Universidad Nacional de Asunción

Tesis de Maestría

Entorno para Experimentación de Vulnerabilidades en la Enseñanza de Buenas Prácticas de Programación

Autor:
Uri Pablo Yael Rozenworcel

Orientador:
D.Sc. Benjamín Barán

Una tesis presentada en cumplimiento de los requisitos
para el grado de Máster en Tecnologías de la Información y Comunicación

Diciembre 2015

Declaración de Autoría

Yo, Uri Pablo Yael Rozenworcel, declaro que esta tesis titulada "Entorno para Experimentación de Vulnerabilidades en la Enseñanza de Buenas Prácticas de Programación" y el trabajo presentado son de mi autoría.

Firma:

Fecha:

"La seguridad no es tan solo una materia en una malla curricular; es una perspectiva o enfoque desde donde deben observarse todos los contenidos. No puede agregarse seguridad como si fuera un módulo, sino que debe estar presente en cada etapa de un proyecto, sistema o emprendimiento."

Uri Pablo Yael Rozenworcel

UNIVERSIDAD NACIONAL DE ASUNCIÓN

Abstract

Facultad Politécnica
Laboratorio de Computación Científica y Aplicada

Máster en Tecnologías de la Información y Comunicación

Entorno para Experimentación de Vulnerabilidades en la Enseñanza de Buenas Prácticas de Programación

por Uri Pablo Yael Rozenworcel

This work provides an overview of methodologies and frameworks used in teaching concepts of information security on formal educational institutions, and proposes a framework focused on easiness of use, from the instructor's perspective, including eight expandable scenarios and experiences of usage in classroom.

Agradecimientos

Agradezco a mi familia, amigos y colegas quienes me dieron su apoyo durante todo el proceso; también a todos aquellos quienes de una forma u otra aportaron a mi formación profesional y académica.

Índice general

Índice de figuras

Índice de cuadros

Acrónimos y Siglas

ARP	*Address Resolution Protocol* (Protocolo de Resolución de Direcciones))
CWE	*Common Weakness Enumeration*
CSRF	*Cross-Site Request Forgery* (Falsificación de Petición en Sitios Cruzados)
CWSS	*Common Weakness Scoring System*
DNS	*Domain Name System* (Sistema de Nombres de Dominio)
DOM	*Document Object Model* (Modelo de Objetos del Documento)
DRY	*Don't repeat yourself* (No te repitas)
HTTP	*Hypertext Transfer Protocol*
IEEE	*Institute of Electrical and Electronics Engineers* (Instituto de Ingeniería Eléctrica y Electrónica)
LAN	*Local Area Network* (Red de Área Local)
OWASP	*Open Web Application Security Project* (Proyecto Abierto de Seguridad de Aplicaciones Web)
SANS	*SysAdmin Audit, Networking and Security Institute*
SQL	*Structured Query Language* (Lenguaje de Consulta Estructurado)
SWEBOK	*Software Engineering Body of Knowledge*
SWECC	*Software Engineering Coordinating Committee*
XSS	*Cross-Site Scripting* (Secuencias de Comandos en Sitios Cruzados)

Capítulo 1

Introducción

1.1. Contexto

Las necesidades y presiones del mercado hacen que en gran medida las empresas contraten a estudiantes que aún no han finalizado el proceso de aprendizaje de pre-grado [1]. Dichas necesidades y presiones llevan en ocasiones a que las universidades decidan descartar cierto contenido teórico de base para dar foco a las tecnologías más nuevas y a aquellas herramientas que los alumnos requieren para iniciarse en el ambiente laboral.

Los contenidos académicos omitidos en la malla curricular pueden no parecer imprescindibles inicialmente, pero sí suelen ser trascendentes cuando, por el avance tecnológico, los jóvenes ya iniciados en el ambiente laboral deben enfrentar un cambio de escenario. Las personas con bases teóricas pobres pueden entonces sufrir una especie de obsolescencia profesional, donde se ven faltos de capacidad en la medida que nuevos productos del mercado reemplazan a aquellos utilizados durante su período de formación académica, o sencillamente por un cambio de paradigmas.

Los cursos de seguridad de la información deben cubrir una vasta cantidad de áreas y temas, tales como: arquitecturas de las computadoras, aspectos legales relativos a la criminología y delitos informáticos, criptografía, base de datos, interacciones entre el humano y el computador, obtención de información, técnicas forenses, teoría de la información, administración y gestión, matemáticas, dispositivos móvil, redes de computadoras, sistemas operativos, aspectos éticos y filosóficos, lenguajes de programación, ingeniería de software, estadística, entre otros. Dicha amplitud impide ahondar suficientemente en

todas las áreas, por lo que muchos autores intentan confeccionar guías generales que puedan ser aplicadas en forma transversal, y que sean mayormente válidas para cada caso [1]. Es crucial seleccionar buenas técnicas pedagógicas para que los estudiantes puedan adquirir los conocimientos necesarios en un espacio de tiempo razonable.

Debido a todo el conocimiento de base requerido para adentrarse en seguridad informática, muchas universidades abordan la seguridad como foco central sólo en algunas materias electivas o en algunos módulos de otras materias hacia el final del programa. Así como el método científico es transversal a prácticamente toda actividad formativa en el área, la seguridad debería ser un eje temático en las carreras de grado a las que se está haciendo referencia. Los educadores y guías deberían enfocar los contenidos desde distintos ángulos, y uno de ellos debería ser la seguridad.

Debe diferenciarse entre, por un lado las 'cosas de seguridad' y por el otro las 'cosas seguras' o 'realizar las cosas en forma segura'. Según se indica en [2] el 84 % de los ataques apuntan a los programas, y no a la red o a los componentes *hardware* que están mejor protegidos por ser en general más robustos desde su diseño.

Desde que una persona empieza a aprender programación, debe tomar en cuenta ciertos conceptos y tenerlos presente en su vida cotidiana. Por ejemplo:

- en ocasiones algunos pequeños problemas terminan dañando en forma seria debido a no tratarlos en forma adecuada o en el momento adecuado;
- la gente inteligente puede cometer errores tontos;
- generalizando un poco, los programadores (y probablemente las personas en general) cometen el mismo error de seguridad una y otra vez;
- es necesario expresarse de forma tal que personas no expertas en el área puedan comprender el mensaje. Esto es especialmente importante en lo relacionado al uso de herramientas que bien aplicadas brindan seguridad y mal aplicadas pueden implicar vulnerabilidades.

En opinión de Jacob West[1] y Matt Bishop[2], existen diversos mitos por los cuales muchas instituciones educativas no aumentan los esfuerzos de incorporar en la práctica la enseñanza de codificación segura:

- no hay espacio en la malla curricular. No se necesitan más cursos, sino revisar el contenido de los ya existentes incluyendo el foco en la robustez;
- si los alumnos aprenden a escribir programas seguros, cuando salgan al mercado laboral ¿aceptarán las compañías el aumento del costo y tiempo que toma crear un programa?, ¿los clientes pagarían precios más altos?, ¿realmente se requerirá a los alumnos que pongan en práctica lo que aprendieron?;
- ya lo estamos haciendo. Salvo ciertas excepciones, en los centros de formación la evolución de los programas educativos y de las técnicas de enseñanza no es tan rápida como los avances en el campo de la seguridad de la información lo requiere. No es posible afirmar que los programas están adecuados, sin una validación constante del estado del arte en la materia.

1.2. Motivación del Trabajo

Al momento de crear sistemas informáticos, la seguridad debe ser un objetivo primario desde el inicio. En gran cantidad de casos, los sistemas que no tuvieron dicho objetivo ya en la etapa de diseño, deben luego pasar por una reestructuración y por diversos cambios, por supuesto que a un importante costo en recursos (e.g. tiempo de desarrollo, atrasos en fechas límite, dinero) [3].

La simplicidad del código rinde sus frutos ya que resulta más fácilmente mantenible y con menos probabilidad de errores por desatención del desarrollador. Los errores de código que pueden considerarse *bugs* clásicos, cuya detección puede ser automatizada y validada en sucesivas verificaciones, no son el único motivo por el que existen las vulnerabilidades. Otras fuentes de vulnerabilidades –que pueden derivar en huecos de seguridad– son los errores conceptuales en protocolos e interacciones, susceptibilidad a la falta de autenticación, autorización o de chequeo de integridad.

[1] Jacob West se desempeña actualmente como CTO (*Chief Technology Officer*) en HP Enterprise Security Products [2].

[2] Matt Bishop es un reconocido profesor en la Universidad de California [2] que, según su perfil en Google Scholar, a la fecha fue citado 3687 veces y desde el 2009 su índice H es de 26.

En cierto sentido, las técnicas de desarrollo de software generalmente enseñadas en los cursos son inadecuadas para crear programas seguros debido a que se centran en la correctitud. Una definición de correctitud es [3]: *cumplimiento de las especificaciones que determinan cómo los usuarios pueden interactuar con el sistema y cómo debe comportarse el sistema cuando es utilizado en forma correcta.*

Mientras la correctitud se encarga de las funcionalidades del sistema, la seguridad se encarga de la *falta de funcionalidades*. Es decir, se intenta validar que sin importar las acciones llevadas a cabo, no se podrá efectuar cierta acción. Las comprobaciones automatizadas son métodos para validar la existencia de una funcionalidad, pero por ahora no existe un método que asegure la inexistencia de una funcionalidad.

La inclusión de experiencias prácticas en el aula podría ayudar a los participantes a fijar los conceptos teóricos. Este trabajo está motivado por el hecho de que muchos instructores y profesores se dedican a la enseñanza como una actividad laboral relevante intelectualmente, pero no económicamente. Es decir que el tiempo dedicado a la preparación de los cursos en los que participan en general es reducido y que la adaptación, configuración y realización de experiencias prácticas excede al tiempo disponible. Además, otros aspectos a considerar en muchas instituciones son los requerimientos de equipos y configuración, no siempre disponibles.

1.3. Motivación de la Propuesta

En opinión del autor del presente trabajo, el efecto producido en los alumnos demuestra que es necesario alternar la estrategia pedagógica utilizada. El trabajo en grupo permite que los integrantes ordenen sus pensamientos para compartirlos y comparen sus conocimientos enfrentándolos a las diferentes propuestas que hagan otros integrantes. Limitar el tamaño del grupo a sólo dos personas parece tener mejores resultados que grupos mayores donde algunos integrantes toman una posición pasiva y hasta pierden interés ante unos pocos integrantes activos. En ocasiones los grupos parecen dividirse en dos equipos: los activos que realizan las tareas y se involucran, y los pasivos que se dejan llevar y no participan, por lo que tampoco aprenden.

Las dificultades indicadas en la sección 1.2 son en gran medida la causa de la poca cantidad de horas dedicadas a prácticas respecto a las dedicadas a la teoría. El autor de

este trabajo tiene la esperanza de que el costo de realizar ciertas actividades prácticas disminuirá notoriamente si se cuenta con un entorno de experimentación sencillo de administrar y utilizar. Dicha disminución permitirá mejorar el equilibrio entre actividades teóricas y prácticas.

1.4. Objetivos del Trabajo

El presente trabajo tiene como objetivo ofrecer una visión general de las metodologías y marcos utilizados en la enseñanza de conceptos de seguridad de la información en las instituciones de educación formal. Como resultado, se brinda una herramienta práctica que puede ser utilizada en clase.

Dados los beneficios reportados sobre la realización en clase de experiencias como laboratorios prácticos, y revisando las dificultades encontradas por los instructores en las distintas técnicas de enseñanza [1–6], la herramienta brindada se diferencia de otras en la simplicidad de preparación y administración del entorno, ya que según los datos recabados, la mayoría está de acuerdo en que estas experiencias son beneficiosas para los estudiantes, pero que los instructores no cuentan con recursos suficientes (*e.g.* tiempo, componentes) para llevarlas a cabo.

1.5. Organización del Documento

En el Capítulo 2 se presentan algunas técnicas de enseñanza con aspectos positivos y negativos, seguido de buenas prácticas y una descripción de ciertas características de los estudiantes. Luego, el Capítulo 3 presenta herramientas disponibles, institutos y organizaciones, guías, *rankings* y entornos de trabajo. El entorno propuesto en el presente trabajo se encuentra en el Capítulo 4 donde se describen los componentes y escenarios. Finalmente, el Capítulo 5 relata el uso ya realizado del entorno propuesto mientras que el Capítulo 6 presenta las conclusiones a modo de discusión final.

Capítulo 2

Aprendiendo a Programar

2.1. Técnicas de Enseñanza

A continuación se resumen algunas observaciones sobre distintas técnicas o enfoques, entresacadas de [1, 7, 8], utilizadas en la lista de cursos sobre seguridad y criptografía conformada por Aviel Rubin[1], y la lista sobre computación y seguridad de la información creada por Heather Hinton[2].

Enfoque de clase tradicional pasiva. La técnica es considerada pasiva ya que el educando sólo recibe sin mayor interactividad y sin gran incidencia sobre el proceso mental a través del que incorpora la información. Esta metodología parece ser la más utilizada en la actualidad y resulta beneficiosa cuando el contenido está conformado por bases teóricas, cuando se desea mayor cobertura en amplitud que en profundidad, y cuando provee las bases teóricas necesarias para otros cursos posteriores donde se ahonde utilizando otras técnicas. Algunas técnicas que pueden utilizarse en el enfoque de clase tradicional incluyen el basarse en ejemplos prácticos, partir de descripciones informales para arribar a definiciones formales, realizar ejercicios que resulten intrigantes a los alumnos como descifrar un código criptográfico. Algunos riesgos latentes son que algunos alumnos se vuelvan extremadamente pasivos, tengan baja participación, o que no realicen los esfuerzos suficientes para internalizar un material complejo.

[1] http://cs.nyu.edu/cs/dept_info/course_home_pages/fall96/G22.3033.02.
[2] http://www.ee.ryerson.ca:8080/~hhinton/compsec/security.html.

Enfoque de escriba. La técnica puede ser puesta en práctica partiendo el contenido y asignando distintas secciones a distintos alumnos o grupos, pidiendo que los mismos tomen nota de los conceptos clave para luego preparar una presentación que será expuesta en clase o se encontrará disponible en un sitio web. Este enfoque posiciona a los participantes en forma más activa, y abre lugar a la inclusión de aspectos complementarios propuestos por los mismos alumnos y validados o corregidos luego por el instructor. Un potencial riesgo es que muchos participantes se encuentren tan presionados por su asignación que no den la suficiente atención a los contenidos que no le fueron asignados. Esta técnica, mal conducida, puede ser contraproducente malgastando valiosas horas y hasta disminuyendo la calidad del contenido transmitido siendo que la producción de los estudiantes es incorporada al material del curso.

Enfoque experto/mentor. En algunas situaciones la experiencia educativa puede ser expandida y mejorada contando con múltiples instructores, cada uno abordando su propia especialidad. De esta forma, el encargado del curso toma un papel de guía y coordinador, concentrando su esfuerzo en la coherencia de los contenidos a través de enfatizar los puntos de encuentro y complementando cuando es necesario. El rol del encargado del curso es crítico ya que es quien debe lograr compatibilidad y coherencia entre los distintos instructores, por lo que un mal desempeño de la función puede resultar en repetición de contenidos, contradicciones y dificultades al momento de evaluar a los alumnos.

Enfoque de tutoría. Esta técnica se centra en el uso de material con énfasis en metodologías autodidactas, y principalmente es utilizada en cursos donde el participante busca una certificación. Un riesgo latente es la utilización de material buscado por el participante en Internet, cuyo contenido no sea correcto o preciso y que lleva no sólo a una pérdida de tiempo, sino también a la tarea de desaprender algo para reaprenderlo luego en forma correcta.

Enfoque de proyecto. Otras técnicas pueden incluir componentes de este enfoque donde el participante debe involucrarse o crear un proyecto experimental, monografía o una presentación. Es aconsejable que las actividades sean prácticas, lo cual potencia aspectos positivos de este enfoque como la aplicación conjugada de conocimientos adquiridos en diversas áreas, el espacio disponible para la creatividad y la incorporación y fijación de conocimientos por medio de experiencias personales. Puede considerarse como aspecto negativo que el estudiante debe tener fuertes conocimientos de base, sobre

los que se desarrollan las actividades. A modo de ejemplo, muchos cursos proponen la creación de un sistema de mensajería con diversas variantes, para lo cual deben tenerse ya conocimientos sobre redes, seguridad, programación, estructura de datos, entre otros.

Enfoque de sinergia entre investigación y enseñanza. Esta técnica intercala en forma constante las clases en su forma tradicional, con experiencias investigativas realizadas por los participantes permitiendo mantener un orden coherente en los contenidos y validar los resultados en forma constante, mientras se da oportunidad para adentrarse en los últimos adelantos del área estudiada.

Enfoque de laboratorios de ataque y defensa aislados. La diferencia entre estos laboratorios y los ya mencionados en otros enfoques, es que aquí se hace una aislación entre dos equipos: atacantes y defensores. Los primeros tienen como objetivo vulnerar equipos que están siendo administrador y monitoreados por los segundos. El valor de mantener el aislamiento es representar la realidad con mayor fidelidad, incentivar la creatividad así como la propuesta de estrategias. Como aspectos negativos de este enfoque pueden resaltarse el costo en recursos y esfuerzo para crear y mantener laboratorios completos aislados de otras redes para garantizar que ningún ataque cause un daño a componentes externos y que no haya accidentalmente fuga de información.

El siguiente cuadro resume los enfoques descritos anteriormente:

Técnica	**Características**
Enfoque de clase tradicional pasiva	▪ Comunicación en una sola vía: del instructor al estudiante ▪ Poco espacio para diversidad de procesos de aprendizaje ▪ Técnica más utilizada en la actualidad

Enfoque de escriba	▪ Los alumnos presentan el contenido que les fue asignado ▪ Participación activa por lapsos de tiempo ▪ Permite la adición de materiales no contemplados por el instructor ▪ Esta técnica mal conducida puede ser más perjudicial que valiosa (*e.g.* material de baja calidad, desperdicio de tiempo, explicaciones deficientes)
Enfoque experto/mentor	▪ Varios instructores, cada uno con su especialidad ▪ Encargado del curso toma el rol de guía y coordinador ▪ Riesgo de falta de coherencia entre contenidos
Enfoque de tutoría	▪ Uso de metodologías autodidactas ▪ Muy utilizado en certificaciones ▪ Riesgo de malas interpretaciones

Enfoque de proyecto	▪ Puede ser un complemento a otros enfoques ▪ Permite incluir experiencias más vivenciales ▪ Logra conjugar conocimientos de distintas áreas ▪ Para que sea efectivo y viable en el tiempo, el participante ya debe tener cierto conocimiento de base
Enfoque de sinergia entre investigación y enseñanza	▪ El estudiante se sitúa en una posición más activa ▪ Puede ser un complemento a otros enfoques ▪ Intercala clases tradicionales con otras metodologías ▪ Motiva la creatividad personal
Enfoque de laboratorios de ataque y defensa aislados	▪ Método esencialmente activo ▪ Promueve trabajo colaborativo compartiendo así los conocimientos ▪ Puede requerir mayor cantidad de recursos y preparación que los otros enfoques ▪ Se requiere cuidados para no dañar componentes ajenos a las prácticas

Cuadro 2.1: Técnicas de enseñanza

2.2. Buenas Prácticas

El concepto de buenas prácticas debe ser entendido como un conjunto de sugerencias relacionadas a cómo realizar las etapas de los procesos (*e.g.* ciclo de vida del *software*), normas de cómo escribir código fuente, entre otros aspectos [9]. No existe una definición formal de cuáles son exactamente las buenas prácticas, ni cuándo algo deja de ser un consejo particular para convertirse en una buena práctica. No obstante, puede decirse que todo consejo que se considere buena práctica, debe ser aplicable a un amplio espectro de escenarios y situaciones; finalmente y desde un punto de vista empírico, es probable que una aplicación tenga menos vulnerabilidades que otra semejante, si los programadores adoptaron buenas prácticas en la segunda y no en la primera. En general las buenas prácticas tienen mucho de sentido común.

Si bien este trabajo no se centra en presentar una taxonomía de vulnerabilidades ni sus tratamientos, es conveniente resaltar algunas características que permiten realizar agrupamientos para tratar las distintas clases de problema.

Muchos lenguajes de programación ofrecen librerías o funciones específicas para evitar las vulnerabilidades más comunes como inyección de código y manejo de sesiones. A modo de ejemplo, el lenguaje PHP provee el paquete denominado *Magic Quotes*[3] que, aunque muy utilizado anteriormente, fue declarado obsoleto a partir de la versión 5.3.0 y eliminado a partir de la versión 5.4.0. Dicho paquete era utilizado para manipular caracteres especiales de forma a aumentar considerablemente la complejidad de realizar inyecciones *Structured Query Language* (SQL) con los mismos; no obstante, el programador puede cometer ciertos descuidos que permitan la inyección aunque el paquete esté en uso.

Uno de los principales métodos de prevención ante ataques de inyección de SQL es utilizar mecanismos que impliquen separación entre código fuente y datos. Estos mecanismos pueden proveer las validaciones y acciones necesarias en vez de basarse en lo realizado por el programador. La centralización de estos mecanismos permite que el mejoramiento de un único lugar repercuta en gran cantidad de módulos, por ejemplo, teniendo una única función que trate (realice el escapado de) los caracteres especiales.

[3] El manual del paquete puede encontrarse en `http://php.net/manual/es/security.magicquotes.php`.

Otro mecanismo es utilizar llamadas parametrizadas o procedimientos almacenados que soporten tipado fuerte[4], en vez de ejecutar consultas construidas en forma dinámica o con comandos que ejecutan código dinámico ya sea en el programa navegador del usuario, en el sistema operativo o en el motor de base de datos.

Suponiendo que un atacante logre inyectar código, la capacidad del mismo se ve caracterizada principalmente por los privilegios con los que se ejecuten los comandos. A modo de ejemplo, si explotando una vulnerabilidad de una aplicación se logra inyectar código SQL para borrar una tabla, el ataque será efectivo sólo si el usuario con el que la aplicación se conecta a la base de datos tiene permiso de borrado sobre dicha tabla. Por ello, se aconseja ejecutar el código utilizando los mínimos permisos requeridos para cumplir la tarea. Para esto, puede ser necesaria la utilización de distintos usuarios, cada uno con un conjunto distinto de permisos, lo que podría disminuir el daño ocasionado por un ataque exitoso [10].

El diseño de un sistema establece ciertas interacciones entre elementos y algunos procesos requieren cierto orden en la ejecución de algunas acciones. Los análisis de vulnerabilidad deben abarcar interacciones que, aunque son posibles, no están incluidas en el flujo normal de funcionamiento. Esto es especialmente crítico en lo que se refiere a los controles y validaciones en esquemas cliente–servidor.

En [10] se hace especial hincapié en realizar las validaciones del lado del servidor, aún cuando algunas o todas esas validaciones se realizan también en el cliente. Debe considerarse que existe la posibilidad de que un atacante logre evitar los controles en el cliente o modificar los datos en un momento posterior a dichos controles. En general, la validación de entrada se realiza contra una lista negra de caracteres o patrones que deben ser evitados. Otro método de validación que resulta más seguro, pero también mucho más costoso de implementar, es realizar la validación contra una lista blanca permitiendo sólo lo que se compadece con la lista y rechazando el resto.

Validar la salida de datos es igual o más importante que validar su entrada, según sea el escenario específico. Por ejemplo, la validación de salida puede detener la exposición de los nombres de usuario cuando se esperaba obtener un número como resultado. Por ello, los mensajes de error deben brindar al usuario la mínima cantidad de información

[4]Un lenguaje de programación es fuertemente tipado cuando se realiza un control del tipo de dato según fue declarado. Los parámetros de las funciones y los operandos de las operaciones deben coincidir con el tipo exacto que se espera, o bien debe realizarse una conversión de tipo de dato.

útil, sin revelar en lo posible datos que permitan mejorar un ataque. Los detalles deben ser registrados en bitácoras siendo estas sólo accesibles a quienes corresponda. Las contraseñas no deben ser registradas en las bitácoras.

En lo que respecta a inyección de código, se recomienda utilizar en lo posible llamadas a librería antes de considerar procesos externos. El código debe ejecutarse en una jaula o caja de arena (*sandbox*) que ponga límites a los recursos disponibles. Prácticamente todas las prevenciones respecto a inyección de SQL son válidas para evitar inyección de código, siempre que se realicen las adaptaciones necesarias. La construcción de una consulta al motor de base de datos es similar a la construcción de parámetros que serán pasados a uno o más comandos.

La clasificación realizada por *Common Weakness Enumeration* (CWE)/*SysAdmin Audit, Networking and Security Institute* (SANS) de errores más peligrosos, correspondiente al año 2011, pone en tercer lugar al desbordamiento de memoria (*buffer overflow*) debido a operaciones de copiado que no controlan el tamaño de la entrada. Algunos lenguajes (*e.g.* Ada, C#) proveen protección contra *buffer overflow*, aunque los controles pueden ser desactivados por el programador.

Realizar controles del tamaño de entrada no garantiza inmunidad ante este tipo de ataques. Es muy importante realizar los controles en los lugares indicados. Es posible que un vector de ataque incluya una expansión o modificación de la entrada como resultado de algún proceso intermedio. A modo de ejemplo puede considerarse un proceso en el que se recibe el nombre de un archivo a ser abierto para copiar su contenido; el control es realizado sobre el tamaño del archivo, pero una vulnerabilidad podría expandir la entrada concatenando múltiples veces el mismo archivo, o un flujo arbitrario de caracteres generados.

En el cuarto puesto de la referida clasificación se encuentran los errores de neutralización de los datos de entrada en la generación dinámica de contenido web. Este tipo de ataques es denominado *Cross-Site Scripting* (XSS). Puede inyectarse código que será ejecutado en el navegador web de un visitante en forma análoga a como se inyectan comandos de SQL o de código. Esta técnica es muy potente ya que no sólo el visitante puede convertirse en víctima, sino que además puede convertirse en atacante sin saberlo, ejecutando código que forma parte de ataques a terceros. Algunas librerías que pueden ser utilizadas para

evitar vectores de ataques con XSS son Anti-XSS de Microsoft[5], el módulo de codificación ESAPI de OWASP[6] y Wicket de Apache[7].

Existen algunas opciones de configuración propuestas que resultan en un aumento de complejidad considerable para que los vectores de ataque XSS sean efectivos. Una de las principales propuestas es la política de mismo origen (*same origin policy*). Esta política permite el acceso al *Document Object Model* (DOM) sin restricciones específicas entre distintos scripts originados en el mismo sitio. La definición de sitio está dada por la combinación de esquema (*e.g.* http, https), nombre del servidor y número de puerto. La política del mismo origen se aplica a XMLHttpRequest[8], teniendo en cuenta la cabecera de control de acceso denominada *Access-Control-Allow-Origin* que opcionalmente puede ser provista. La cabecera *Access-Control-Allow-Origin* permite el acceso a los orígenes definidos en la misma y lo prohíbe a los demás orígenes. Si bien la política de mismo origen es una herramienta potente, aún no se encuentra implementada por todas las tecnologías que se basan en protocolo HTTP ni por todos los programas navegadores. Algo similar ocurre con la bandera HttpOnly que puede ser indicada en los archivos denominados *cookies*, que no está implementada en todos los navegadores.

Distintos entornos de programación web (e.g. Django[9], zend[10]) facilitan desde el diseño la separación entre código fuente y datos, de forma a poder realizar controles más precisos y evitar que los datos se conviertan en código ejecutable. Aún cuando se utilicen entornos de programación que fomenten el buen diseño, un programador descuidado puede escribir código vulnerable a la modificación del código ejecutable a partir de los datos inyectados.

En el quinto lugar de la clasificación publicada por CWE/SANS en 2011 figura la falta de autenticación para funciones críticas. Como se comenta en dicho documento, los atacantes no tienen por qué necesariamente ingresar por la puerta principal ya que pueden encontrarse muchos otros puntos de acceso que pueden no parecer obvios a los creadores del sistema ni a quienes hayan diseñado los sistemas de seguridad del mismo.

[5] Más información en `http://msdn.microsoft.com/en-us/security/aa973814.aspx`.
[6] Más información en `https://www.owasp.org/index.php/Category:OWASP_Enterprise_Security_API/es`.
[7] Más información en `https://wicket.apache.org`.
[8] Más información en los sitios para el desarrollador de Mozilla `https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy` o Chrome `https://developer.chrome.com/extensions/xhr`.
[9] Más información en `https://www.djangoproject.com`.
[10] Más información en `http://framework.zend.com`.

El sistema debe ser dividido en áreas: anónima, normal, privilegiada y de administración. Cada una de esas categorías podría requerir distintos métodos y o momentos de autenticación. Es útil listar todos los canales de comunicación y describir los mecanismos aplicados a cada canal. Las interacciones entre módulos a través de los distintos canales pueden llevar a vectores de ataque exitosos evitando completar exitosamente las autenticaciones correspondientes. Al igual que con los errores más peligrosos hasta aquí descritos, las validaciones deben hacerse en el servidor aún cuando ya hayan sido realizadas en el cliente debido a que no debe confiarse en que no hayan sido vulneradas desde el lado del cliente. De igual forma, deben utilizarse las librerías provistas por el lenguaje utilizado y evitarse, hasta donde sea posible, la ejecución de código externo.

2.3. El Estudiante y sus Habilidades

Las personas aprenden a través de distintos procesos. La estrategia de aprendizaje o de enseñanza debe ser seleccionada teniendo en cuenta tanto los contenidos como el público destinatario. En [4] se presenta un enfoque para realzar las buenas prácticas y estilo de programación a lo largo de una malla curricular. Según los autores de dicho documento, la técnica es también utilizada por escuelas de abogados para enseñar escritura legal.

Por diversos motivos las materias introductorias abordan la robustez y programación segura, pero al momento de cursar materias avanzadas, estos tópicos no son profundizados y por lo tanto los estudiantes no los incorporan efectivamente dentro de sus habilidades. Los motivos más comunes son falta de tiempo o de recursos; esto en general es debido a la gran extensión de materiales que deben ser cubiertos.

Los trabajos de programación en ambientes educativos parecen tener como principal objetivo ‘que funcione’; el problema es cómo se demuestra o prueba que algo funciona. En ocasiones basta con ejecutar el mismo programa en una computadora distinta o introducir valores de usuario que no se ajustan a lo esperado para que el programa ya no funcione como se esperaba. Esto es debido a que la sentencia de que el programa funciona es verdadera en algunos escenarios y para ciertos universos de datos, donde algunas fallas de programación permanezcan ocultas debido a que la máquina no ejecutó algunas sentencias específicas.

En la definición de un ejercicio o práctica se establecen los objetivos que serán evaluados. Estos objetivos responden a las áreas centrales del curso y no necesariamente a la buena codificación. Algunas características están implícitamente incluidas, pero no se establecen como objetivo: validación de datos de entrada, tratamiento de excepciones, mensajes de error, entre otros. Por ello puede considerarse que no se logra desarrollar en el estudiante las habilidades necesarias para que el mismo diseñe y programe sistemas seguros.

Como se propone en [4], podría resultar beneficioso incorporar objetivos secundarios, que no necesariamente responden a los fines específicos del curso. Para ello puede crearse una lista de buenas prácticas y estilos que será válida a lo largo de cada curso. Es factible adoptar alguna lista sugerida ya existente o bien basarse en las estadísticas de errores más frecuentes para elaborar prácticas que los eviten. No es necesario incluir en cada trabajo o ejercicio todos los puntos de la lista definida, pero sí es recomendable indicar un conjunto cada vez mayor durante el desarrollo del curso. De esta forma, los alumnos se habituarán a considerar aspectos relativos a la robustez y seguridad como requerimientos implícitos, sin que formen parte de los objetivos particulares del curso. A medida que el alumno avanza en la malla curricular desarrollará sus habilidades teniendo siempre presente los aspectos de seguridad estudiados, sin haber requerido cursar materias adicionales.

Una propuesta es diferenciar entre dos clases de características que se encuentran en lo que se denomina un 'mal programa'. Estas son: programación robusta y programación segura. Es conveniente limitar lo referido por programación robusta como 'que el programa actúe en forma correcta si es posible, y caso contrario termine con un error elegante'. Para ello se dan las siguientes definiciones:

- *Paranoia*: el programa no debe confiar en nada que no haya generado él mismo.
- *Estupidez*: presumir que el usuario no respetará ningún manual, formatos ni longitudes de entrada de datos.
- *Implementaciones peligrosas*: mantener ocultas las funciones y estructuras de datos internas para que los usuarios no puedan luego modificarlas o invocarlas erróneamente.
- *No puede suceder*: manejar situaciones que parezcan imposibles porque aunque inicialmente lo sean, modificaciones posteriores podrían hacerlas posibles.

De igual forma, conviene limitar el significado de programación segura, y una posibilidad es restringirlo a que el programa no agregará o eliminará privilegios o información a menos que tal cuestión sea su función específica.

2.4. Algunas Experiencias

Las diversas técnicas de enseñanza resumidas en el Capítulo 2.1 pueden ser aplicadas de distintas formas. En [11–14] se indican algunos aspectos positivos del uso de juegos en la enseñanza, y en [15] se presenta un juego en particular, cuyos autores catalogan de herramienta flexible y altamente interactiva que puede ser utilizada por instituciones para la concienciación sobre la seguridad. Dicho juego, llamado CyberCIEGE, tiene una versión lanzada en el año 2005 por la Escuela Naval de Postgrado del gobierno de los Estados Unidos[11].

El entorno de experimentación presentado en este trabajo no fue conceptualizado como un juego de video con una historia o línea narrativa, sin embargo podría ser beneficioso enmarcar su uso en actividades competitivas entre grupos de participantes contabilizando el tiempo que requiere a cada grupo hallar una o más soluciones a los distintos escenarios (ver Capítulo 4.1.1).

Los autores del entorno propuesto en este documento sugieren que los participantes deban elaborar un documento técnico una vez finalizada la experimentación. La confección del documento permitirá que los participantes ahonden en algunos conocimientos y expliciten el proceso que realizaron en cada caso. A continuación se ejemplifican algunos de los puntos que podrían solicitarse por cada escenario, independientemente de que el participante haya logrado o no superar con éxito el nivel sobre el que está escribiendo en el informe.

- Indicar el objetivo o condición de éxito del escenario.
- Explicar los intentos realizados indicando para cada uno si fue o no exitoso.
- Comentar qué se cree está sucediendo del lado del servidor, por lo que cada intento funciona o no.

[11] Más información en `http://cisr.nps.edu/cyberciege/news.html`.

- Citar las fuentes consultadas, si las hubo.
- Describir algunas precauciones que puede tomar el programador para minimizar o erradicar la vulnerabilidad.

En el Capítulo 5 se describen algunas experiencias ya realizadas entre los años 2013 y 2015[12] utilizando el entorno desarrollado para este trabajo en el estado de evolución que se encontraba en cada una de las oportunidades, e incluyendo los escenarios 1 a 5 en los primeros dos años y 6 a 8 en el año 2015. Cabe destacar desde ya que de todos los participantes que realizaron una encuesta voluntaria posterior, casi la totalidad de los encuestados indicó que la actividad dio la oportunidad de poner en práctica conceptos teóricos aprendidos en distintas materias. Dos preguntas en particular permiten darse una idea de la incidencia de la enseñanza de seguridad de la información en las mallas curriculares: la primera consultaba cuántas materias el encuestado ya cursó, y la segunda consultaba cuántas de dichas materias incluían conceptos de seguridad de la información. Tomando en cuenta las experiencias en los años 2013 y 2014, en promedio, los encuestados dijeron haber cursado 42 materias y que sólo 3 (incluyendo la materia en la que se desarrolló el laboratorio aquí descrito) tratan temas de seguridad de la información ya sea en forma práctica o teórica; esto es, poco más del 7 %.

[12] Las experiencias fueron realizadas en el ámbito de la cátedra Redes de Computadoras 2 y participaron algunos alumnos que cursan la carrera Ingeniería Informática o Ingeniería Electrónica dictadas en la Universidad Católica "Nuestra Señora de la Asunción".

Capítulo 3

Herramientas Disponibles

3.1. Institutos y Organizaciones

Existen diversos grupos de trabajo que aportan herramientas para facilitar la inclusión de mejoras a los sistemas en lo que respecta a la seguridad de la información. Las herramientas pueden ser librerías completas, entornos de comprobación, análisis y estadísticas o bien recopilaciones sobre sugerencias útiles.

SysAdmin Audit, Networking and Security Institute (SANS) es una organización con fines de lucro que agrupa a profesionales de la seguridad informática. Fue fundada en 1989 y sus principales objetivos son recopilar información referente a la seguridad de la información, y ofrecer capacitación y certificación en el ámbito de la seguridad informática. El sitio web del instituto es `http://www.sans.org`.

Common Weakness Enumeration (CWE) es un proyecto de la comunidad de *software* que tiene como objetivo la creación de un catálogo de debilidades y vulnerabilidades de *software*. El objetivo del proyecto es comprender mejor los defectos en el *software* y crear herramientas automáticas que pueden ser utilizados para identificar, corregir y prevenir estos defectos. El proyecto está patrocinado por The MITRE Corporation, que es una organización sin fines de lucro creada en 1958 y desde entonces sirve de referente a proyectos y fuerzas militares de los Estados Unidos.

Open Web Application Security Project (OWASP) es una organización abierta que mantiene canales de comunicación donde los interesados pueden conocer las novedades sobre

eventos, proyectos y redes de personas relacionadas a distintas áreas de la seguridad informática. La comunidad genera contenidos teóricos y aporta herramientas o mejoras a módulos existentes con la finalidad de compartir sus conocimientos y experiencias.

Algunas empresas ofrecen servicios de consultoría y auditoría en seguridad o dictan cursos tanto presenciales como a distancia, y entre sus insumos principales se encuentran distintas herramientas gratuitas –muchas de las cuales son presentadas en la Sección 3.4. Por ello, algunas de estas empresas participan en forma activa en el desarrollo, mantenimiento y mejoras de dichas herramientas ya que les permite adquirir un mejor entendimiento de su funcionamiento y adaptarlas a sus necesidades particulares además de contar con lo último en lo que a vectores de ataque y vulnerabilidades se refiere. Existe también gran colaboración proveniente de grupos de investigación de diversas universidades.

3.2. Guías

Las guías son un compendio de reglas y normas acordadas para facilitar diversos aspectos como el ordenamiento de la información, comprender trabajos de terceros, facilitar los procesos de comprobación y validación, entre otros. Existen autores que comparten su conocimiento relatando su experiencia profesional y presentando en forma estructurada las metodologías que les resultaron más útiles en sus casos particulares; aunque muchas de esas publicaciones no pueden considerarse científicas, las guías que aportan son un aporte importante. Este el el caso de [16] donde Peter Kim relata el uso que da a distintas herramientas para conducir un test de penetración.

El *Software Engineering Body of Knowledge* (SWEBOK) [17] es un documento promovido por la *Institute of Electrical and Electronics Engineers* (IEEE) y creado por la *Software Engineering Coordinating Committee* (SWECC). Dicho documento es considerado como una guía al conocimiento presente en el área de la Ingeniería del Software y supone un paso esencial hacia el desarrollo de la profesión porque representa un amplio consenso respecto a los contenidos de la disciplina.

Las quince áreas de conocimiento en que se divide SWEBOK son [17]: requerimientos, diseño, construcción, comprobación, mantenimiento, gestión de la configuración, gestión

de la ingeniería de software, proceso, herramientas y métodos, calidad, prácticas profesionales, economía de la ingeniería de software, fundamentos de la computación, fundamentos matemáticos y fundamentos de la ingeniería. Como medida del crecimiento de SWEBOK cabe destacarse que la versión 2 planteaba la división en 10 áreas mientras que la versión 3[1] lo hace en 15 áreas.

Por otro lado, SWEBOK propone, en su tercera versión, las siguientes ocho disciplinas que guardan relación directa: ingeniería informática, ciencias de la computación, administración general, matemáticas, gestión de proyectos, gestión de calidad e ingeniería de sistemas. En comparación con la versión anterior, fue omitida la disciplina 'ergonomía de software' que aborda aspectos de amigabilidad y diseño en las interfaces conjugado con las limitaciones y habilidades del ser humano en lo relativo al aspecto físico.

De entre los cursos de ingeniería de software dictados en las distintas universidades, sólo unos pocos abordan la seguridad y lo hacen en forma escueta en sólo algunas de las fases del ciclo de vida del software. Aunque en SWEBOK no se da un énfasis a la seguridad, sí se brindan muchas de las prácticas que derivan en sistemas seguros. En [5] se propone un curso de ingeniería de software desde una perspectiva de seguridad con una duración de 14 semanas. El material del curso deriva completamente de la guía SWEBOK a través de la posibilidad de dar distintas prácticas planteadas en la guía con un significado más aplicado a la seguridad.

En la librería para el desarrollador Mac [18] puede encontrarse que la misma es una guía de programación segura con taxonomías, definiciones y consejos. Según se establece en dicho documento, la codificación segura es la práctica de escribir programas resistentes a ataques maliciosos o al mal comportamiento de las personas o programas. Cabe entonces cuestionarse si es posible que cierto código considerado seguro, pueda convertirse con el tiempo en inseguro, por ejemplo, por cambios en componentes externos al mismo. Revisando la definición debe entenderse que un código seguro no puede convertirse en uno no seguro, aunque sí puede darse el caso de que un código inseguro haya sido considerado erróneamente como seguro.

El supuesto de que una función es segura se basa en que las acciones realizadas por la misma son seguras en forma atómica (*i.e.* cada una de ellas por separado). Distintos equipos de trabajo publican nuevas vulnerabilidades encontradas en componentes claves

[1] La tercera versión es hasta la fecha la más actual.

de algunos sistemas operativos, protocolos o librerías críticas. Aún cuando un programador haya sido cuidadoso intentando codificar en forma segura, debe considerar que cierto módulo o trozo es inseguro si utiliza elementos que (en forma más o menos novedosa) son vulnerables.

Por lo explicado anteriormente, un módulo que no impide el aprovechamiento de una vulnerabilidad existente en algún módulo utilizado internamente, debe ser catalogado como no seguro.

Según los creadores de la guía para codificación segura [18] que fue publicada en noviembre de 2014, la mayoría de las vulnerabilidades de seguridad se encuentra en una de las siguientes categorías: desbordamiento de pila (*buffer overflows*), no validación de datos de ingreso (*unvalidated input*), las condiciones de carrera (*race conditions*), problemas de control de acceso (*access-control problems*), y debilidades en la autenticación, autorización o encriptación.

En [18] se propone una lista de aspectos relacionados a la seguridad en el desarrollo (Apéndice A de la referencia), y una lista de guías de seguridad para la utilización de programas o componentes de terceros (Apéndice B de la referencia).

Categoría	**Descripción**
Buffer overflow	Ocurre cuando se intenta escribir datos más allá del límite (tanto inicial como final) de la memoria asignada o disponible. En general, este tipo de error permite la sobre-escritura de datos y ser el disparador de un vector de ataque que implica elevación de privilegios.

Validación de datos de entrada	Todo programa que permite al usuario ingresar datos, que obtiene datos de la red o hasta de un archivo, debe verificar que los datos se encuentran dentro del rango esperado. Además de controlarse el tipo de dato (*e.g.* cadenas, números, direcciones de memoria), debe validarse que el valor del dato pertenezca al conjunto de valores válidos. Cuando el dato que se espera es el nombre de un color, no basta con verificar que el dato es una cadena de texto sino además que dicho nombre de color se encuentra en el conjunto de los nombres válidos de color. Los vectores de ataque más usuales incluyen la inyección de comandos (*e.g.* SQL, llamadas al sistema operativo, llamadas a código que se ejecuta en otro programa como un navegador web). No obstante, otras formas podrían incluir malformación en las estructuras de datos, como es el caso de un archivo mp3 que puede ser leído por los programas de reproducción multimedia. En gran medida, la flexibilidad de los programas utilitarios es factible gracias a las convenciones de formatos y estructuras de datos. A modo de ejemplo, algunos formatos de imagen deben indicar la resolución apropiada en píxeles, y posiblemente el valor -1 debería ser detectado como un dato inválido.
Condición de carrera	Se dice que dos o más procesos están en condición de carrera si el resultado de los mismos depende del orden de ejecución. Puede existir una vulnerabilidad si no se controla que todos los prerrequisitos hayan sido cumplidos antes de que un proceso realice sus funciones. Los protocolos de comunicación son especialmente sensibles a ataques donde se falsea la identidad del remitente y se proveen datos especialmente diseñados para conseguir modificar un funcionamiento, ganar acceso sin la autenticación apropiada o dejar al servidor o cliente en un estado inconsistente.

Problemas de control de acceso	Existen dos conjuntos fácilmente distinguibles: los controles físicos y los controles lógicos. El primer grupo está destinado a gestionar y restringir la accesibilidad a componentes del sistema tales como computadoras, equipos de red, equipos de distribución, entre otros; el segundo grupo está orientado a regular tanto los recursos asignados a los procesos (*e.g.* archivos, conexiones), como las capacidades y alcance de los distintos usuarios.
Debilidades en la autenticación, autorización o encriptación	En diversas ocasiones, el programador utiliza librerías en forma incorrecta, creando como consecuencia agujeros de seguridad. Un ejemplo de ello es la correcta utilización de clave pública y privada en todas las interacciones entre cliente y servidor, pero cometiendo el error de dejar la clave privada disponible a otros procesos que podrían tener acceso a la misma enviándola a un atacante. El atacante podrá de esta forma impersonar al usuario real debido a que, en cierta forma, la identidad de una de las partes está dada por la tenencia de la clave privada.

CUADRO 3.1: Categorización de vulnerabilidades de seguridad propuesta en la guía para codificación segura de MAC

3.3. Categorías y *Rankings*

El agrupamiento por semejanza facilita el estudio y la búsqueda de soluciones. Los *rankings* permiten enfocar los esfuerzos según sea la urgencia. Al utilizar una lista de prioridades deben comprenderse cuáles son los criterios aplicados. A modo de ejemplo, una lista puede priorizar vulnerabilidades según el daño económico estimado mientras que otra lista puede hacerlo según la probabilidad de que un atacante aproveche dicha vulnerabilidad; claramente, para tomar buenas decisiones se debe conjugar ambas listas adecuadamente.

La lista de los 25 errores más peligrosos en el software (ver apéndice D) creada por SANS, CWE y MITRE[2], reúne lo que dichas instituciones consideran vulnerabilidades encontradas en forma extendida y con gran frecuencia, y que representan serios riesgos. En general, estas vulnerabilidades son fáciles de hallar y explotar; son consideradas graves porque permiten a los atacantes tomar control total de los sistemas o robar datos, entre otras consecuencias. Esta categorización puede ser utilizada en forma de herramienta con fines educativos o de concienciación.

El listado anteriormente referido utiliza el sistema de puntaje denominado *Common Weakness Scoring System* (CWSS) y, como puede encontrarse en [10], la vulnerabilidad CWE-89 relativa a *SQL Injection* se destaca en primer puesto con 93.8 puntos, seguida de CWE-78 relativa a *OS Command Injection* con 83.3 puntos. Las restantes vulnerabilidades obtuvieron puntajes menores a 80. Estos datos deben ser considerados por las instituciones educativas como una llamada de atención sobre la importancia de brindar conocimiento y herramientas eficaces a los educandos a fin de minimizar los errores de diseño y programación que puedan ser explotados en forma de vulnerabilidad.

Uno de los principales y más activos proyectos de la fundación OWASP es denominado *OWASP Top 10* que provee una lista de lo que esta organización considera los 10 riesgos de seguridad más críticos en aplicaciones web [19]. Dicha lista se encuentra en el Apéndice E y el ordenamiento es realizado a través de una metodología propia de *Open Web Application Security Project* (OWASP); esto es, luego de identificar un riesgo se evalúa la posibilidad de explotación y el impacto en que podría derivar[3].

3.4. Entornos de Trabajo y Aprendizaje

Este trabajo no se adentra en profundidad en las áreas conocidas como *Ethical Hacking* ni *Penetration Testing*, aunque está relacionado en forma estrecha. El entorno de experimentación y aprendizaje resultante del presente trabajo podría ser utilizado como iniciación a cursos más avanzados donde se utilicen herramientas automatizadas de búsqueda de vulnerabilidades.

[2]The MITRE Corportation, conocida comúnmente como MITRE, es una organización estadounidense sin fines de lucro localizada en Bedford, Massachusetts y McLean, Virginia. Provee ingeniería de sistemas, investigación y desarrollo, y soporte sobre tecnologías de la información al gobierno de Estados Unidos de América

[3]Más detalles del *Risk Rating Methodology* pueden encontrarse en `https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology`

Existen diversos entornos de trabajo, así como programas, que permiten realizar experiencias prácticas. Los mismos se diferencian en los requerimientos de base, la complejidad de configuración, etcétera. A continuación se presenta un conjunto de herramientas –algunas gratuitas y otras pagas– que son muy utilizadas tanto a nivel de investigación como en trabajos de penetración. Aunque no todas están incluidas en [16], allí pueden encontrarse diversos ejemplos de su utilización y la interacción entre las mismas. En general, cada herramienta cuenta con su documentación propia donde pueden encontrarse ejemplos y guías prácticas.

El proyecto WebGoat[4], que se desarrolla dentro de OWASP, es una herramienta que permite practicar con algunas vulnerabilidades básicas en la programación de sitios web. Un proyecto icónico de esta organización es *OWASP Web Testing Environment* que incluye máquinas virtuales con herramientas ya instaladas, diversos utilitarios y documentación destinados a proveer un ambiente listo para investigar, demostrar y mejorar las habilidades relacionadas a la seguridad. Otros proyectos también importantes son *Offensive Web Testing Framework* – OWASP OWTF, *Zed Attack Proxy* – ZAP, y WebScarab. Estos proyectos están orientados a *penetration testing* e incluyen módulos de proxy para interceptar y modificar datos, inyección de código, manipulación de conexiones para realizar ataques de hombre en el medio (*man in the middle*), entre otros.

Websecurify es una empresa que provee herramientas para encontrar vulnerabilidades en aplicaciones web, varias con orientación al uso en sistema operativo Mac. La página principal es Secapps (`https://www.secapps.com`), que hace uso de otras herramientas creadas por la misma empresa y de uso gratuito como WebReaver (`https://www.webreaver.com`) y Proxy.app (`http://www.proxyapp.io`). Funcionan como escáner de aplicaciones web y proxy para interceptar solicitudes y respuestas en forma de extensión de los navegadores Chrome y Firefox.

Nikto Web Scanner es una herramienta que permite explorar servidores web en busca de archivos peligrosos, componentes desactualizados y otros problemas alegando estar libre de falsos positivos según su página principal (`https://cirt.net/Nikto2`) donde puede encontrarse una lista de los módulos y funcionalidades. Nikto tiene licencia GPL y es de uso gratuito aunque en forma paga es posible obtener archivos útiles con datos y servicios provistos por la empresa Netsparker.

[4] `https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project`.

La herramienta w3af (http://w3af.org) es un entorno de ataque con auditoría escrito en lenguaje Python y licenciado bajo GPLv2.0. Permite la inyección del código malicioso a ser ejecutado en casi cualquier parte de la consulta HTTP como dirección, valor de *cookies*, valores GET, datos, sección multi-parte, entre otros. Los módulos incluidos permiten soporte de proxy, autenticación básica, modificación de los datos del navegador, gestión de *cookies*, caché DNS y carga de archivos. La lista completa de extensiones puede encontrarse en http://w3af.org/plugins.

Burp Suite es un producto de la empresa Portswigger[5] que consta de diversos módulos: proxy para interceptar, inspeccionar y modificar tráfico entre el navegador y la aplicación víctima; una araña que explora el contenido y las funcionalidades; un escáner para detectar en forma automatizada diversas vulnerabilidades; una herramienta de intrusión que permite modificar el vector de ataque ajustándolo para explotar vulnerabilidades poco usuales; una herramienta de repetición para reenviar solicitudes individuales; una herramienta de secuencias para evaluar la aleatoriedad de los datos de sesión; y otros módulos que permiten guardar el estado del trabajo en progreso y programar extensiones propias. La Figura 3.1 muestra un ejemplo de resultado de un análisis de vulnerabilidades.

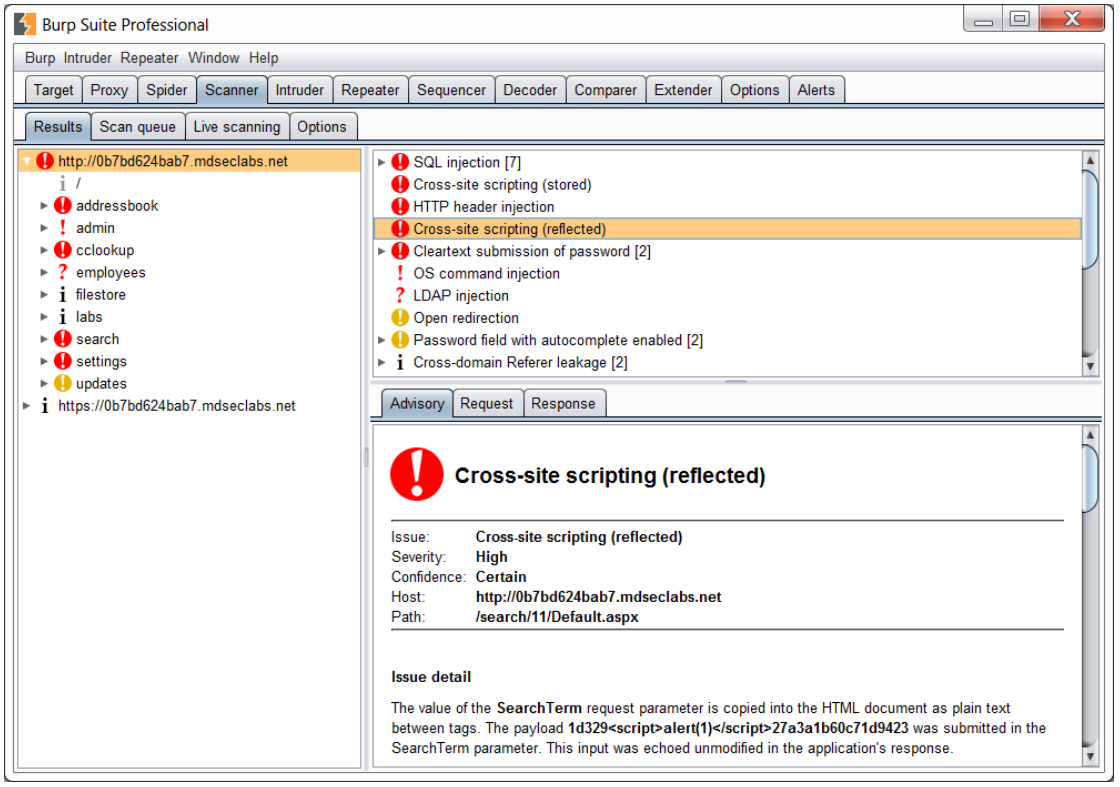

FIGURA 3.1: Herramienta Burp.

[5]Más información en https://portswigger.net/burp.

Dos herramientas muy útiles al momento de buscar explotar vulnerabilidades a través de inyección de código SQL son SQLmap (`http://sqlmap.org`) y sqlninja (`http://sqlninja.sourceforge.net`). Ambas herramientas utilizan distintas técnicas para tratar de realizar las inyecciones con distintos objetivos como: acceso, manipulación de datos en la base de datos, creación de roles u otros objetos, ejecución de comandos en el sistema operativo a través del motor de base de datos, sustracción de un archivo de volcado de datos o de respaldo, entre otros. Existen otras herramientas cuyo propósito principal es escanear a una víctima (equipo o aplicación) y que incluyen ejemplos de cómo explotar las vulnerabilidades encontradas asignando además una valoración de criticidad a cada una; algunos ejemplos son Nexpose y Nessus.

El proyecto *Browser Exploitation Framework* – BeEF es una herramienta que permite realizar ataques creando una *botnet*[6]. La arquitectura de BeEF (ver Figura 3.2) consta de un atacante que se conecta a una interfaz de usuario web desde donde se dirigen las acciones a los navegadores comprometidos (denominados *zombies*) a través del servidor de comunicación denominado *BeEF CS*. El servidor de comunicación se encarga de gestionar la lista de *zombies* conectados, enviar los comandos y en varios casos, recolectar los resultados.

La Figura 3.3 muestra una de las secciones de la interfaz de usuario desde donde puede verse la lista de navegadores web comprometidos, un conjunto de acciones que pueden solicitarse e información encontrada. Algunas acciones requieren parámetros que definan el comportamiento, como por ejemplo reproducir un sonido en un navegador, para lo cual debe indicarse la dirección del archivo de sonido.

Existe una gran cantidad de entornos que permiten buscar vulnerabilidades y explotarlas en aplicaciones web. Shay Chen mantiene una lista[7] actualizada donde pueden compararse más de 60 herramientas en aspectos como precio por modalidad de uso, la precisión en arañas para detectar la estructura y contenido del sitio, inyección de SQL, XSS reflejado, inclusión de archivo remoto (*Remote File Inclusion* – RFI), redirecciones no validadas y archivos no referenciados, de respaldo o viejos que quedan accesibles por descuido.

[6] Término que hace referencia a un conjunto de computadoras o dispositivos informáticos que reciben instrucciones a ejecutar de un coordinador o *master* a través de uno o más canales de comando y control.

[7] Lista mantenida por Shay Chen: `http://www.sectoolmarket.com/price-and-feature-comparison-of-web-application-scanners-unified-list.html`.

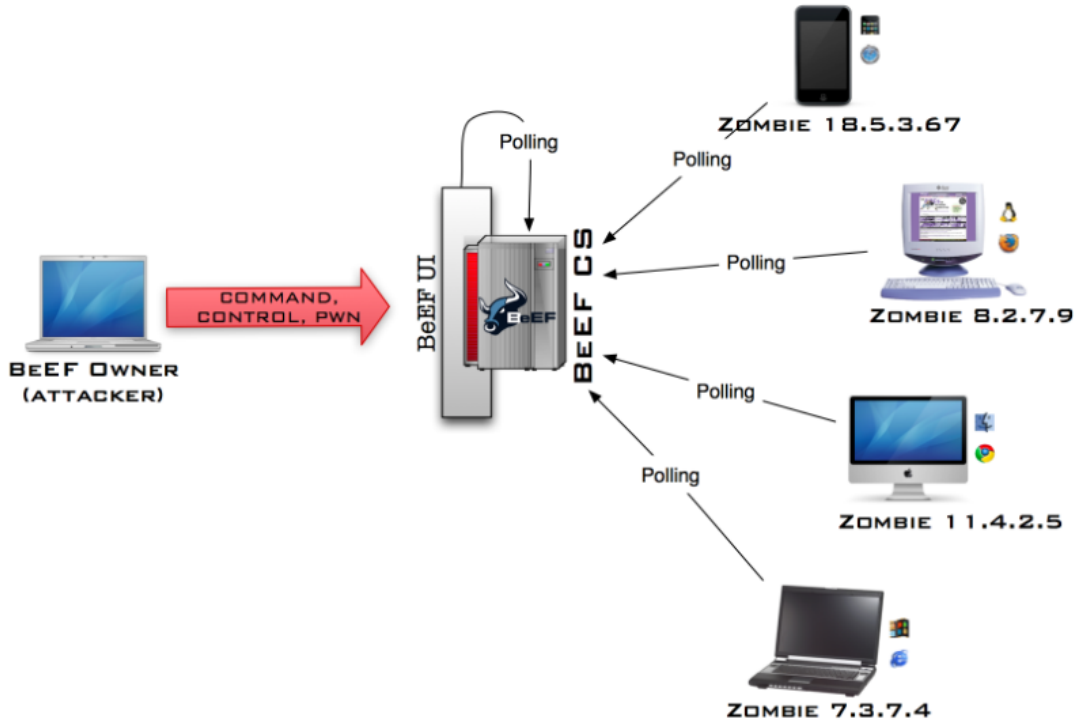

FIGURA 3.2: Arquitectura de Browser Exploitation Framework – BeEF.

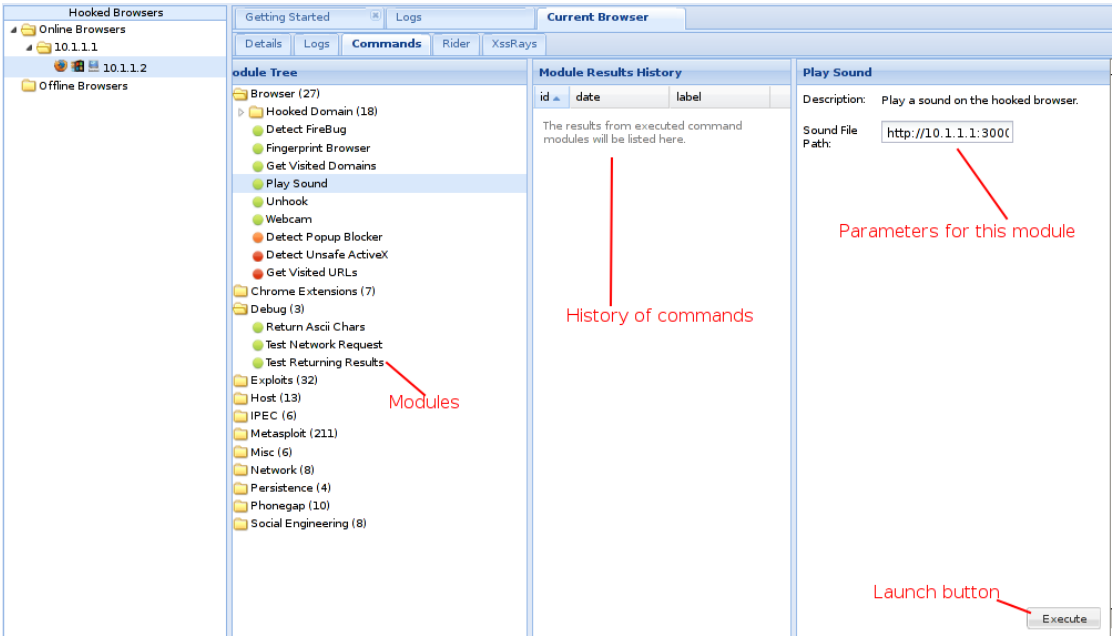

FIGURA 3.3: Interfaz de usuario de BeEF.

Game of Hacks[8] es un juego educativo donde se presentan al jugador trozos de código en distintos lenguajes como SQL, PHP, JavasCript, C, C#, Python, entre otros. Dichos trozos pueden estar relacionados a interacciones con la base de datos, control de permisos, registro de bitácora, presentación de datos a usuario o procesos en general. Como se

[8] http://www.gameofhacks.com.

aprecia en la Figura 3.4, el participante no requiere registrarse en la plataforma y antes de iniciar una partida debe indicar uno de tres niveles de dificultad en que desea jugar.

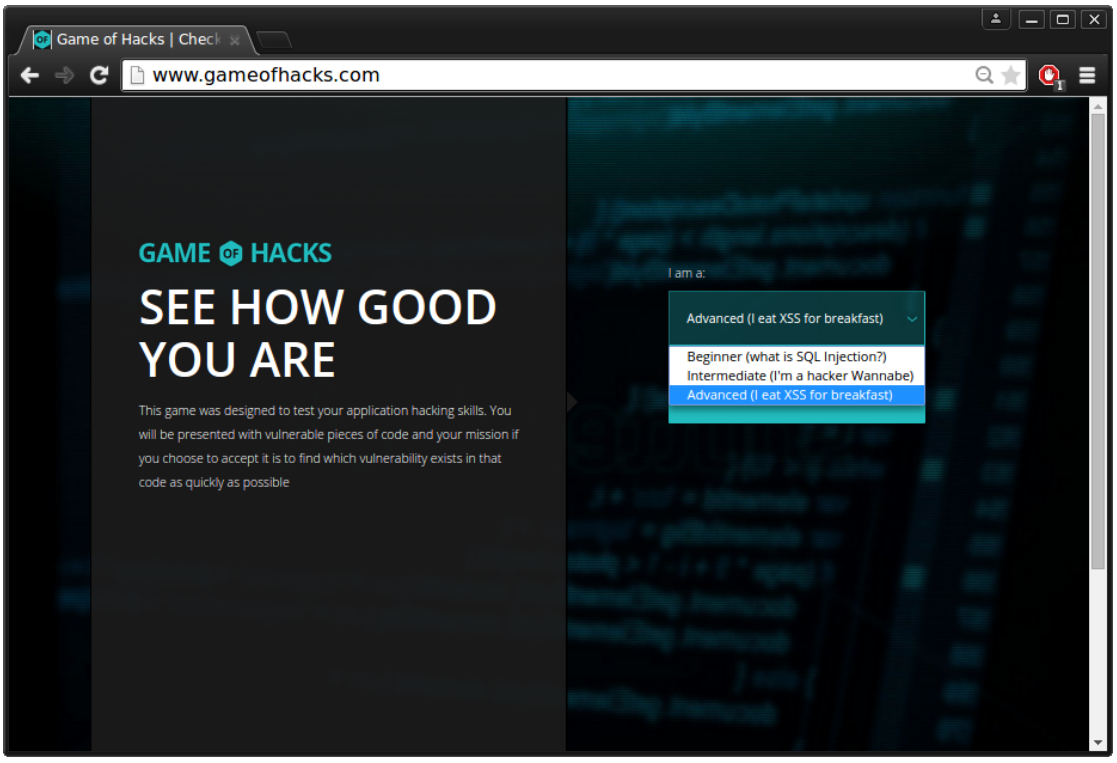

FIGURA 3.4: Pantalla de inicio de Game of Hacks.

Durante cada partida se presentan cinco escenarios que consisten en una función o trozo de código a evaluar y un conjunto de opciones de respuesta. La pregunta en general es: ¿qué vulnerabilidad está expuesta en este código? La Figura 3.5 muestra un ejemplo de escenario donde el participante tiene un minuto para seleccionar la respuesta correcta y al finalizar recibe su puntaje acumulado en la partida.

El sitio *Game of Hacks* permite a los participantes proponer escenarios nuevos, pero como puede notarse, todos los escenarios son teóricos y no se incluyen ejercicios prácticos donde el código se ejecute. Tampoco se consideran vulnerabilidades que provienen del orden en que se ejecutan las funciones y no de la implementación puntual de cada función. En este sentido, los escenarios se aproximan más al test unitario que a la comprobación de sistemas.

La factibilidad de realizar un ataque tiene relación no solo con los recursos necesarios (herramientas, aplicativos, conocimiento) sino también con los lugares de la red e infraestructura donde el atacante pueda tener incidencia. Un claro ejemplo de ello es que

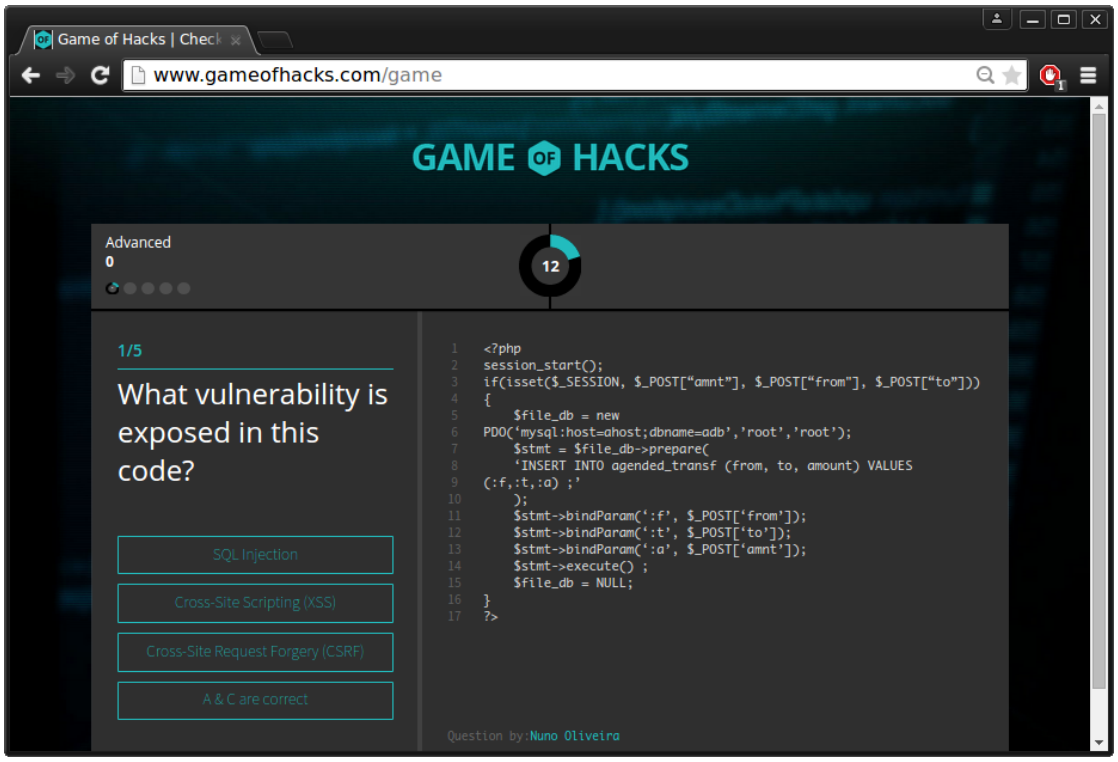

FIGURA 3.5: Escenario propuesto dentro del juego Game of Hacks.

muchos ataques requieren poder escuchar en la LAN en forma promiscua para luego inyectar respuestas que deriven en un encaminamiento de paquetes a través de un equipo controlado por el atacante, envenenar la caché DNS, falsear respuestas ARP, entre otros. Para ello, deben tenerse en cuenta los vectores de ataque que explotan vulnerabilidades en los protocolos de red. La herramienta *NETwork Attacks Framework* – NETA [6] permite simular el funcionamiento de una red compuesta por nodos y realizar ataques a la comunicación entre ellos a través de los distintos protocolos.

Existen diversos sistemas operativos orientados a la investigación o trabajo en el área de seguridad, algunos de ellos son Kali (antes conocido como BackTrack), Parrot Security OS, Pentoo, Tails[9]. De igual forma, existen algunas imágenes de sistemas operativos listas para ser instaladas o montadas como máquinas virtuales que contienen vulnerabilidades de manera intencionada. Diversos cursos de seguridad hacen uso de la máquina virtual Metasploitable, y hasta existe un curso de *ethical hacking* llamado *Metasploit Unleashed* donde se enseña su uso a profundidad; dicho curso es proveído en forma gratuita

[9]Puede encontrarse una lista extensa y comentada de sistemas operativos orientados a la seguridad en https://en.wikipedia.org/wiki/Security-focused_operating_system.

por *Offensive Security* en un esfuerzo por concienciar y recaudar fondos para niños desfavorecidos en África oriental[10]. Otras distribuciones también dedicadas a la investigación o trabajo en el área de seguridad son Damn Vulnerable Linux, Samurai Web Testing Framework (SamuraiWTF) y LAMPSecurity. A nivel aplicaciones, existen algunas que pueden ser instaladas en un servidor propio para realizar las experiencias, así como sitios intencionalmente vulnerables para realizar experiencias. Alguno son: Mutillidae (proyecto de OWASP), Damn Vulnerable Web App (DVWA), buggy web application (bWAPP), el juego de video para entrenamiento y aprendizaje llamado CyberCIEGE [15] (ver Figura 3.6), y los sitios `hack.me`, `hackaserver.com`, `www.hacking-lab.com` y `hackyourselffirst.troyhunt.com`[11].

FIGURA 3.6: Captura de pantalla del juego de video CyberCIEGE.

[10] `https://www.offensive-security.com/metasploit-unleashed`.
[11] Pueden encontrarse más recursos en el sitio `https://www.vulnhub.com/resources`.

Capítulo 4

Propuesta de Entorno

4.1. Descripción de Componentes

El lenguaje de programación utilizado en la construcción de la herramienta desarrollada en este trabajo es PHP debido a su uso extendido y popularizado. Otros lenguajes que pudieron haber sido seleccionados son Ruby o Python pero los autores priorizaron el hecho de que PHP es anterior a dichos lenguajes por lo que es más probable que los instructores, profesores o guías estén familiarizados con él. Como motor de base de datos fue elegido PostgreSQL y con poco esfuerzo podrían realizarse las adaptaciones necesarias para utilizar MySQL, Oracle u otro motor, dada la simplicidad de las funciones y demás objetos de base de datos.

El entorno se conforma principalmente de escenarios. Algunos componentes están destinados a la administración, otros a las experiencias propuestas, y otros a la recuperación de datos. La intención del entorno es servir como una herramienta de uso simple que permita incluir experiencias prácticas en los cursos regulares. Los escenarios que se presentan son de nivel básico, y no son suficientemente complejos como para satisfacer las necesidades de una materia que se especialice en seguridad.

En el diseño del entorno se tuvo en cuenta que el instructor puede agregar nuevos escenarios o modificar a los escenarios existentes. Para ello se necesitan algunos conocimientos de programación. Los escenarios podrán ser modificados, ampliados o extendidos en la

medida que las listas de OWASP y CWE/SANS sean actualizadas. Una forma alternativa de utilizar la herramienta propuesta es que los participantes puedan crear o modificar escenarios y que los mismos sean probados por sus pares.

4.1.1. Escenarios

Cada escenario plantea un desafío distinto, y todos guardan relación tanto con el Ranking de Errores Más Peligrosos CWE/SANS (ver Apéndice D) como con la lista de 10 riesgos de seguridad más críticos en aplicaciones web (ver Apéndice E) mantenida por OWASP, salvo el quinto escenario que plantea una situación real y patente pero no cubierta por dichas listas aunque sí por algunas guías [20, 21].

El instructor puede agregar nuevos escenarios o modificar los existentes, aunque para ello debe tener conocimientos suficientes de programación. Los escenarios propuestos son relativamente simples y sirven como ejemplo de que el entorno puede mejorarse aumentando los desafíos. A continuación se describen los ocho escenarios incorporados hasta el momento[1].

4.1.1.1. Escenario 1

El primer escenario demuestra una de las consecuencias de un error típico de programación relacionado al limpiado[2] de datos. La vulnerabilidad generada por el error de programación permite que se inyecte código SQL y ocupa el primer lugar en ambas listas: CWE/SANS con 93,8 puntos bajo el identificador CWE–89, y OWASP con identificador A1. El escenario también se relaciona con el segundo puesto de la lista OWASP sobre déficit en autenticación y manejo de sesiones.

Específicamente, el escenario se centra en explotar la vulnerabilidad en el método de autenticación. El objetivo planteado es ingresar, sin conocer ni adivinar, usuario y contraseña; algunos trozos de código relevante al escenario pueden encontrarse en el Apéndice C.1. La Figura 4.1 muestra la pantalla inicial a la que se enfrenta el participante. Se espera que el participante, aunque requiera de una pequeña investigación en Internet (*e.g.*

[1] Fueron incluidos ocho escenarios siendo esta una cantidad suficiente que permite ejemplificar el tipo de experiencias que pueden realizarse en un ambiente de clase sin excesiva preparación, y dadas las limitaciones del tiempo disponible.

[2] El término más apropiado en inglés es *sanitization*. En diversos sitios dicho término es traducido como *sanitización*, aunque dicha palabra no está incluida al diccionario de la Real Academia Española.

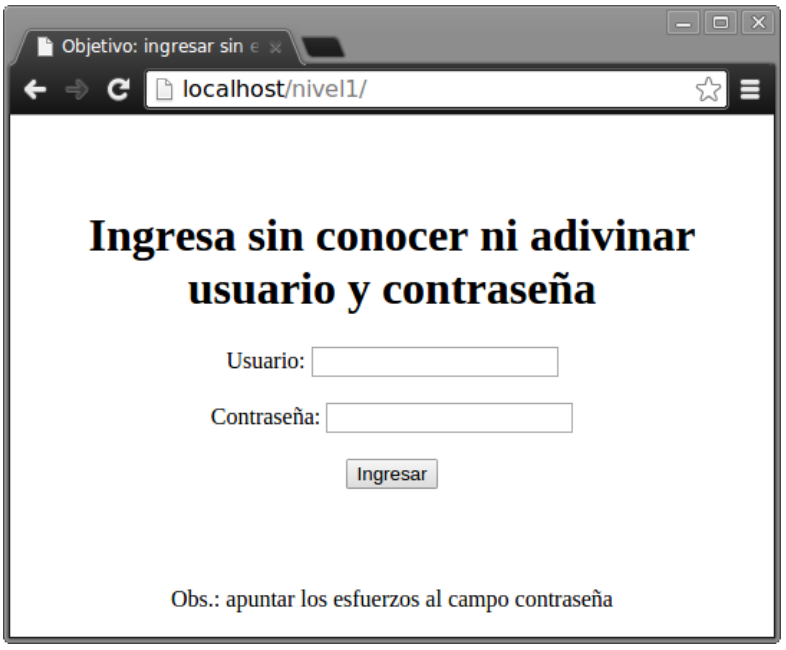

FIGURA 4.1: Pantalla del escenario 1.

http://www.redeszone.net/seguridad-informatica/inyeccion-sql-manual-basico o http://www.securityartwork.es/2013/11/21/evasion-de-autenticacion-con-inyeccion-sql) realice una inyección de código que cambie la sentencia lógica evaluada por el motor de base de datos en la aceptación del usuario permitiendo así su acceso al sistema.

4.1.1.2. Escenario 2

Según el listado confeccionado y mantenido por CWE/SANS, la inyección de código puede corresponder conceptualmente al tipo de vulnerabilidades con identificador CWE–79 –que se encuentra en cuarto puesto– cuya raíz es la inapropiada neutralización de datos de entrada durante la generación del contenido. Además, la vulnerabilidad denominada CWE–134 trata acerca de la falta de control sobre los formatos de cadena. Según el listado mantenido por OWASP, el tercer riesgo más crítico es el de XSS causado por la falta de limpiado de datos ingresados por el usuario y el correspondiente escape de ciertos caracteres como los símbolos mayor y menor.

En la Figura 4.2 puede verse la primera pantalla que presenta el segundo escenario; para superarlo, el participante debe lograr inyectar código JavaScript entre los datos de una persona para que se despliegue una alerta a los usuarios que carguen la página.

Cada persona listada posee un botón que permite ingresar a la edición de los campos correspondientes a sus datos (ver Figura 4.3).

FIGURA 4.2: Pantalla del escenario 2, listado de datos.

El ejercicio plantea que el código inyectado sólo despliegue un cartel emergente donde se muestre un texto elegido por el participante. Este método, por simple que parezca, tiene gran potencia ya que podría obtener información de sesión del usuario y enviarla a un sitio externo utilizando *HttpRequest* que es el método de pedido *Hypertext Transfer Protocol* (HTTP). También podrían descargar datos al cliente y acceder a información existente en el DOM, entre otros métodos que forman parte de los ataques. En concreto, el participante realiza inyección de código a través de uno de los campos y verifica si tuvo o no éxito reingresando al formulario. El Apéndice C.2 muestra parte del código fuente involucrado de forma a observar lo que reciben y ejecutan los usuarios que ingresen a la página atacada.

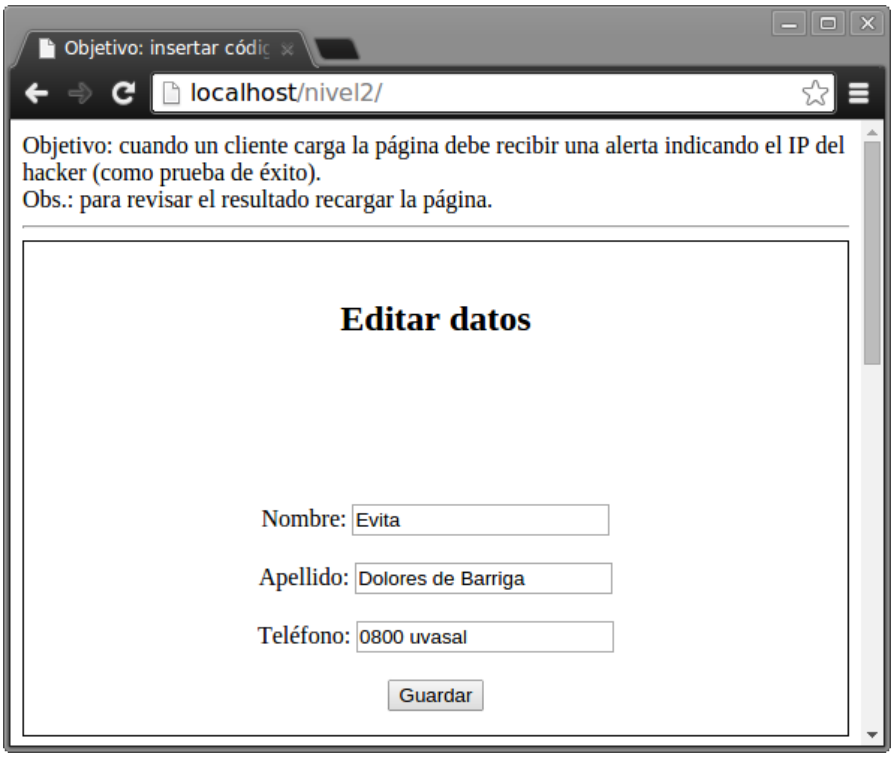

FIGURA 4.3: Pantalla del escenario 2, edición de datos.

4.1.1.3. Escenario 3

Este escenario es similar al anterior y provee la misma interfaz al participante, pero difiere en el objetivo. El tercer escenario solicita al participante que el código JavaScript inyectado redirija al usuario en forma automática a otro sitio completamente diferente. La peligrosidad de este ataque aumenta si el sitio vulnerable utiliza marcos (*frames*) ya que en tal caso el usuario podría no notar el hecho de que parte del sitio que observa fue reemplazado por contenido malicioso que podría imitar al original.

A modo de ejemplo, la Figura 4.4 muestra la composición de una página web que utiliza distintos marcos para desplegar el contenido. Esta técnica es bastante utilizada debido a que facilita modificaciones posteriores y minimiza la cantidad de código fuente porque un mismo componente es invocado desde distintas páginas en vez de que cada página contenga una copia propia del código fuente.

Este escenario está relacionado al error con identificador CWE–79 que trata sobre la inapropiada neutralización de datos de entrada, con CWE–352 que se encuentra en el puesto once y trata sobre *Cross-Site Request Forgery* (CSRF), y con CWE–601 que se relaciona con las redirecciones a sitios no confiables. Revisando la lista de OWASP

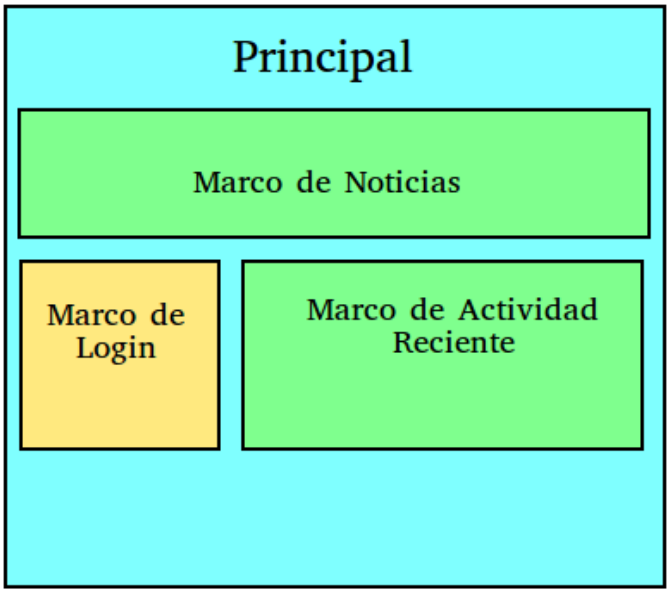

FIGURA 4.4: Ejemplo de uso de marcos en un sitio.

se relaciona al octavo puesto sobre CSRF y al décimo puesto sobre no validación de redirecciones y reenvíos. Actualmente algunos entornos como Django[3] y los navegadores web recientes implementan cabeceras que controlan el origen del contenido y no permiten que un marco *iframe* sea cargado con contenido de un dominio distinto a los autorizados o al del sitio principal. Esta vez, el participante realiza inyección de código JavaScript y verifica si tuvo o no éxito cargando nuevamente la página que lo debería redirigir al sitio externo inyectado por el atacante. Un ejemplo del código JavaScript que permite realizar el ataque se encuentra en el Apéndice C.3.

4.1.1.4. Escenario 4

El cuarto escenario toma un rumbo distinto al de los escenarios anteriores, ya que se centra en el método utilizado por un formulario al enviar los valores y en la mala práctica de utilizar campos ocultos para definir el comportamiento de una funcionalidad. El escenario está relacionado con el error CWE–79 ya referido anteriormente, y con los errores CWE–862 (autorización faltante) y CWE–829 (inclusión de funcionalidades desde ámbitos de control no confiables). En lo referente al listado de OWASP, el escenario se relaciona al sexto puesto (exposición de información sensible) y al séptimo puesto (falta de control de nivel de acceso).

[3]Sitio oficial del proyecto: `https://www.djangoproject.com`.

En ingeniería de *software* el principio *Don't repeat yourself* (DRY) establece que en lo posible no debe duplicarse o repetirse código en distintos lugares cuando por diseño podría crearse un componente a ser utilizado en dichos lugares. Este principio puede derivar en la provisión de una función que devuelva datos según los parámetros recibidos. El problema con esta función supuesta surge cuando no se limita el universo de posibles parámetros según el contexto. Por ejemplo, en operaciones de autenticación puede ser necesario obtener un datos sensibles, pero puede convertirse en una vulnerabilidad si el usuario tiene acceso a dichos datos sensibles.

FIGURA 4.5: Pantalla del escenario 4, listado de usuarios.

Como objetivo del escenario se solicita al participante que obtenga la contraseña de un usuario. La interfaz muestra una lista de usuarios (ver Figura 4.5) y permite desplegar la dirección de correo de cada usuario en particular a través de un botón relacionado a cada uno. El formulario que contiene a los usuarios y da funcionalidad a los botones tiene como acción cargar una página donde se encuentra una función que retorna el dato solicitado por parámetro.

La explotación de la vulnerabilidad se da si el participante logra modificar el dato solicitado (*i.e.* cambiar el parámetro que recibe la función). Los detalles del código fuente y una solución se encuentran en el Apéndice C.4. Como se indica en la Sección 1.2: mientras la correctitud se encarga de las funcionalidades del sistema, la seguridad se encarga de la falta de funcionalidades. Este escenario demuestra la peligrosidad de que el usuario tenga acceso a modificar el funcionamiento del sistema por falta de validación, exposición de código y en general, por mal diseño.

4.1.1.5. Escenario 5

El quinto escenario enfoca un error humano que puede dar lugar a diversas vulnerabilidades. El objetivo es demostrar la gravedad del error en forma general. Para ello se muestra al participante un texto que habla sobre la importancia de tener copias de respaldo antes de realizar modificaciones importantes a ciertos archivos. El error humano surge debido a que algunas personas dejan esas copias de respaldo en ubicaciones alcanzables por un usuario, y luego se olvidan de borrar los archivos temporales. El error de procedimiento no está incluido en las listas referidas debido a que las mismas se centran en las aplicaciones y no en la configuración de servidores, pero figura en guías como [20, 21].

Para realizar una copia de respaldo rápida, una forma muy utilizada es copiar el archivo y agregar un sufijo que indique que el mismo es una copia vieja o de respaldo (*e.g.* utilizando extensiones como .old, .bkp). El servidor web, a menos que sea configurado en forma distinta, permite descargar al cliente los archivos cuyas extensiones no sean bloqueadas o para interpretar (*e.g.* .php, .aspx). Otra forma de exposición de código se da cuando el programador deja comentarios que incluyen datos sensibles como contraseñas, nombres de usuario u otra información sensible. El Apéndice C.5 muestra el tipo de archivos, su ubicación y la revelación de datos que da el escenarios.

4.1.1.6. Escenario 6

Este escenario es similar al primero en el sentido de inyección de código SQL, pero difiere en la finalidad. Mientras el primer escenario sólo buscaba que una consulta a la base de datos retorne al menos un registro, el sexto escenario se enfoca en el alcance que podría

tener la inyección. El escenario está asociado en forma directa a la vulnerabilidad de mayor riesgo en la lista creada y mantenida por CWE/SANS: CWE–89 (relacionada a la falta de neutralización de código SQL), pero se ve potenciada por CWE–250 que está relacionada a la ejecución con permisos innecesarios.

Revisando la lista de riesgos más peligrosos de OWASP, el escenario se relaciona con el primer puesto debido a la posibilidad de inyectar código SQL que resulte en el borrado de datos, el tercer puesto debido a la falta de limpieza de datos de entrada que posibilita un ataque de tipo XSS y al quinto puesto relacionado a mala configuración de seguridad ya que la operación de borrado no se ve obstruida por permisos.

Las interfaces de usuario utilizadas son las mismas que las mostradas en las Figuras 4.2 y 4.3, en las que se lista a las personas existentes y se permite la edición de los datos de una persona seleccionada. Para superar el escenario, el participante debe aprovechar la vulnerabilidad existente que permite inyectar código SQL con la consecuencia de borrar todos los datos de la tabla que contiene información de usuarios, por ejemplo, utilizando la sentencia `delete` y verificando el éxito de su ataque al no encontrar datos en la lista de usuarios existentes que muestra la página. El Apéndice C.6 muestra una posible solución al escenario.

4.1.1.7. Escenario 7

El escenario propuesto en esta sección tiene como meta el robo de información de las *cookies* a través de inyección de código JavaScript. El objetivo del escenario es demostrar cómo un atacante podría robar y alterar la información que contienen dichos archivos. Este ataque podría causar graves daños como la alteración de artículos y cantidades agregados a un carrito de compras en línea o permitir que el atacante gane acceso a una aplicación a través de la cuenta de la víctima aún sin conocer nombre de usuario ni contraseña.

Este ataque posee características de XSS encontrándose en el cuarto puesto de la tabla de CWE/SANS bajo el identificador CWE–79 y en el tercer puesto de la lista OWASP aunque se relaciona en forma más directa con el segundo puesto de ésta última, relacionado al déficit en autenticación y manejo de sesiones. El esfuerzo del atacante para

modificar los datos es menor si las *cookies* son almacenadas en texto plano en vez de encriptado, el octavo puesto de la lista CWE/SANS lo ocupa la vulnerabilidad CWE–311 relacionada a la falta de encriptación de datos sensibles.

Las pantallas a las que accede el usuario son las mostradas en las Figuras 4.2 y 4.3, en las que se lista a las personas existentes y se permite la edición de los datos de una persona seleccionada. El participante debe inyectar código JavaScript en un formulario para acceder a la *cookie* ya creada por la herramienta donde se almacena la información de sesión. El Apéndice C.7 muestra una posible solución al escenario utilizando inyección de código JavaScript en un campo que es almacenado en la base de datos.

4.1.1.8. Escenario 8

El octavo escenario aborda un tipo de vulnerabilidad que no está relacionada con la inyección de código ni errores en los controles de permisos, sino que se centra en un área del diseño de protocolos seguros. El escenario demuestra en forma práctica lo que podría ocurrir cuando se requiere que un dato sea aleatorio, pero debido a la implementación se asignan variables con una aleatoriedad pobre.

A modo de ejemplo, HTTP es un protocolo sin estado, por lo que en cada transacción el navegador envía al servidor el valor de una variable de sesión. Dicho dato es utilizado por el servidor para recuperar la información que tiene sobre la sesión del cliente. En la Sección 4.1.1.7 se demostró cómo podría obtenerse el dato de sesión de un usuario autenticado y utilizarlo para ganar acceso simulando ser el cliente original. Si el atacante adivina el identificador de sesión de otro usuario, puede utilizar la sesión de éste último sin necesidad de interceptar paquetes que intercambian entre cliente y servidor, ni de realizar un ataque directo a la víctima.

En el ejercicio propuesto, se presenta al usuario una pantalla (ver Figura 4.6) que da ciertas pistas induciendo al participante a solicitar cupones en el recuadro (a) y analizar la aleatoriedad de los códigos. En el recuadro (b) se mantiene un historial de los códigos vistos y su equivalencia en monto. El objetivo del escenario es que el participante deduzca la lógica detrás de la generación de códigos y utilizando el recuadro (c) encuentre el cupón por valor de un millón de Dólares.

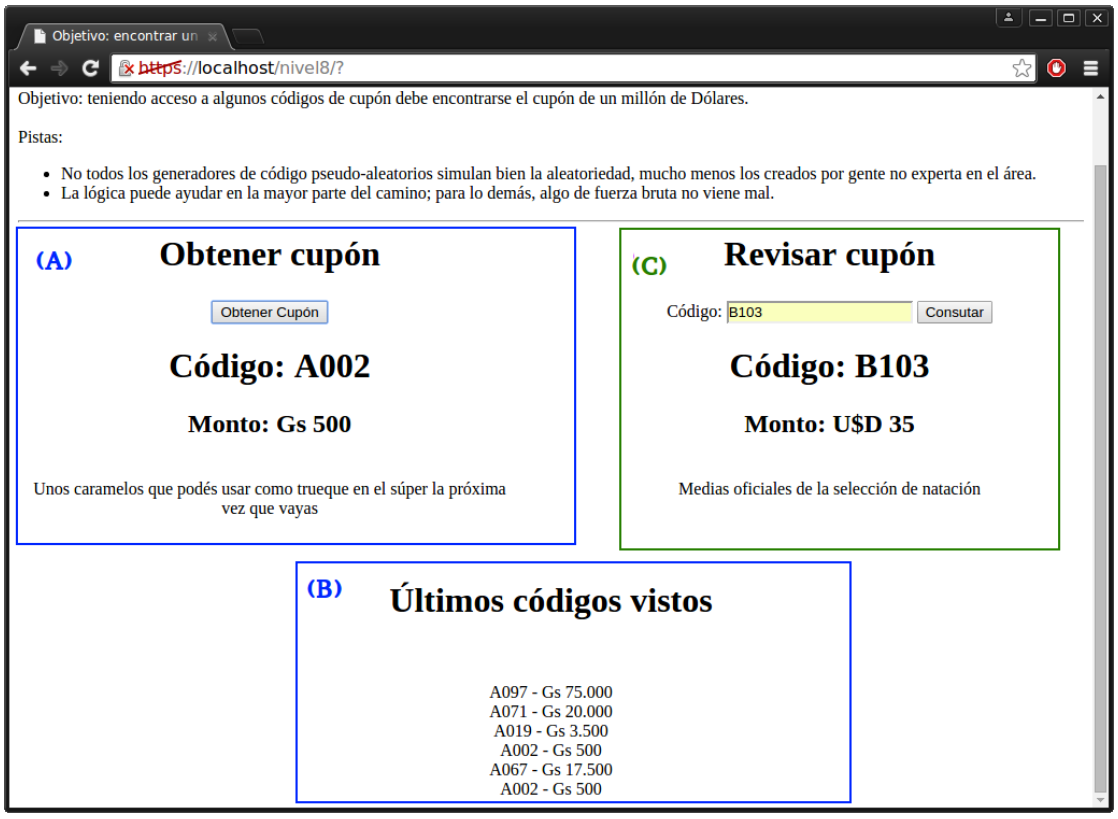

FIGURA 4.6: Pantalla del escenario 8 donde se pueden obtener y consultar cupones.

Existe una amplia gama de situaciones en las que la falta de aleatoriedad genera problemas de seguridad por lo que no se recomienda que personas no expertas utilicen –salvo con fines de investigación o educativos– generadores pseudo-aleatorios de diseño propio [22, 23]. En ocasiones se hace uso de componentes de terceros que pueden tener vulnerabilidades conocidas o que requieren cierta configuración para funcionar en forma segura. Estas dos últimas posibilidades ocupan el quinto y el noveno puesto en la lista de riesgos más peligrosos de OWASP bajo los títulos de 'Mala configuración de seguridad' y 'Utilización de componentes con vulnerabilidades conocidas', respectivamente. En concreto, el participante puede observar la relación entre cupón y premio para luego intentar deducir el código del cupón especificado en el ejercicio, para lo cual hace uso de una entrada del formulario donde puede verificar el premio que corresponde a cada código que el participante desea consultar. El participante puede probar tantos códigos de cupones como desee, pero claramente le será muy difícil lograr el objetivo por búsqueda exhaustiva. El Apéndice C.8 muestra el razonamiento utilizado en la generación de códigos.

4.1.2. Interfaz del Participante

Los participantes deben tener una cuenta creada por el administrador, aunque el sistema podría ser fácilmente extendido para que dicha cuenta sea creada por el mismo participante. Esto ahorraría tiempo al administrador ya que en vez de que una persona cree muchas cuentas, cada participante podría crear la suya propia. No obstante, en un ambiente de clase de una institución educativa, podría ser deseable que sea el administrador quien habilite las cuentas ya que es posible que el proceso deba seguir un protocolo formal si se asignará una calificación con cierto peso dentro de la nota del alumno.

El inicio de sesión (ver Figura 4.7) requiere que el usuario ingrese la dirección de correo electrónico con que fue registrado, y una contraseña.

FIGURA 4.7: Pantalla de inicio de sesión.

Una vez autenticado, el participante sin permisos de administrador accede a la pantalla principal que se muestra en la Figura 4.8. La interfaz contiene un menú de niveles y una barra de opciones donde: el botón 'menú' lleva a la página principal, 'Restablecer datos' borra los datos de las tablas específicas del participante y los inserta nuevamente a partir de las tablas plantilla, y finalmente 'Salir' que quita al participante del sistema llevándolo a la pantalla de inicio (ver Figura 4.7).

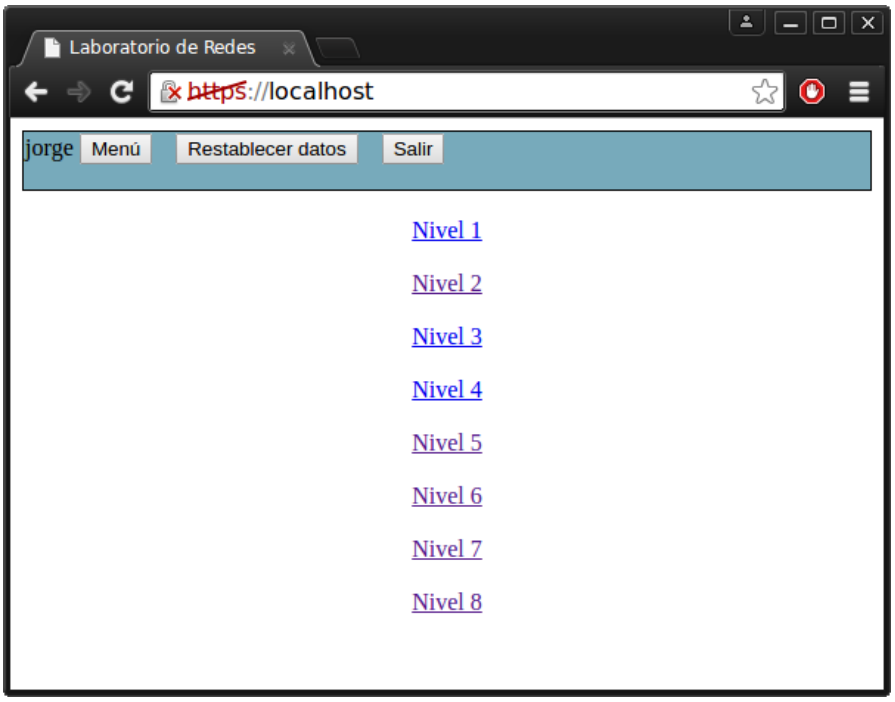

FIGURA 4.8: Página principal del participante.

4.1.3. Administración

Cuando la cuenta del participante tiene permisos de administrador, a la barra de opciones se agrega el botón 'Administrar' (ver Figura 4.9). Cuando el usuario presiona dicho botón, se despliega la página mostrada en la Figura 4.10.

Desde la página de administración pueden crearse y eliminarse cuentas de participantes. De ser requerido, pueden agregarse más campos a la tabla de participantes, modificarse la página de creación y una función en base de datos que inserta los datos. Aunque este proceso requiere cierto conocimiento de programación, cualquier persona que desee extender el perfil del participante podrá hacerlo con poco esfuerzo.

Otra funcionalidad de la página de administración es restablecer los datos para todos los participantes registrados. Aún cuando cada participante tiene la posibilidad de limpiar sus propios datos desde un botón en la barra superior, es probable que quien coordina una actividad desee que los datos estén limpios al iniciar la práctica sin que cada usuario deba hacerlo por sí mismo.

El esquema de base de datos (ver Apéndice B) contiene datos base de los que se sirven algunos escenarios y destina algunas tablas a almacenar una copia de los datos base para

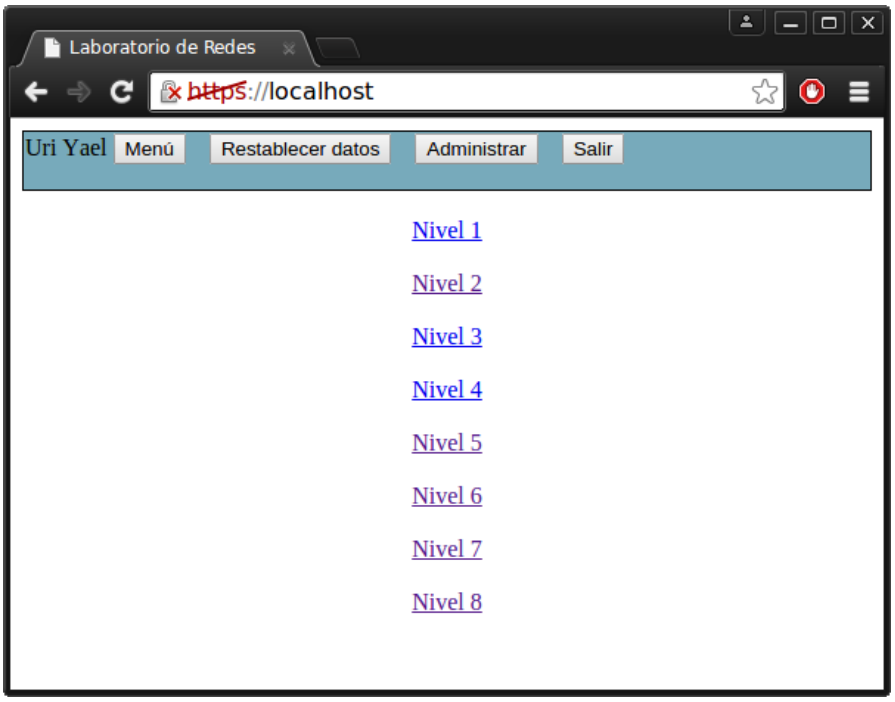

FIGURA 4.9: Página principal del participante.

cada participante. Cada usuario puede realizar el reciclado de su copia de datos ya que algunos escenarios incluyen su modificación.

4.2. Utilización en Clase

Uno de los principales beneficios del entorno de experimentación propuesto es su bajo requerimiento de recursos físicos necesarios para realizar las experiencias. Dos requerimientos básicos y suficientes son, primero, contar con una computadora o máquina virtual que cumpla la función de servidor y, segundo, una red que permita a las computadoras o dispositivos de los participantes cargar en un navegador las páginas web provistas por el servidor.

La computadora o máquina virtual bajo dominio del instructor debe contar con un programa servidor web que incluya un módulo intérprete del lenguaje PHP, y con un motor de base de datos. El Apéndice B provee las secuencias de comando necesarios para crear el esquema de base de datos válidas para el motor PostgreSQL y puede ser fácilmente modificado para otros motores como MySQL, Oracle, entre otros.

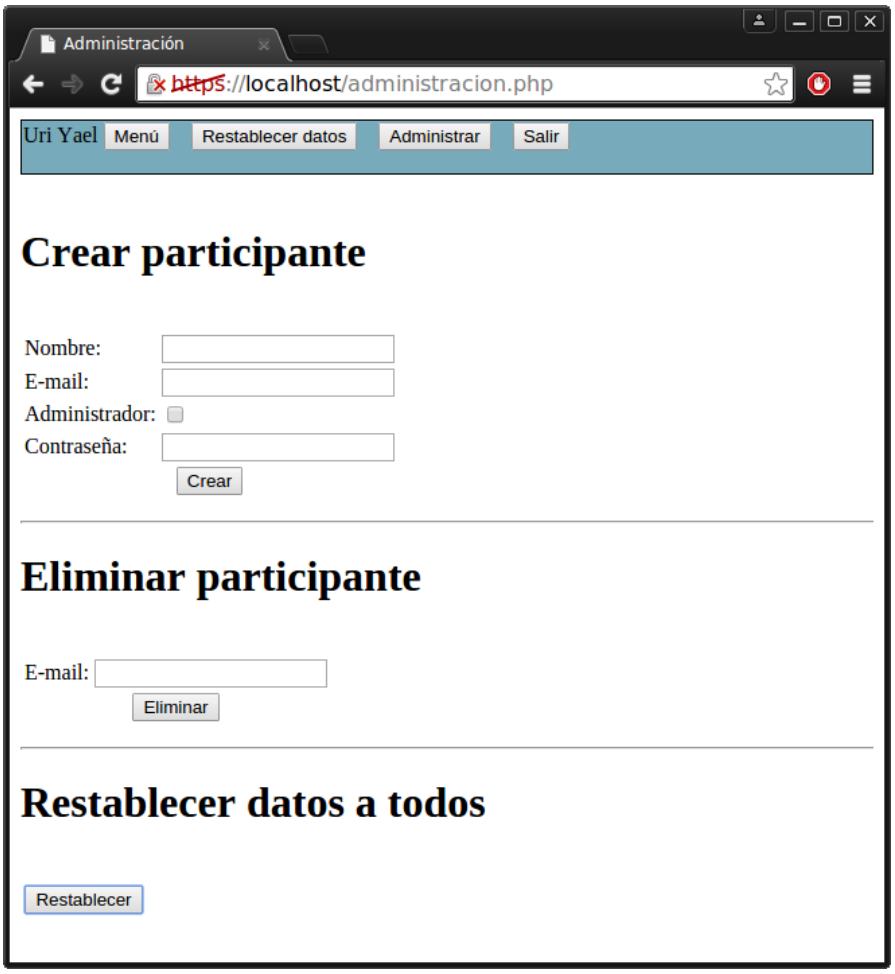

FIGURA 4.10: Página principal del participante.

La red que permite la conexión entre los dispositivos cliente y el servidor puede ser creada con medios guiados o inalámbricos. Si el espacio utilizado no es un laboratorio que ya cuente con componentes de red como conmutadores y el cableado correspondiente, bastaría con utilizar un punto de acceso inalámbrico para crear una red Wi-Fi.

El coordinador de la práctica debe establecer las reglas que regirán durante la realización de las experiencias. Por ejemplo, debe definirse si los participantes podrán consultar entre sí o acceder a Internet en busca de información y ejemplos.

No todos los participantes avanzan al mismo ritmo y el bagaje de experiencias con que cada uno de ellos cuenta influirá en forma decisiva sobre el tiempo que requerirán las personas para finalizar exitosamente cada escenario. Por motivos de organización, deben establecerse los tiempos máximos factibles.

Es aconsejable que los instructores o coordinadores estén atentos en todo momento al

avance de cada participante a fin de detectar cuando alguno de ellos se encuentra atascado y no consigue avanzar. Es importante que las personas que actúen de facilitadores durante la experiencia intenten comprender el proceso deductivo realizado por los participantes para indicar algunas asunciones erróneas o sugerir una idea con la que el participante logre salir del estancamiento y continúe avanzando hacia una solución del ejercicio.

Dependiendo del conocimiento específico de los encargados y realizadores de las prácticas, resulta de gran utilidad realizar comentarios sobre variaciones posibles de los ataques, consideraciones sobre las consecuencias y comparaciones con escenarios y situaciones reales.

Capítulo 5

Experiencia Realizada

5.1. Antecedentes

El autor del presente trabajo, en clases de la cátedra Redes de Computadoras dictadas en la Universidad Católica "Nuestra Señora de la Asunción", tuvo la oportunidad de introducir conceptos generales de seguridad y hacer énfasis en aquellos que se relacionan a programación y algunos protocolos. En diversas ocasiones, y aunque existen excepciones, la mayoría de los alumnos captan las ideas a partir de ejemplos concretos pero tienen dificultades al momento de abstraerse y comprender el caso general. Esto puede deberse a la poca o nula cantidad de experiencias prácticas relacionadas a seguridad de la información que se desarrollan en la carrera.

Los alumnos de la cátedra antes mencionada fueron consultados sobre los que, en su opinión, son los motivos principales por los que creen que no es mayor el número de experiencias similares en su carrera. A este respecto, los alumnos distinguen tres motivos principales:

- Falta de horas cátedra disponibles para dedicar a la experimentación y de horas de los profesores fuera de clase para preparar en forma adecuada los materiales.
- La cantidad de esfuerzo que requieren la actualización continua y el mantenimiento de los materiales para mantenerse actualizado con el avance del estado del arte.
- El nivel de complejidad inherente a crear y ejecutar las experiencias con los materiales disponibles.

En el año 2013, debido al interés de gran parte de los alumnos a los que se refieren los párrafos anteriores, se propuso realizar –dentro del ámbito de la cátedra– ejercicios prácticos básicos que puedan fijar conceptos a través de la experimentación. El autor de este trabajo inició así la creación de unos pocos ejercicios que ayudasen a los alumnos a experimentar algunas consecuencias de vulnerabilidades en una página web. La presente tesis de maestría permitió evolucionar a los ejercicios realizados convirtiéndose éstos en un entorno de experimentación que actualmente cuenta con ocho escenarios distintos (ver Capítulo 4).

5.2. Ejecución

Los pocos ejercicios creados inicialmente fueron resueltos por los participantes en los años 2013 y 2014 utilizando un espacio de 3 horas cátedra cada vez, es decir, en 135 minutos reloj. La evolución del entorno en el marco del presente trabajo permitió que en agosto del año 2015 se realizaran el doble de ejercicios en 4 horas cátedra. Esto fue posible gracias a la eficiencia y simplicidad de la administración del entorno.

El laboratorio de seguridad llevado a cabo en agosto del año 2015 contó con 17 participantes que optaron por realizarlo en parejas aunque una minoría prefirió la forma individual. La consigna fue enfrentarse a los escenarios pudiendo consultar cualquier material de apoyo incluyendo los que pudieran encontrar en Internet. El autor de este trabajo actuó como coordinador y guía asegurándose en todo momento de que los participantes comprendieran las consignas y ayudando a través del razonamiento a cada persona que no lograba avanzar adecuadamente.

Durante la ejecución de la experiencia los participantes realizaron anotaciones y capturas de pantalla para utilizar dicha información en el informe de la experiencia que debieron confeccionar. En el informe cada pareja o participante individual debía explicar las vías de acción o estrategias que tomaron para cada escenario y comentar sobre el proceso, aunque no hayan logrado completar en forma exitosa el objetivo de todos los escenarios.

En la gran mayoría de los casos, considerando todas las veces en que se realizó el laboratorio de seguridad en el ámbito de la citada cátedra Redes de Computadoras, existe una marcada diferencia en la profundidad de la comprensión entre los participantes que tuvieron con anterioridad la oportunidad de abordar conceptos de seguridad en forma

FIGURA 5.1: Alumnos de la cátedra Redes de Computadoras participando del laboratorio de seguridad en agosto del año 2015.

FIGURA 5.2: Participantes del laboratorio de seguridad en agosto del año 2015.

práctica. Dicha oportunidad pudo darse ya sea por un trabajo práctico realizado en alguna cátedra o por interés personal fuera del ámbito de la universidad.

Durante el proceso de este trabajo se introdujo al entorno de experimentación el concepto de aislamiento entre participantes. Para ello, cuando se crea un nuevo participante, el entorno en forma automática está creando un usuario particular en el motor de base de datos, creando un nuevo juego de tablas, asignando los permisos correspondientes sobre cada objeto y cargando los datos iniciales. Esta funcionalidad simplifica la administración del entorno y permite que cada participante restablezca su copia de datos iniciales sin afectar a los demás participantes. En los laboratorios anteriores al del año 2015, el coordinador debía dedicar tiempo a la recuperación de datos cada vez que un participante

realizaba una inyección exitosa afectando a los demás participantes; en el año 2015 esta funcionalidad permitió que el coordinador destinara un tiempo considerablemente mayor a dar seguimiento y apoyo a cada alumno.

5.3. Resultados

La experiencia logró que los participantes comprendan a través de la experimentación algunas de las consecuencias de las vulnerabilidades que pueden encontrarse en diversas páginas y aplicaciones web. Se enfatizó en todo momento que el objetivo es mejorar la comprensión y adquirir práctica en aspectos de la seguridad pero que, considerando que cada participante puede tener una base de conocimientos y experiencias distinta, la meta no es finalizar exitosamente todos los escenarios sino lograr avanzar según las posibilidades de cada uno.

La mayoría de los escenarios registran en la bitácora (ver Apéndice B) la modificación de datos o parámetros utilizados por cada participante. Esto brinda la posibilidad de un posterior análisis sobre cuántos intentos fueron requeridos por cada grupo o participante individual y deja entrever las distintas estrategias adoptadas.

Todas las veces que se realizó el laboratorio de seguridad se solicitó a los participantes que completaran una encuesta sobre la cantidad de experiencias prácticas que realizaron en la carrera, la necesidad de realizar aún más actividades prácticas y su apreciación sobre los ejercicios o el entorno. El Apéndice A resume y analiza los datos recabados, en base a los cuales puede concluirse que en su gran mayoría, los participantes califican la experiencia como exitosa y beneficiosa debido a que logra que cada persona se adentre o profundice en los conceptos de seguridad de modo factible, considerando la cantidad de tiempo y los recursos disponibles.

La opinión común es que las materias deberían llevar a la práctica mayor cantidad de contenidos teóricos, que el entorno utilizado facilita el aprendizaje práctico y la experimentación, que los escenarios abordan aspectos relevantes, y que las experiencias adquiridas influirán en futuros trabajos a realizar tanto en la materia como –en algunos casos– en la vida profesional.

Capítulo 6

Discusión Final

Debido a todo el conocimiento de base requerido para adentrarse en seguridad informática, muchas universidades abordan la seguridad como foco central sólo en algunas materias electivas o en algunos módulos de otras materias hacia el final del programa. Así como el método científico es transversal a prácticamente toda actividad formativa en el área, la seguridad debería ser un eje temático en las carreras de grado a las que se está haciendo referencia. Los educadores y guías deberían enfocar los contenidos desde distintos ángulos, y uno de ellos debería ser la seguridad.

Las estadísticas sobre vulnerabilidades más frecuentes, así como las listas de errores más frecuentes deben ser consideradas por las instituciones educativas como una llamada de atención sobre la importancia de brindar conocimiento y herramientas eficaces a los educandos. Muchas empresas privadas toman como política la capacitación continua a sus empleados no sólo por brindarles beneficios no materiales además del salario, sino porque resulta más económico invertir en educación que en solucionar los problemas causados por malos diseños y errores introducidos por desconocimiento.

Los cursos de seguridad de la información deben cubrir una vasta cantidad de áreas y temas. En general, el tiempo disponible impide ahondar suficientemente en todas las áreas, por lo que muchos autores intentan confeccionar guías generales que puedan ser aplicadas en forma transversal, y que sean mayormente válidas para cada caso [1]. El tiempo y los recursos que requieren las actividades de experimentación como laboratorios son los principales motivos por los que muchas instituciones educativas no aumentan la cantidad de prácticas. No obstante, existen distintas formas de incluir actividades a lo

largo de la malla curricular que resulten en experimentación y aplicación práctica de las teorías [2, 4]. Es crucial seleccionar buenas técnicas pedagógicas para que los estudiantes puedan adquirir los conocimientos necesarios en un espacio de tiempo razonable.

Este trabajo está motivado por el hecho de que, en general, el tiempo del que disponen muchos instructores y profesores para preparar los cursos es menor al que requiere la adaptación, configuración y realización de experiencias prácticas. El presente trabajo tiene como objetivo brindar una herramienta práctica que por su simplicidad de instalación, mantenimiento y gestión así como por los escasos recursos que requiere, podría disminuir notoriamente el costo en tiempo y en materiales de realizar actividades prácticas en clase. Dichas actividades reportan valiosos beneficios sorteando dificultades encontradas por los instructores en las distintas técnicas de enseñanza [1–6].

La herramienta presentada se diferencia de otras en la simplicidad de preparación y administración del entorno, el cual se conforma principalmente de escenarios y de componentes destinados a la administración y recuperación de datos. Los escenarios permiten a los participantes llevar a la práctica nociones básicas y adentrarse en el área de seguridad. Cada escenario plantea un desafío distinto basado en las vulnerabilidades frecuentes mencionadas en las listas de OWASP y CWE/SANS de manera a abarcar situaciones actuales y de alto impacto, aunque uno de los escenarios propuestos plantea también situación real no cubierta por dichas listas, pero sí por algunas guías [20, 21].

En el diseño del entorno se consideró que el instructor puede agregar nuevos escenarios o modificar los existentes. Los escenarios deberán ser actualizados, ampliados o extendidos en la medida que las listas de OWASP y CWE/SANS sean actualizadas. Una forma alternativa de utilizar la herramienta propuesta es que el objetivo de los participantes sea crear o modificar escenarios y que los mismos sean probados por sus pares. Los autores –en su calidad de profesores de la cátedra Redes de Computadoras– realizaron experiencias que resultaron ser muy alentadoras tanto en las carreras de Ingeniería Electrónica como de Ingeniería Informática de la Universidad Católica “Nuestra Señora de la Asunción”.

Posibles trabajos futuros son:

- La inclusión de más escenarios aumentando la dificultad o abarcando vulnerabilidades todavía no incluidas, por ejemplo, la carga de archivos al servidor.

- Adaptar la herramienta para simplificar la creación de escenarios destinados a ataques de denegación de servicio.
- Agregar un componente que permita al participante, una vez que descubra la contraseña, acceder a una consola donde pueda ejecutar código SQL en la base de datos.
- Incluir escenarios cuyos objetivos sean que el participante deba modificar el código fuente para eliminar la vulnerabilidad.
- Programar componentes que faciliten la obtención de datos estadísticos a partir de la información de bitácora almacenada debido a las acciones realizadas por los participantes.
- Diseñar e implementar un esquema en el que se muestre información al participante a modo de pistas en la medida que los intentos por superar el nivel son fallidos.

Apéndice A

Impresión de los Estudiantes

El Capítulo 5 hace referencia a tres experiencias ya realizadas entre los años 2013 y 2015 en el ámbito de la cátedra Redes de Computadoras 2 con distintos alumnos que cursan la carrera Ingeniería Informática o Ingeniería Electrónica dictadas en la Universidad Católica "Nuestra Señora de la Asunción". Los alumnos que realizaron la experiencia en el año 2013 dieron su parecer sobre la metodología utilizada en clase y la necesidad no satisfecha que tienen de aplicar los contenidos teóricos. Estos comentarios fueron realizados en forma oral por lo que no quedaron formalmente registrados o resumidos.

A.1. Experiencia realizada en diciembre 2014

En la segunda experiencia realizada a finales del año 2014 –en cuya fecha ya habían sido iniciadas actividades relacionadas al presente trabajo– los alumnos que participaron tuvieron la oportunidad de completar un formulario que contenía las siguientes preguntas:

- ¿Cuántas materias aproximadamente cursaste en la carrera?
- ¿Cuántas de las materias que cursaste incluyeron temas de seguridad de la información (en forma teórica o práctica)?
- ¿El laboratorio de seguridad realizado en la materia Redes de Computadoras 2 dio la oportunidad de poner en práctica conceptos teóricos aprendidos en distintas materias?

- ¿Por qué crees que en general pocas materias realizan laboratorios prácticos en clase?

Las dos primeras preguntas permitió calcular que aproximadamente el 7 % de las materias hasta esa fecha cursadas, incluyeron contenidos relacionados a la seguridad de la información. La tercera pregunta fue respondida afirmativamente por todos los participantes indicando esto que la experiencia realizada pudo llevar a la práctica no solo conceptos aprendidos en la cátedra dentro de la cual se estaba llevando a cabo la actividad, sino también de conceptos aprendidos en otras materias.

Las respuestas que dieron los alumnos a la cuarta pregunta fueron catalogadas dentro de cuatro grupos según los motivos de base, de forma a poder concluir cuál era el mayor obstáculo según la perspectiva de los alumnos. A continuación se encuentran los grupos y el porcentaje de alumnos que tomaron uno de los motivos principales:

Motivo	**Porcentaje**
Falta de tiempo (horas de clase)	56 %
Alto esfuerzo de mantenimiento y actualización	11 %
Complejidad para llevar a cabo la práctica	11 %
Otros	22 %

Cuadro A.1: Motivos por los que pocas materias realizan laboratorios prácticos en clase según los alumnos

A.2. Experiencia realizada en agosto 2015

La tercera experiencia fue realizada durante agosto del año 2015. El entorno de experimentación propuesto en este trabajo fue utilizado por su autor y 18 alumnos de la cátedra Redes de Computadoras 2. Durante la experiencia, las tareas de administración

fueron simplificadas por el entorno debido principalmente a algunas de las funcionalidades y conceptos incorporados pocos meses antes de agosto, entre ellos, la utilización de sesiones, la capacidad de restauración de datos gestionada por el usuario, y el aislamiento en base de datos al dotar a cada participante de un conjunto propio de usuario, tablas y permisos evitando así que las acciones de un participante afecten a los demás.

Los alumnos completaron un formulario cuyo contenido se resume en el Cuadro A.2. En lo referente a cantidad de materias cursadas o con experiencias prácticas, se solicitó un dato aproximado ya que los alumnos no llevan la cuenta exacta de memoria. A partir del tercer punto la valoración debía realizarse en una escala del 1 (mínimo o en desacuerdo) a 5 (máximo o de acuerdo).

Aspecto a valorar	**Promedio en escala 1 (mínimo) a 5 (máximo)**
Cantidad aproximada de materias cursadas en la carrera	40 materias
Cantidad aproximada de experiencias prácticas ya realizadas en la carrera	4 materias
Las materias deberían llevar a la práctica mayor cantidad de contenidos teóricos	3,47
El entorno utilizado facilita el aprendizaje práctico y la experimentación	3,50
Los escenarios abordan aspectos relevantes	3,71
Este laboratorio aportó experiencias que influirán en futuros trabajos a realizar	3,71

Cuadro A.2: Apreciación de los alumnos que participaron en la experiencia realizada durante agosto del año 2015

Como fue indicado anteriormente, este trabajo está motivado por el hecho de que muchos instructores y profesores se dedican a la enseñanza como una actividad laboral relevante intelectualmente, pero no económicamente. Es decir que el tiempo dedicado a la preparación de los cursos en los que participan en general es reducido y que la adaptación, configuración y realización de experiencias prácticas excede al tiempo disponible. Además, otros aspectos a considerar son los requerimientos de equipos y configuración.

La valoración dada por los participantes en el Cuadro A.2 puede interpretarse como señal de que el entorno aún necesita mejorar y que ya en su estado actual es una herramienta beneficiosa que permite aprender a través de la experimentación llevando a la práctica conocimientos teóricos.

Apéndice B

Esquema de Base de Datos

En la Figura B.1 pueden observarse las tablas y relaciones que conforman el esquema de base de datos. La base de datos tiene dos funciones principales: almacenar en forma de bitácora las acciones relevantes realizadas por los participantes, y almacenar los datos utilizados en los distintos escenarios.

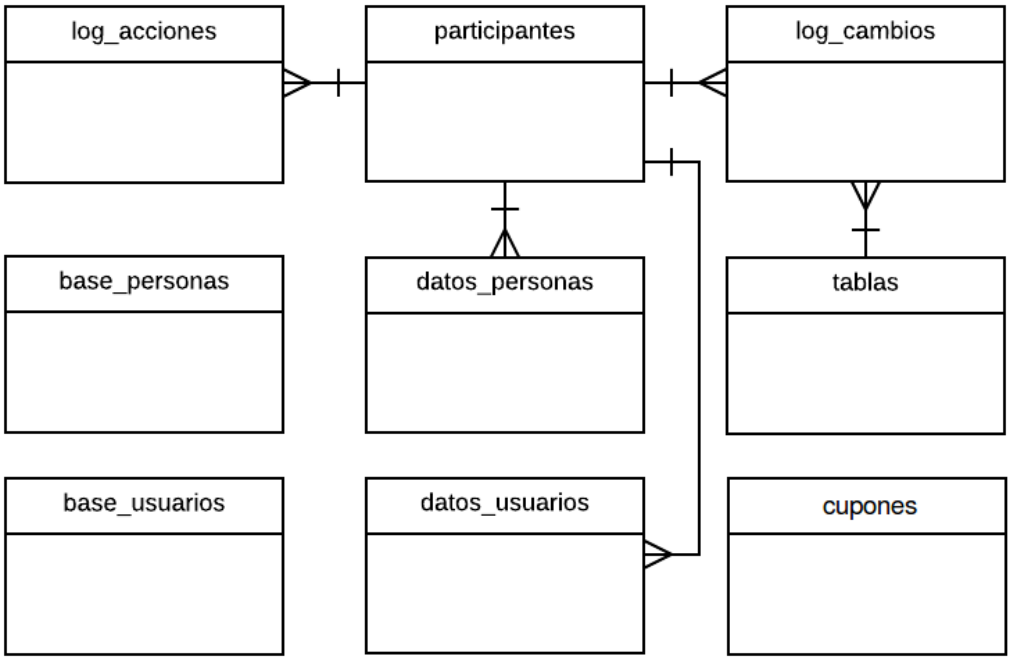

Figura B.1: Diagrama entidad relación simplificado.

A continuación se describe la función y composición de cada una de las tablas.

`Participantes`. La tabla contiene información sobre las personas o grupos que realizan las experiencias. Algunas columnas brindan información sobre el participante y otras son utilizadas en el momento de autenticación y para mantener una sesión.

`Log_acciones`. La tabla almacena acciones relevantes realizadas por los participantes como su autenticación en el entorno de experimentación y los intentos de inyección de código cuando debido a las características del escenario la información no sería almacenada en otra tabla.

`Log_cambios`. La tabla es utilizada para registrar las modificaciones que los participantes realizan a los datos. Algunos escenarios requieren la inyección de código persistente, que es almacenado en la base de datos entre los campos de ciertas tablas (*e.g.* tabla `personas`), en tal caso esta tabla registra el valor original y el actualizado.

`Datos_personas`. La tabla contiene un conjunto de datos para cada participantes de forma tal que los demás no se vean afectados por los cambios introducidos en los campos destinados a la identidad de las personas.

`Datos_usuarios`. La tabla cumple la misma función que `datos_personas` pero sobre usuarios. Cabe mencionar que esta tabla no es utilizada por el entorno más que para simular una posible tabla existente en sistemas vulnerables.

Las tablas `base_personas` y `base_usuarios` almacenan los datos originales que son insertados en `datos_personas` y `datos_usuarios` por cada nuevo participante que se registra en la plataforma. Los datos base son también utilizados cuando se acciona un mecanismo de restauración para un participante dado.

`Tablas`. La tabla contiene un registro por cada tabla y columna existente en el esquema de forma tal a que en la tabla `log_cambios` se especifique qué tabla (registro completo) o campo (sólo el campo indicado de la tabla dada) fue modificado.

Por simplicidad del diagrama las tablas `datos_personas` y `datos_usuarios` figuran una sola vez; sin embargo, dicho par de tablas es creado por cada participante agregando el número que lo identifica como sufijo al nombre de tabla. Esta técnica permite aislar las experiencias a través de permisos por usuario.

`Cupones`. La tabla contiene los datos de cupones utilizados en el octavo escenario (ver Sección 4.1.1.8) y es utilizada únicamente con operaciones de lectura.

A continuación se transcribe el código SQL de creación del esquema, utilizando la simplificación arriba mencionada.

```
1  CREATE TABLE 'participantes' (
2  'id' serial NOT NULL,
3  'nombre' character varying NOT NULL,
4  'contrasenha' character varying NOT NULL,
5  'mail' character varying NOT NULL,
6  'admin' boolean NOT NULL
7  ) WITHOUT OIDS;
8  ALTER TABLE 'participantes' ADD CONSTRAINT 'participantes_pk
       ' PRIMARY KEY('id');
9
10 CREATE TABLE 'base_usuarios' (
11 'id' serial NOT NULL,
12 'nickname' character varying NOT NULL,
13 'contrasenha' character varying NOT NULL,
14 'mail' character varying NOT NULL
15 ) WITHOUT OIDS;
16 ALTER TABLE 'base_usuarios' ADD CONSTRAINT 'base_usuarios_pk
       ' PRIMARY KEY('id');
17
18 CREATE TABLE 'datos_usuarios' (
19 'id' serial NOT NULL,
20 'participante_id' bigint NOT NULL,
21 'nickname' character varying NOT NULL,
22 'contrasenha' character varying NOT NULL,
23 'mail' character varying NOT NULL
24 ) WITHOUT OIDS;
25 ALTER TABLE 'datos_usuarios' ADD CONSTRAINT '
       datos_usuarios_pk' PRIMARY KEY('id');
26
27 CREATE TABLE 'base_personas' (
28 'id' serial NOT NULL,
```

```
29 'nombre' character varying NOT NULL,
30 'apellido' character varying NOT NULL,
31 'telefono' character varying NOT NULL
32 ) WITHOUT OIDS;
33 ALTER TABLE 'base_personas' ADD CONSTRAINT 'base_personas_pk
      ' PRIMARY KEY('id');
34
35 CREATE TABLE 'datos_personas' (
36 'id' serial NOT NULL,
37 'participante_id' bigint NOT NULL,
38 'nombre' character varying NOT NULL,
39 'apellido' character varying NOT NULL,
40 'telefono' character varying NOT NULL
41 ) WITHOUT OIDS;
42 ALTER TABLE 'datos_personas' ADD CONSTRAINT '
      datos_personas_pk' PRIMARY KEY('id');
43
44 CREATE TABLE 'log_login' (
45 'id' serial NOT NULL,
46 'participante_id' bigint NOT NULL,
47 'nombre_usuario' character varying,
48 'contrasenha' character varying,
49 'exito' boolean NOT NULL,
50 'ip' character varying NOT NULL,
51 'fecha' timestamp NOT NULL,
52 'observacion' character varying
53 ) WITHOUT OIDS;
54 ALTER TABLE 'log_login' ADD CONSTRAINT 'log_login_pk'
      PRIMARY KEY('id');
55
56 CREATE TABLE 'tablas' (
57 'id' character varying NOT NULL
58 ) WITHOUT OIDS;
```

```
59 ALTER TABLE 'tablas' ADD CONSTRAINT 'tablas_pk' PRIMARY KEY(
       'id');
60
61 CREATE TABLE 'log_cambios' (
62 'id' serial NOT NULL,
63 'participante_id' bigint NOT NULL,
64 'tabla_id' character varying NOT NULL,
65 'valor_anterior' character varying NOT NULL,
66 'valor_nuevo' character varying NOT NULL,
67 'fecha' timestamp NOT NULL,
68 'ip' character varying NOT NULL
69 ) WITHOUT OIDS;
70 ALTER TABLE 'log_cambios' ADD CONSTRAINT 'log_cambios_pk'
       PRIMARY KEY('id');
71
72 CREATE TABLE 'cupones' (
73 'id' serial NOT NULL,
74 'codigo' character varying NOT NULL,
75 'monto' character varying NOT NULL,
76 'descripcion' character varying NOT NULL
77 ) WITHOUT OIDS;
78 ALTER TABLE 'cupone' ADD CONSTRAINT 'cupones_pk' PRIMARY KEY
       ('id');
79
80
81 ALTER TABLE 'log_cambios' ADD CONSTRAINT 'log_cambios_tabla'
        FOREIGN KEY ('tabla_id') REFERENCES 'tablas'('id') ON
       UPDATE RESTRICT ON DELETE RESTRICT;
82 ALTER TABLE 'datos_personas' ADD CONSTRAINT '
       datos_personas_participantes' FOREIGN KEY ('
       participante_id') REFERENCES 'participantes'('id') ON
       UPDATE RESTRICT ON DELETE RESTRICT;
```

```
83 ALTER TABLE 'datos_usuarios' ADD CONSTRAINT '
     datos_usuarios_participantes' FOREIGN KEY ('
     participante_id') REFERENCES 'participantes'('id') ON
     UPDATE RESTRICT ON DELETE RESTRICT;
84 ALTER TABLE 'log_login' ADD CONSTRAINT '
     log_login_participantes' FOREIGN KEY ('participante_id')
     REFERENCES 'participantes'('id') ON UPDATE RESTRICT ON
     DELETE RESTRICT;
85 ALTER TABLE 'log_cambios' ADD CONSTRAINT '
     log_cambios_participantes' FOREIGN KEY ('participante_id'
     ) REFERENCES 'participantes'('id') ON UPDATE RESTRICT ON
     DELETE RESTRICT;
```

Apéndice C

Código Fuente Relevante

La herramienta propuesta permite crear niveles o escenarios según las necesidades del instructor. En particular, este trabajo implementa ocho escenarios que sirven como ejemplo de una gama mayor y más compleja de escenarios que podrían plantearse. A continuación se muestran trozos de código extraídos de los distintos escenarios.

C.1. Escenario 1

El primer escenario se centra en explotar la vulnerabilidad en el método de autenticación. El objetivo planteado es ingresar sin conocer ni adivinar usuario y contraseña.

Texto removido con fines educativos para no revelar la respuesta. Consultas: ing.uriyael@gmail.com

C.2. Escenario 2

El segundo escenario tiene como objetivo que cuando un cliente cargue la página reciba una alerta indicando un mensaje dejado por el atacante como prueba de éxito. La página recupera los datos guardados en la base de datos sobre personas. El resultado es mostrado en forma de tabla incluyendo un botón para realizar la edición de los datos.

Texto removido con fines educativos para no revelar la respuesta. Consultas: ing.uriyael@gmail.com

C.3. Escenario 3

El tercer escenario tiene como objetivo redirigir a los clientes que carguen la página a un sitio elegido por el atacante. La página crea una tabla donde se muestran los datos almacenados en la base de datos. Para modificar los datos de una persona se utiliza un formulario con componentes HTML. Abajo se encuentra un ejemplo de código en lenguaje JavaScript que permitiría superar el escenario debido a que no se realiza una limpieza de los datos que ingresan o salen de la base de datos.

Texto removido con fines educativos para no revelar la respuesta. Consultas: ing.uriyael@gmail.com

C.4. Escenario 4

El cuarto escenario se basa en que el usuario tenga acceso a modificar el comportamiento del sistema saliendo de las acciones permitidas. La acción del formulario que se encuentra en la interfaz inicial (ver Figura 4.5) es cargar la página cuyo código se muestra a continuación.

Texto removido con fines educativos para no revelar la respuesta. Consultas: ing.uriyael@gmail.com

C.5. Escenario 5

El quinto escenario ejemplifica los riesgos de que haya archivos en ubicaciones accesibles al usuario, cuya extensión no tienen un significado para el servidor web. En general, los servidores web son configurados para descargar al cliente los archivos cuya extensión no está entre aquellas a interpretar.

Para superar el escenario, el participante debe encontrar la copia de respaldo hecha del archivo `conexion.php`. Si bien el participante podría utilizar algún método de fuerza bruta para encontrar estos archivos, se espera que entre los primeros intentos manuales se encuentre alguno de los siguientes archivos:

Texto removido con fines educativos para no revelar la respuesta. Consultas: ing.uriyael@gmail.com

C.6. Escenario 6

El sexto escenario tiene como objetivo eliminar datos del sistema atacado. El ataque es realizado desde las mismas interfaces mostradas en las Figuras 4.2 y 4.3 en las que se lista a las personas existentes en la tabla `datos_personas` y se permite la modificación de algunos campos para una persona seleccionada.

Texto removido con fines educativos para no revelar la respuesta. Consultas: ing.uriyael@gmail.com

C.7. Escenario 7

El séptimo escenario tiene como objetivo robar a la víctima datos de las *cookies*. El ataque es realizado desde las mismas interfaces mostradas en las Figuras 4.2 y 4.3 en las que se lista a las personas existentes en la tabla `datos_personas` y se permite la modificación de algunos campos para una persona seleccionada.

Para modificar los datos de una persona se utiliza un formulario con componentes HTML. Abajo se encuentra un ejemplo de código en lenguaje JavaScript que permitiría superar el escenario debido a que no se realiza una limpieza de los datos que ingresan o salen de la base de datos.

Texto removido con fines educativos para no revelar la respuesta. Consultas: ing.uriyael@gmail.com

C.8. Escenario 8

El octavo escenario ejemplifica un tipo de vulnerabilidad relacionada a la aleatoriedad de los datos ya sea por errores de diseño, de configuración o de implementación. Para ello, la interfaz de usuario (ver Figura 4.6) contiene un botón con el que se obtiene en forma aleatoria un cupón de los almacenados en la tabla `cupones` de la base de datos y otro para consultar la existencia de códigos.

Para completar con éxito el ejercicio, el participante debe encontrar el cupón cuyo valor es de un millón de Dólares. Quienes emiten los cupones deberían asegurarse de que existe muy baja probabilidad de que la elección de un código al azar coincida con un cupón real. Es razonable entonces que existan muchas más combinaciones válidas que

la cantidad de cupones, y que dichos cupones tengan un código asignado en forma aleatoria. Si no existen tendencias o parcialidades, entonces conocer códigos válidos no revela información sobre otros códigos existentes.

Texto removido con fines educativos para no revelar la respuesta. Consultas: ing.uriyael@gmail.com

Apéndice D

Ranking de Errores Más Peligrosos CWE/SANS

Se presenta la lista[1] de los 25 errores más peligrosos en el software creada por SANS y CWE. El listado utiliza el sistema de puntaje denominado CWSS.

Puesto	Puntaje	ID	Nombre
01	93,8	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
02	83,3	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
03	79,0	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
04	77,7	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
05	76,9	CWE-306	Missing Authentication for Critical Function
06	76,8	CWE-862	Missing Authorization
07	75,0	CWE-798	Use of Hard-coded Credentials
08	75,0	CWE-311	Missing Encryption of Sensitive Data

[1]El presente documento utiliza la lista al año 2011, no existiendo una publicación más actualizada a la fecha de este trabajo. La clasificación es mantenida en la dirección http://cwe.mitre.org/top25.

09	74,0	CWE-434	Unrestricted Upload of File with Dangerous Type
10	73,8	CWE-807	Reliance on Untrusted Inputs in a Security Decision
11	73,1	CWE-250	Execution with Unnecessary Privileges
12	70,1	CWE-352	Cross-Site Request Forgery (CSRF)
13	69,3	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
14	68,5	CWE-494	Download of Code Without Integrity Check
15	67,8	CWE-863	Incorrect Authorization
16	66,0	CWE-829	Inclusion of Functionality from Untrusted Control Sphere
17	65,5	CWE-732	Incorrect Permission Assignment for Critical Resource
18	64,6	CWE-676	Use of Potentially Dangerous Function
19	64,1	CWE-327	Use of a Broken or Risky Cryptographic Algorithm
20	62,4	CWE-131	Incorrect Calculation of Buffer Size
21	61,5	CWE-307	Improper Restriction of Excessive Authentication Attempts
22	61,1	CWE-601	URL Redirection to Untrusted Site ('Open Redirect')
23	61,0	CWE-134	Uncontrolled Format String
24	60,3	CWE-190	Integer Overflow or Wraparound
25	59,9	CWE-759	Use of a One-Way Hash without a Salt

Cuadro D.1: Ranking de Errores Más Peligrosos CWE/SANS

Apéndice E

Ranking de riesgos de OWASP

A continuación se encuentra la versión para el año 2013 de la lista de 10 riesgos de seguridad más críticos en aplicaciones web mantenida por OWASP, no existiendo una publicación más actualizada a la fecha de este trabajo.

Puesto	Nombre
A1	Inyección
A2	Déficit en autenticación y manejo de sesiones
A3	Cross-Site Scripting (XSS)
A4	Referencias inseguras a objetos en forma directa
A5	Mala configuración de seguridad
A6	Exposición de información sensible
A7	Falta de control de nivel de acceso
A8	Cross-Site Request Forgery (CSRF)
A9	Utilización de componentes con vulnerabilidades conocidas
A10	No validar las redirecciones y reenvíos

Cuadro E.1: 10 riesgos de seguridad más críticos en aplicaciones web según OWASP

En [19] se provee información general sobre la probabilidad o riesgo de explotación y el impacto técnico por cada riesgo de la lista. El posicionamiento utilizó una metodología basada en *OWASP Risk Rating Methodology* que define el riesgo como el producto entre la probabilidad de ocurrencia y el impacto. Para ello, se realizan los siguientes pasos:

- Identificar el riesgo recolectando la información necesaria sobre los componentes involucrados, los vectores de ataque que pueden ser utilizados y el impacto en caso de una explotación exitosa.
- Estimar la probabilidad a través de asignar valores entre 0 y 9 a: el nivel de habilidad que requeriría un atacante para explotar la vulnerabilidad, el nivel de motivación que tienen los posibles atacantes, los recursos necesarios para llevar adelante la explotación, la cantidad de personas que conforman los posibles grupos de atacantes.
- Estimar el impacto técnico (pérdida de confidencialidad, integridad, disponibilidad, trazabilidad de las acciones a un individuo), impacto del negocio (daño financiero, de reputación, grado de incumplimiento del servicio o violación de privacidad).
- Determinar la severidad del riesgo (dividiendo la escala del 0 al 9 en bajo, medio o alto) según el rango en que se encuentren las probabilidades y los niveles de impacto.
- Decidir qué solucionar.
- Adaptar el modelo de valoración del riesgo al negocio para lograr un mejor aprovechamiento.

Bibliografía

[1] William Yurcik and David Doss. Different approaches in the teaching of information systems security. In *Proceedings of the Information Systems Education Conference*, 2001.

[2] Jacob West and Matt Bishop. Security education for the new generation. In *Proc. RSA Conference (2014)*, 2014.

[3] Frank Schindler. Coping with security in programming. *Acta Polytechnica Hungarica*, 3(2):65–72, 2006.

[4] Matt Bishop and BJ Orvis. A clinic to teach good programming practices. In *Proceedings of the 10th Colloquium for Information Systems Security Education*, pages 168–1174, 2006.

[5] Manar Abu Talib, Adel Khelifi, and Leon Jololian. Secure software engineering: A new teaching perspective based on the swebok. *Interdisciplinary Journal of Information, Knowledge, and Management*, 5:83–99, 2010.

[6] NESG Network Engineering & Security Group. *NETA: A NETwork Attacks Framework. Architecture and Usage*. University of Granada, 2013.

[7] Michael Kölling. The problem of teaching object-oriented programming, part 1: Languages. *Journal of Object-oriented programming*, 11(8):8–15, 1999.

[8] Michael Kölling. The problem of teaching object-oriented programming, part 2: Environments. *Journal of Object-Oriented Programming*, 11(9):6–12, 1999.

[9] Belk, Coles, Goldschmidt, Howard, Randolph, Saario, Sondhi, Tarandach, Vähä-Sipilä, and Yonchev. Fundamental practices for secure software development. SAFECode, 2011.

[10] Bob Martin, Mason Brown, Alan Paller, Dennis Kirby, and Steve Christey. 2011 cwe/sans top 25 most dangerous software errors. *Common Weakness Enumeration*, 7515, 2011.

[11] Maximillian Dornseif, Felix C Gärtner, Thorsten Holz, and Martin Mink. An offensive approach to teaching information security:"aachen summer school applied it security". Technical report, Technical Report AIB-2005-02, Aachen, 2005.

[12] Fred Gutierrez. Stingray: a hands-on approach to learning information security. In *Proceedings of the 7th conference on Information technology education*, pages 53–58. ACM, 2006.

[13] Kurt Squire. Changing the game: What happens when video games enter the classroom? *Innovate: Journal of Online Education*, 1(6), 2005.

[14] Dino Schweitzer, Dan Gibson, and Michael Collins. Active learning in the security classroom. In *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on*, pages 1–8. IEEE, 2009.

[15] Benjamin D Cone, Cynthia E Irvine, Michael F Thompson, and Thuy D Nguyen. A video game for cyber security training and awareness. *computers & security*, 26 (1):63–72, 2007.

[16] P. Kim. *The Hacker Playbook: Practical Guide to Penetration Testing*. Createspace Independent Pub, 2014. ISBN 9781494932633. URL https://books.google.com.py/books?id=iEKJoAEACAAJ.

[17] Alain Abran and Pierre Bourque. *SWEBOK: Guide to the software engineering Body of Knowledge*. IEEE Computer Society, 2014.

[18] Apple Inc. Secure coding guide. In *Secure Coding Guide*, 2014.

[19] OWASP Foundation. OWASP top 10 - 2013. Technical report, OWASP Foundation, 2013. URL https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.

[20] Matthew Heckathorn. Network monitoring for web-based threats. Technical report, DTIC Document, 2011.

[21] OWASP Foundation. OWASP testing guide v4. Technical report, OWASP Foundation, 2014. URL https://www.owasp.org/index.php/OWASP_Testing_Project.

[22] Christof Paar and Jan Pelzl. URL http://dx.doi.org/10.1007/978-3-642-04101-3.

[23] Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno. *Cryptography Engineering: Design Principles and Practical Applications*. Wiley Publishing, 2010. ISBN 0470474246, 9780470474242.