición de la estructura `Scenario` y definiciones de otras estructuras.

En la Figura 4 se muestra un fragmento de la gramática libre de contexto de LISLab, que reconoce la estructura *Tool* vista en el ejemplo.

4. Definición e Instanciación de escenarios

En la Sección "Metodología de trabajo" de [2] se describen un conjunto de componentes y procesos que son utilizados para definir los escenarios de experimentación.

Los componentes son esencialmente repositorios de máquinas virtuales y aplicaciones. Los procesos son planificación, preparación e instanciación de los escenarios. Si se piensa en evaluar las prácticas realizadas sobre los escenarios, existirían también procesos de evaluación.

Con estos antecedentes en mente, pensando en automatizar las tareas y utilizando el lenguaje LISLab, podemos inferir que:

- El resultado de la planificación es la especificación en LISLab del escenario planteado.

```
tool_def:  TOOL_TOK tool_id OPENBRACE
             NAME_TOK EQUAL string SEMICOLON
             VERSION_TOK EQUAL string SEMICOLON
             OPERATING_SYSTEM_ATTR_TOK EQUAL operating_system_id
                 SEMICOLON
             [RUNNING_TOK EQUAL BOOL SEMICOLON]
             [SERVICE_ATTR_TOK EQUAL service_id SEMICOLON]
             [VULNERABILITIES_TOK EQUAL OPENBRACE
                 list_vulnerability_id CLOSEBRACE SEMICOLON]
             [EXPLOITABLE_VULNERABILITIES_TOK EQUAL OPENBRACE
                 list_vulnerability_id CLOSEBRACE SEMICOLON]
           CLOSEBRACE SEMICOLON
```

Figura 4. Fragmento de la Especificación de la Gramática de LISLab

- Determinando en forma precisa cómo se manejan los repositorios y creando herramientas, se pueden automatizar los procesos de preparación e instanciación.
- Para la evaluación, se pueden definir procesos que recolecten información que permita verificar el cumplimiento de los objetivos de la práctica, por ejemplo: recolectar logs de sistema.

A continuación se describen los componentes de la herramienta de especificación de escenarios diseñada. Es pertinente aclarar que ya existe una primera versión implementada de dicha herramienta, la cual está siendo evaluada.

4.1. Componentes de la Herramienta

En la Figura 5 se muestra un diagrama de componentes de la herramienta y la interacción entre los mismos.

El *Compilador* toma una especificación de práctica en el lenguaje LISLab y genera una práctica que pasará a formar parte del repositorio de prácticas.

El *Repositorio de Prácticas* almacena las prácticas generadas por el compilador, de forma tal que puedan ser utilizadas posteriormente por el módulo de instanciación.

El *Repositorio de Imágenes de Disco* contiene las imágenes disponibles para la definición de prácticas. Las imágenes referenciadas en la especificación de la práctica son utilizadas por el módulo de instanciación.

El *Repositorio de Aplicaciones* contiene las aplicaciones disponibles para ser instaladas en las máquinas. Las aplicaciones referenciadas en la especificación de la práctica son utilizadas por el módulo de instanciación.

El *Módulo de Instanciación* es el 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.

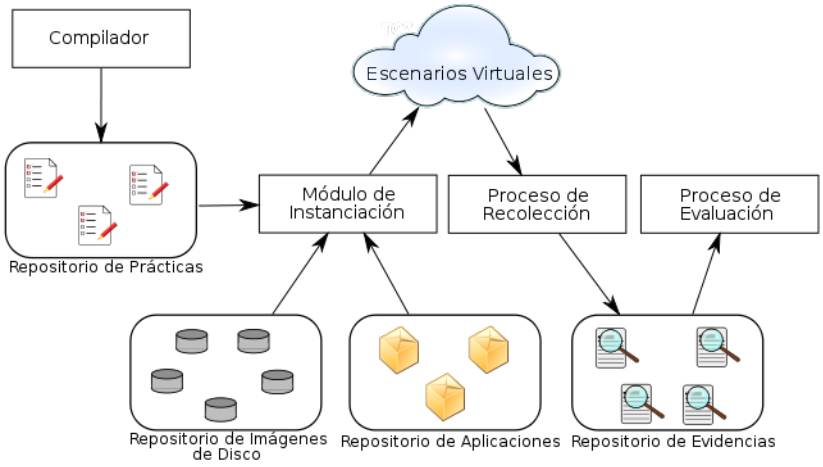

Figura 5. Componentes de la herramienta

Los *Escenarios Virtuales* son redes de máquinas virtuales que reflejan los escenarios definidos en la especificación de la práctica. Se genera uno de estos escenarios por cada participante que realice la práctica.

El *Proceso de Recolección* se encarga de obtener información de los escenarios virtuales, necesaria para evaluar la práctica, y almacenarla en el *Repositorio de Evidencias*. La información que debe obtenerse de los escenarios está definida en la especificación de la práctica.

El *Proceso de Evaluación* 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. Diseño del compilador

En la Figura 6 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 generen los componentes necesarios para poder instanciar una práctica. Es pertinente aclarar, que es en este punto que se produce la asociación con el producto de virtualización utilizado, ya que los componentes generados serán dependientes del producto de virtualización.

En la primera versión de la herramienta, se decidió utilizar UML [14] como producto de virtualización y VNUML [15] como herramienta de instanciación. Para ello se realizaron extensiones a dicha herramienta para soportar conceptos no implementados, como ser: creación de usuarios, instalación de aplicaciones y herramientas de seguridad, parametrización de las instancias y definición de reglas de firewall.

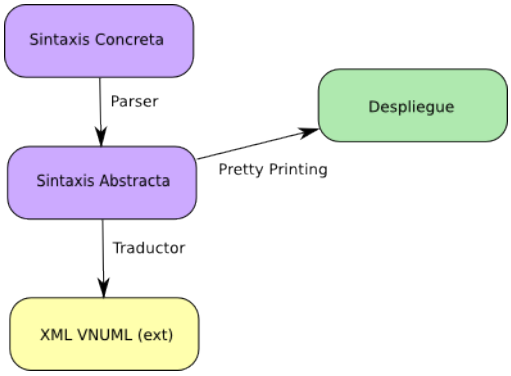

Figura 6. Módulos Involucrados en el Diseño del Compilador

El compilador consta de dos partes:

- *Frontend*
  El *frontend* está compuesto por un *parser*, cuya salida es una representación, en sintaxis abstracta, de una definición válida del lenguaje, a partir de una entrada en el lenguaje de sintaxis concreta. La sintaxis abstracta es, 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. Recordamos que para la primera versión de la herramienta, 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.

5. Discusión de una práctica: *Man in the Middle*

El objetivo de la práctica 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* [6].

En la Sección "Discusión de una Práctica: *Man in the Middle*" de [2] se describía el proceso para armar el escenario correspondiente. Dicho proceso se llevaba adelante manualmente. Para crear el atacante se tomaba, de un repositorio de máquinas base, una máquina virtual con sistema GNU/Linux, a la cual se le instalaban las herramientas necesarias. Las máquinas víctimas se generaban de manera similar. Luego se replicaba el escenario tantas veces como participantes realizaban la práctica. Los procesos de ejecutar y detener el escenario también eran manuales, teniendo que levantar o detener tres máquinas por cada instancia de escenario.

En contraposición, con la herramienta desarrollada, luego de que se tiene definido el escenario a un nivel conceptual, se procede a especificarlo en el lenguaje LISLab. Es importante destacar que dicha especificación se hace una única vez y luego se almacena en el *repositorio de prácticas*

```
Scenario mitm {
  name = "Man In The Middle";
  description =
   "A scenario to show
    a MitM attack";
  year = 2008;
  course = "FSI";
  attackers = {mitm_attacker};
  victims = {client_victim,
    server_victim};
};
Attacker mitm_attacker {
  name = "vm-att";
  ram = 256;
  tools = {ettercap};
};
Victim client_victim {
  name = "vm-cv";
  ram = 256;
  applications = {send_message};
};
Victim server_victim {
  name = "vm-sv";
  ram = 256;
  applications = {reply_message};
};
Network NMitm {
  range = range_mitm;
  switched = true;
};
Tool ettercap {
  name = "ettercap";
  version = "0.7.3-1.2";
  running = false;
};
```

Figura 7. Especificación de MitM en LISLab

En la Figura 7 se presenta un fragmento de la especificación en LISLab de la práctica. Esta especificación incluye la definición del escenario (*mitm*), la máquina atacante (*mitm_attacker*), la víctima servidor (*server_victim*), y la víctima cliente (*client_victim*). También se muestra la definición de la red switcheada utilizada en la práctica (*NMitm*). Por último se puede ver la definición de la herramienta *ettercap*, que será instalada en *mitm_attacker*.

Una vez especificado, se procede a compilar y el *módulo de instanciación* es quien se encarga de replicar los escenarios tantas veces como se le indique.

Para instanciar la práctica, se ejecuta el comando `start_scenario`, indicando el nombre de la práctica y la cantidad de instancias como parámetro.

En este momento, 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.

Para resumir, podemos destacar que con la herramienta se obtienen mejoras en el proceso de creación de escenarios, entre las cuales se destacan las siguientes:

- No se requiere conocimiento del producto de virtualización, o del sistema operativo utilizado al momento de definir las prácticas. Las diversas configuraciones de máquinas, redes, instalación de aplicaciones, definición de reglas de firewall, etc. se realizan de manera transparente al instanciar las prácticas.
- La replicación del escenario para cada participante se realiza de manera automática, disminuyendo el tiempo necesario para instanciar las prácticas.
- Las máquinas de las instancias pueden compartir la misma imagen base de disco, utilizando la tecnología de *Copy on write*.

6. Puesta en Producción

La herramienta se puso en producción en un curso en el cual se utilizaron 10 escenarios para una práctica. Cada escenario estaba compuesto de una máquina virtual que disponía de un servidor web con una aplicación vulnerable [1]. Las máquinas contaban con 750MB de espacio en disco y 256Mb de memoria RAM. El objetivo de la práctica consistía en explotar la vulnerabilidad y obtener una *shell reversa*[2].

Instancias anteriores de dicho laboratorio se ejecutaban sobre dos servidores Sun Fire X2100 x64 equipados con 2GB de memoria RAM, en la infraestructura que se describe en [2]. En este contexto, son necesarios 7.5GB de espacio en disco, y los servidores estaban destinados exclusivamente a la ejecución de las máquinas virtuales.

En contrapartida, utilizando la herramienta descripta en este trabajo, el laboratorio fue ejecutado en un *notebook* con 2GB de memoria RAM y con un procesador Core 2 Duo de 1.5Ghz. En esta experiencia se utilizó 900MB de disco físico en total.

Una particularidad a destacar de esta experiencia es que la utilización de la herramienta permitió que el curso, en particular el laboratorio, se pudiera dictar en aulas de la institución receptora. De esta manera, se pudo instrumentar una

[2] 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.

modalidad de dictado en la cual los participantes no tenían que trasladarse de su lugar de trabajo.

Finalmente podemos concluir que este ensayo permitió comprobar que la herramienta cumple con los requerimientos funcionales y de performance necesarios.

7. Conclusiones y Trabajo Futuro

Se implementó una primer versión de la herramienta para la definición e instanciación de escenarios.

En particular se destaca la creación del lenguaje LISLab, siendo éste uno de los resultados más relevantes del trabajo desarrollado. La definición de este lenguaje permite expresar los componentes y actores de los escenarios de seguridad, a un nivel de abstracción adecuado, y en gran medida de forma independiente a la tecnología utilizada.

Este lenguaje también actuará como interfaz al resto del *framework*, permitiendo a los demás módulos manejar conceptos abstractos, sin necesidad de conocer los detalles de implementación.

Esta primera versión de la herramienta permite confirmar la factibilidad de concebir un *framework* que facilite y automatice la creación de nuevos escenarios de experimentación para seguridad informática. También permite verificar la hipótesis de que utilizando otros paradigmas y productos de virtualización, se pueden optimizar los recursos.

Durante la fase de verificación de la herramienta, se vio que es posible migrar la mayoría de los escenarios utilizados en el laboratorio de los actuales cursos impartidos por el GSI, lo cual constituye un aporte importante.

A partir de este trabajo, surgen varias lineas de trabajo a futuro. Con respecto a la definición del lenguaje de especificación de escenarios, es factible extenderlo para:

- Formalizar la definición de vulnerabilidades.
  La especificación de las vulnerabilidades que se ilustran en las diferentes prácticas actualmente son simplemente un texto donde se las describe. Puede resultar de interés clasificar las vulnerabilidades y formalizar su especificación.
- Extender la definición de evidencias
  Las evidencias son un conjunto de elementos de las máquinas (*logs*, archivos, procesos) a analizar, cuyo cometido es verificar si el estudiante cumplió el propósito de la práctica. Visto de esta manera, el análisis podría servir como parte del mecanismo evaluador de la práctica.
  Se podrían utilizar lenguajes formales para la especificación de las evidencias que es necesario recolectar del escenario de la práctica. Durante el desarrollo de la primera versión de la herramienta, 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, en el caso de 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.
Esta es una línea de trabajo en la cual se podrian incorporar técnicas y herramientas de análisis forense informático.

- Implementar intérpretes para otras tecnologías de virtualización.
  Una limitación de la herramienta es que solamente se pueden especificar escenarios en los cuales se utiliza linux como sistema operativo. Intérpretes para otras tecnologías permitirían 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 y no hacer uso de VNUML.
- 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 al resto de la implementación, se podría:

- Desarrollar un módulo completo de instanciación de prácticas.
  En la primera versión de la herramienta, para simplificar, se utilizo una versión especialmente modificada de VNUML.
  Este módulo debería ser flexible y configurable. Esto se podría implementar definiendo un lenguaje de instanciación de escenarios, en el se 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, incluir una aplicación en la definición de la práctica requiere incluir 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 que hasta el momento no fueron implementados.
  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 se puede ver, esta primera versión 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 completa de la herramienta que permita su puesta en producción en laboratorios de seguridad informática.

Referencias

1. Awstats vulnerability. `http://www.securityfocus.com/bid/12543/exploit`, Último acceso: 25/04/09.
2. 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.
3. Cisco systems, inc. `http://www.cisco.com`, Último acceso: 21/04/09.
4. CyberCIEGE, An innovative video game and tool to teach network security concepts. `http://cisr.nps.edu/cyberciege/`, Último acceso: 21/04/09.
5. David Dellacca and Connie Justice. Building tomorrow's information assurance workforce through experiental learning. Technical report, Indiana University-Purdue University Indianapolis, 2007.
6. Ettercap. `http://ettercap.sourceforge.net`, Último acceso: 26/04/09.
7. Ji Hu, Dirk Cordel, and Christoph Meinel. A virtual machine architecture for creating IT-security labs. Technical report, Hasso-Plattner Institut, October 2006.
8. Itguru academic edition. `http://www.opnet.com/university_program/itguru_academic_edition/index.html`, Último accesso: 26/04/09.
9. Mod-X Security Challenge. `http://www.mod-x.co.uk`, Último acceso: 21/04/09.
10. Cisco packet tracer. `http://www.cisco.com/web/learning/netacad/course_catalog/PacketTracer.html`.
11. Paravirtualization is state of the art virtualization, xen. `http://www.xen.org/about/paravirtualization.html`, Último acceso: 26/04/09.
12. Sylvia Perez-Hardy. A unique experiential model for teaching network administration. Technical report, Rochester Institute of Technology, 2003.
13. Vassilis Prevelakis. Supporting a security laboratory. Technical report, Drexel University, 2007.
14. User Mode Linux. `http://uml.jfdi.org/uml/`, Último acceso: 25/04/09.
15. VNUML. Virtual Network User Mode Linux. `http://www.dit.upm.es/vnumlwiki/`, Último acceso: 25/04/09.
16. James Walden. A real-time information warfare exercise on a virtual network. Technical report, University of Toledo, 2005.
17. OWASP Webgoat Project. `http://www.owasp.org/index.php/Category:OWASP_WebGoat_Project`, Último acceso: 21/04/09.