UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

DESARROLLO DE PLATAFORMA PARA EL MONTAJE AUTOMÁTICO DE HONEYNETS PARA LABORATORIOS DE CIBERSEGURIDAD

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

SERGIO ROBERTO IGNACIO LEIVA VALDEBENITO

PROFESOR GUÍA:
ALEJANDRO HEVIA ANGULO

MIEMBROS DE LA COMISIÓN:
ÉRIC TANTER
JAVIER BUSTOS JIMÉNEZ

SANTIAGO DE CHILE
2020

Resumen

Las honeynets son redes de computadores con vulnerabilidades conocidas, creadas con el propósito de "atrapar" ataques provenientes del exterior. Si bien tienen vulnerabilidades conocidas y, a veces, evidentes, su finalidad es monitorear de qué manera son atacadas, de forma de estudiar cómo un atacante intenta penetrar una red, o comprometer una de sus máquinas.

Las honeynets también se pueden utilizar para otros fines. Las máquinas, al tener vulnerabilidades conocidas, pueden utilizarse como terreno de prueba para que estudiantes de carreras relacionadas a la computación aprendan cómo atacar y defender sistemas sin tener que hacerlo sobre máquinas reales, o que se encuentren en entornos de producción.

En el siguiente informe se describe una plataforma desarrollada para montar automáticamente redes de máquinas vulnerables, permitiendo a sus usuarios configurar tanto las máquinas que componen la honeynet, como las redes que las conectan, con la finalidad de que los estudiantes puedan descubrir vulnerabilidades tanto en las máquinas, como servicios ocultos o de uso interno de la red.

La plataforma actúa como apoyo a la docencia tradicional, y busca disminuir la carga de tiempo al configurar un laboratorio por métodos más tradicionales. Además, permite monitorear en tiempo real la actividad de los estudiantes durante el laboratorio, con la finalidad de entregarles retroalimentación de forma más rápida, y corregirlos durante la misma sesión, para ayudarlos durante su proceso de aprendizaje. La plataforma también permite acceder a los datos de sesiones anteriores, para revisar la actividad si es necesario.

Como plataforma de apoyo a la docencia, la solución descrita fue probada en un escenario con estudiantes del Departamento de Ciencias de la Computación de la Universidad de Chile. La sesión, que se hizo de forma online, contó con la participación de 45 estudiantes. La percepción en general fue positiva, y se destaca que, según la opinión de los participantes, acompañar las sesiones teóricas con las prácticas ayuda en su proceso de aprendizaje y disminuye sus dudas posteriores. Se destaca también que todos indicaron que participarían en una sesión similar a futuro.

Se concluye que la solución desarrollada genera valor tanto para los estudiantes como para los profesores, a los primeros como un apoyo en el aprendizaje, junto a las clases tradicionales, y a los segundos al disminuir la carga tanto en la planificación de las clases prácticas, como en la obtención de resultados en tiempo real, para mejorar los métodos educativos de ser necesario, y para saber más rápidamente qué están haciendo los estudiantes.

Agradecimientos

Me gustaría agradecer a todas las personas que han hecho mi paso por la Universidad algo más placentero. Es posible que se me olvide gente, pero intenté ser exhaustivo:

- A mis padres, Orieta y José, que me apoyaron desde que decidí venirme a estudiar lejos de casa, y que me acompañaron a la distancia.
- A mis amigues de sección, que fueron y siguen siendo importantes: Andrés, Caro, Fran, Gabi, Nico y Tiare.
- A mis *frens* que conocí en el DCC y que me acompañaron durante la carrear: Barbi, Mati, Jordo, Tito, Dani, Sven, Sebastián(es), Pablo, Franquito, Pancho, Cristóbal y Tato.
- A Nico Varas y a Hiho, que me motivaron a hacer cosas computinas fuera de lo necesario por la carrera. Me enseñaron y motivaron mucho para darle sentido a las cosas que aprendía.
- A les profes que confiaron en mí y me dejaron participar en sus cuerpos docentes: Nancy, María Cecilia, Aidan, Jocelyn y Jeremy.
- A los equipos del CaDCC 2017 y 2018. Estaba escribiendo la lista, pero estaba quedando demasiado larga.
- A Eduardo, que me motivó a escoger este tema para mi memoria.
- A Alejandro Hevia, mi profe guía, que me aportó cada vez que hablamos, y que fue comprensivo con las cosas que han pasado el último año a nivel país y mundial.
- Finalmente, me gustaría agradecerle a Tamy, que ha sido mi compañera de vida por los últimos 6 años. Por acompañarme en mis momentos difíciles y los no tanto. Por darme consejos cuando lo necesitaba, y apoyarme en todo.

Tabla de Contenido

Índice de Ilustraciones

Introducción

Un sistema de información es un sistema diseñado para recolectar, procesar, almacenar y distribuir información. La información almacenada es valiosa no solo para las personas que manejan el sistema, si no que también para quienes lo usan, así como también para posibles entidades externas. Un sistema puede ser atacado por alguien que quiere acceder sin autorización a esta información, por lo que es necesario protegerla.

Los sistemas informáticos son en general bastante complejos, por lo que resulta imposible afirmar que un sistema no puede ser vulnerado. Debido a lo anterior, es importante que quienes administran los sistemas estén preparados ante cualquier posible ataque externo. Sin embargo, por la misma lógica, es imposible estar preparados para cualquier ataque, por lo que es importante saber de qué formas en general pueden ocurrir, y estudiar técnicas lo más pronto posible, antes de que se utilicen contra sistemas reales, y con información considerada como crítica.

Para comprender mejor cómo defender un sistema, es necesario saber de qué forma este puede ser atacado. Por ello, resulta importante que las personas que estudian carreras relacionadas tengan conocimientos de ciberseguridad. Como estos temas tienen aplicaciones y consecuencias en sistemas informáticos usados en producción, es importante que esta formación cuente también con elementos prácticos. Se podría, por ejemplo, hacer que un grupo de estudiantes intente atacar sistemas reales, o revisar el código fuente de algún programa de código abierto, en busca de alguna vulnerabilidad. Si bien ambas opciones tienen el beneficio directo de ayudar en sistemas reales, tienen el problema de que nada asegura que estas vulnerabilidades se pueden encontrar con una dificultad adecuada para los estudiantes, ya que no corresponden a escenarios controlados. Esta es una de las razones por las que resulta atractivo utilizar *honeypots*, que si bien no son utilizadas solamente con una finalidad docente, se pueden aplicar para ello.

Una *honeypot digital*, como su nombre lo indica, es un "tarro de miel", del mundo digital. También conocidas como sistema trampa o señuelo, las *honeypot* son sistemas que simulan ser vulnerables a ciertos ataques, pero que en realidad se utilizan para estudiar cómo alguien intenta atacarlas.

Este concepto se puede extender a lo que es una *honeynet*: en lugar de actuar como un sistema o máquina que es vulnerable a ataques, simula ser una red completa de computadores y dispositivos conectados, también vulnerables a ataques.

El uso de *honeypots* y *honeynets* para investigar y detectar de qué manera se intenta atacar

sistemas informáticos cobra importancia porque, como se dijo anteriormente, estos sistemas reciben ataques, y lo ideal es detectarlos lo más prontamente posible. De esta manera, es más fácil tomar medidas y evitar que el sistema se vea comprometido [3]. La utilización de ambos mecanismos, sin embargo, tiene un par de problemas que no son tan fáciles de resolver.

Uno de los problemas al levantar *honeypots* o *honeynets* es la responsabilidad ante la ley [32]. Si es que uno de estos sistemas, que se sabe que son vulnerables se ve comprometido por un ente malicioso, este lo puede utilizar para atacar a otro sistema, usándolo como proxy para ataques, o haciéndolo parte de una *botnet*. Si bien quienes montan originalmente estos sistemas lo pueden hacer con la intención de investigar, al hacerlos vulnerables a propósito, se deben hacer responsables de su mal uso.

Otro problema posible es que debido a que la mayoría de las *honeypots* y *honeynets* montadas utilizan tecnologías similares, o software que actúa como ellas, existen herramientas que permiten detectar automáticamente si es que una IP está corriendo una *honeypot* [31]. Si alguien está atacando direcciones IP al azar, y sospecha que una dirección corresponde a una *honeypot*, lo más probable es que deje de atacarla. Esto último se puede evitar si es que, por ejemplo, se utilizan servicios como los de Amazon para montar la aplicación en servidores e IPs públicas distintas. Sin embargo, esto podría estar en contra de los términos y condiciones de Amazon Web Services, que en su sección 6 hablan de "no poner en un riesgo de seguridad a sus servicios o una tercera parte" [30].

Más recientemente, y por las razones anteriores, se ha estado estudiando el uso de *honeynets* para enseñar ciberseguridad de manera práctica. Utilizando las tecnologías antes mencionadas, esta estrategia permite enseñar ciberseguridad de manera práctica, como complemento o reemplazo de cursos teóricos, por ejemplo.

Las honeypots se clasifican según el número de requests que reciben (concurrencia), y el nivel de interacción y análisis que ofrecen. Según esto hay honeypots de baja interacción (o Low Interaction, abreviado LI), alta interacción (HI), bajo volumen (LV) y alto volumen (HV). Una honeypot con alta interacción y bajo volumen, por ejemplo, se clasifica como *HILV* [19]. Para el estudio práctico de ciberseguridad, se buscará usar honeypots HILV, porque deben permitir utilizar varios servicios, pero no requieren alta concurrencia, pues solo se conectan a ella un grupo reducido de estudiantes.

Planteamiento del problema

Los conocimientos sobre ciberseguridad permiten, entre otras cosas, detectar problemas de seguridad en sistemas informáticos reales. Todos los sistemas informáticos conectados a una red pueden ser atacados por terceras personas, y hoy en día la mayoría de los sistemas se conectan a internet para obtener y enviar datos.

Por lo anterior, es importante que las y los estudiantes de carreras relacionadas a la computación tengan en su formación materia relacionada a la ciberseguridad, y que esta formación sea no sólo teórica, si no que también práctica, de manera de acercarse a los escenarios que podrían encontrarse en su vida profesional.

Los laboratorios de ciberseguridad buscan formar de forma práctica a estudiantes de carreras de computación. Debido a que cada sistema informático es distinto, y corre distinto software, surge la necesidad de montar laboratorios con sistemas de distintas características, de la manera más fácil posible para disminuir la carga del diseño de escenarios de aprendizaje.

Además del montaje de los laboratorios de manera dinámica, es interesante monitorear qué mecanismos están utilizando los estudiantes en el laboratorio para vulnerar los sistemas en cuestión, con la finalidad de retroalimentarlos durante y en forma posterior a esta actividad docente.

De esta forma en el siguiente trabajo se plantea como problema principal el desarrollo de un sistema que permita montar laboratorios de ciberseguridad de manera rápida y dinámica, además de monitorear la actividad de los estudiantes que asisten al laboratorio.

Relevancia

La relevancia del sistema desarrollado surge de la necesidad de enseñar ciberseguridad de manera más efectiva a los estudiantes. Anteriormente se han utilizado sistemas parecidos para enseñar sobre ciberseguridad, obteniendo una buena retroalimentación de parte de los estudiantes [8]. Por lo tanto, se espera que el trabajo sea una herramienta útil para profesores que busquen enseñar sobre ciberseguridad.

Otro elemento que le aporta relevancia al sistema desarrollado es que, si bien hay experiencias parecidas con sistemas similares, dichos sistemas no fueron liberados al público, o ya no se encuentran disponibles.

Finalmente, se reconoce la importancia que tiene para los estudiantes recibir retroalimentación sobre su proceso de resolución de problemas de manera oportuna [13], de manera de que puedan afrontarlos de manera más efectiva, así como la importancia para el docente de conocer la misma información, para refinar o cambiar la metodología de enseñanza si es necesario.

Objetivos

Objetivo General

El objetivo de esta memoria es desarrollar una plataforma que permita configurar y montar automáticamente redes de máquinas virtuales vulnerables, que permitan enseñar de forma práctica habilidades de ciberseguridad.

Objetivos Específicos

Además del objetivo general, se espera que la plataforma cumpla con las siguientes características:

1. Las máquinas utilizadas deben ser configurables, de manera de poder montar máquinas con distinto software.

2. Las redes entre las máquinas de un escenario deben poder configurarse desde la misma plataforma.
3. Los escenarios configurados utilizando la plataforma deberían poder montarse en otro ambiente de manera fácil, de manera que los estudiantes puedan montarlas en sus computadores.
4. La actividad de los estudiantes que se conecten a la red debe poder ser monitoreada, con la finalidad de observar de qué manera intentan vulnerar el sistema.

Descripción general de la solución

El sistema desarrollado cuenta con tres módulos principales, que se describen de forma general a continuación:

Gestión de máquinas

Para su utilización en los laboratorios, el sistema permite agregar máquinas. Como la plataforma que se usa para montarlas es Docker, estas se agregan ingresando su url de *Dockerhub*, un repositorio de imágenes de Docker de acceso público. Además, desde acá se pueden configurar sus puertos, para que, al momento de montar un escenario, esto se haga sin problemas.

Configuración de laboratorios

Con la finalidad de montar un laboratorio, un usuario del sistema puede seleccionar máquinas que haya agregado anteriormente. Cada laboratorio tiene un identificador, que se utiliza para archivar la información de monitoreo tras terminarse el laboratorio. Posteriormente, el usuario puede definir qué máquinas están conectadas entre sí, en caso de que el laboratorio requiera que los estudiantes hagan un mapa de las redes del laboratorio.

Este módulo genera un archivo *Docker Compose*, que setea las variables necesarias para que las imágenes puedan ser montadas en otro ambiente. Esto tiene la finalidad de que un estudiante pueda experimentar con el laboratorio, aunque no pueda asistir y, por lo tanto, no tenga acceso a la red.

Tras generarse el archivo, el sistema le permite al usuario montarlos en la máquina en la que está montado el servidor.

Monitoreo de laboratorio

Después del paso anterior, si el laboratorio se monta en la máquina en la que corre el servidor del sistema, este le permite al usuario acceder a una interfaz de monitoreo.

Desde ésta, se puede ver la cantidad de accesos que se han hecho a las máquinas, y cuántos comandos se han ejecutado en ellas, además de revisar el detalle de peticiones y los comandos con sus argumentos.

Debido a que el monitoreo se hace desde el servidor en el que se monta el laboratorio, no es posible monitorear la actividad de un estudiante que replicó el escenario en su propio computador o servidor. Lo anterior es inviable, pues requeriría que su computador mande información al servidor de monitoreo. Por la naturaleza de los laboratorios, esto no es posible, pues lo recomendable es que el servidor de monitoreo solo reciba conexiones de un rango de IPs acotados, con la finalidad de prevenir ataques externos que podrían afectar el desarrollo de un laboratorio.

Resumen de los resultados

La plataforma descrita en el siguiente informe está validada mediante una sesión práctica en la que se utilizó. Participaron aproximadamente 45 estudiantes de Ingeniería Civil en Computación, quienes se inscribieron voluntariamente. La muestra, al ser voluntaria, no representa a cualquier estudiante de la carrera, pero los cursos relacionados que se dictan también son electivos, por lo que ocurre un sesgo similar. Tampoco hay manera asegurar un grupo representativo de estudiantes, porque no se puede obligar la participación, por lo que resultaría interesante validar los datos con otro grupo, que tengan otra motivación para participar, además de su motivación personal. Tras la sesión, se les hizo una encuesta de percepción subjetiva a los estudiantes que participaron, en la que se les preguntó por su experiencia durante la sesión. La mayoría estuvo de acuerdo, o muy de acuerdo, con que acompañar el conocimiento teórico con el práctico es algo que deja los contenidos más claros, disminuyendo las dudas a futuro, y el escenario montado parecía ser real, lo que es una de las características principales de la solución.n

Por otro lado, la plataforma funcionó sin problemas, y la información que se obtiene de la actividad de los participantes ayudó considerablemente a entregarles retroalimentación durante el taller. Debido a que la inscripción era voluntaria para cualquier estudiante del departamento, no se sabía con anticipación cuál era su nivel de dominio del tema. Por lo tanto, se utilizó la información obtenida del monitoreo para entregarles *pistas* sobre qué estaban haciendo bien, y qué estaban haciendo mal, con la finalidad de que todos lograran poner en práctica los contenidos vistos durante la parte teórica.

Capítulo 1

Marco teórico

A continuación se describen temas y conceptos que son de relevancia para la comprensión de la plataforma desarrollada.

1.1. Tecnologías

El desarrollo de la solución se separó en dos partes principales: el cliente y el servidor. A continuación se definen las tecnologías utilizadas por la solución en general, y luego las que se utilizaron en específico tanto en el cliente como en el servidor. El orden de las tecnologías descritas fue seleccionado para hacer la descripción autocontenida.

1.1.1. Tecnologías generales

HTTP

HTTP es una sigla en inglés que corresponde a Hypertext Transfer Protocol. Es definida por la W3C como un protocolo a nivel de aplicación para sistemas de información distribuidos, colaborativos y basados en hipertexto [9].

Una transferencia en HTTP tiene dos partes: el cliente, que empaqueta un documento y lo envía a un servidor. El servidor recibe la solicitud, procesa su contenido, y posteriormente le manda la respuesta al cliente.

Los documentos enviados por el protocolo tienen definiciones y estándares específicos para el encabezado del mensaje, que define algunos metadatos para interpretar el contenido del mensaje, pero no hay estándares para el contenido en sí.

HTTP es utilizado ampliamente, principalmente porque la mayoría de las acciones que realizan los navegadores Web utilizan este protocolo. En la aplicación desarrollada en este documento, el protocolo se usa para realizar la comunicación entre el cliente y el servidor, y se utiliza también para algunas consultas externas que realiza el servidor, en particular para obtener los datos de las máquinas agregadas.

REST

La W3C define REST, Representational State Transfer, o Transferencia de Estado Representacional como un tipo de arquitectura de software que define una serie de restricciones para crear servicios web [4].

Los clientes piden un recurso, en general ingresando a una URL, por ejemplo, `http://www.mascotas.cl/perros/13`, y el servidor les entrega una representación del recurso pedido, que en el ejemplo sería la **representación** del perro con la identificación *13*. Tras completarse esto, el cliente se encuentra en un **estado**. Si el cliente accede a un link desde la representación de la respuesta, accede a otro recurso, dejando al cliente en otro estado. El cambio de estado del cliente se denomina **transferencia**.

Según Richardson y Ruby [26], una aplicación REST tiene como características principales la direccionalidad, ser libres de estado, la conectividad y tener una interfaz uniforme.

La **direccionalidad** hace referencia a que la aplicación debe exponer los aspectos interesantes de sus datos como recursos. Los recursos son identificados a través de URIs, que son una secuencia de caracteres que identifican un recurso en la Web. Si un recurso no posee una URI, es inaccesible, por lo que no existe para el cliente.

Una aplicación REST es **libre de estado** porque no guarda estados del cliente en el servidor. Según la definición de Richardson y Ruby [26], esto quiere decir que cada petición HTTP a la aplicación sucede de forma aislada, por lo que el cliente debe enviar toda la información necesaria para que el servidor responda correctamente.

La conectividad hace referencia a que los servicios REST no entregan solamente datos, si no que tambien hipermedia y otros recursos, o enlaces a ellos.

La última característica importante de REST es que posee una interfaz uniforme de comunicación. Esto hace referencia a que hay unas pocas operaciones que se realizan sobre los recursos. La especificación HTTP define varios métodos básicos, siendo los más comunes en aplicaciones de este tipo **GET**, que se utiliza para obtener un recurso desde una URI; **POST**, que se usa para crear nuevos recursos; y **DELETE**, que es una petición al servidor para que elimine un recurso, o para que deje de ser accesible desde su URI.

La comunicación entre el cliente y el servidor de la aplicación desarrollada en este trabajo es de tipo REST: el cliente manda peticiones al servidor para obtener información sobre las máquinas que se han agregado anteriormente, para agregar nuevas máquinas, o para que se realicen operaciones más complejas.

JSON

JSON es una sigla que significa JavaScript Object Notation, y es definida por la W3C como un formato de intercambio de datos ligero, basado en texto e independiente del lenguaje. Define un conjunto pequeño de reglas de formateo para representar datos estructurados de forma portable [1].

El formato tiene un conjunto pequeño de tipos de datos: números, strings, booleanos,

arreglos, y objetos, que se definen como una colección no ordenada de pares nombre-valor, cuyos nombres son strings. También admite datos nulos.

El contenido de las peticiones que se hacen desde el cliente al servidor en la aplicación desarrollada, así como el contenido de las respuestas entregadas por el servidor, están en formato JSON.

1.1.2. Cliente

DOM

El DOM (Document Object Model) es una interfaz independiente del lenguaje que trata un documento HTML como una estructura de árbol, en que cada nodo es un objeto que representa una parte del documento. Hay métodos que utilizan el DOM para dar acceso programático al árbol, con la finalidad de cambiar su estructura, estilo o contenido. Además, los nodos pueden tener manejadores de eventos, que se llaman automáticamente bajo ciertas acciones en el elemento del DOM [5].

El DOM es utilizado por los navegadores web para renderizar documentos HTML, tras parsearlos y convertirlos a la estructura de árbol.

La aplicación desarrollada utiliza un framework llamado *Vue.js*, con la finalidad de manipular el DOM y reaccionar a acciones del usuario, sin necesidad de recargar la página.

Vue.js

Vue.js es un framework de JavaScript de tipo modelo-vista-modelo de vistas [14], que es un patrón de arquitectura de software que busca desacoplar lo máximo posible la interfaz de usuario con la lógica de la aplicación. Sus principales características son los componentes, sus *templates* y la reactividad.

Los **componentes** extienden elementos HTML y encapsulan código reutilizable, Son elementos personalizados a los que el compilador de Vue.js les agrega un comportamiento [34].

Este framework usa una sintaxis basada en HTML para sus ***templates***, los que le permiten renderizarlas al DOM utilizando los datos que maneja el framework. Sus templates se compilan en funciones de renderizado de DOM virtuales. Esto permite que se rendericen componentes en su memoria antes de que se actualicen en el navegador. De esta forma, se calcula el número mínimo de componentes que se deben re-renderizar al cambiar el estado [36].

Vue.js usa un sistema **reactivo** que usa objetos de JavaScript y optimizaciones para el re-renderizado. Cada componente mantiene sus dependencias reactivas, para que el sistema sepa cuándo debe re-renderizar, y qué componentes en específico lo necesitan cuando hay algún cambio en el estado de la aplicación [35].

La aplicación desarrollada utiliza Vue.js para reaccionar a las interacciones del usuario con la página sin tener que recargarla completamente. Además, durante el desarrollo se aprovechó el uso de componentes para reutilizar ciertos elementos visuales sin duplicar código.

1.1.3. Servidor

MongoDB

MongoDB es un programa de bases de datos orientadas a documentos. Utiliza documentos JSON-like con un schema que define los datos de los documentos y sus tipos.

Al igual que una base de datos normal, MongoDB permite hacer consultas por campo, por rango y también por expresiones regulares. Permite definir índices para optimizar queries, y está diseñado para soportar replicación de datos fácilmente, y balancear la carga entre las réplicas [16].

En el proyecto se utilizó MongoDB debido a su facilidad de integrarlo con una aplicación JavaScript. Esto se debe a que, como se indicó anteriormente, contiene documentos JSON-like, los que se pueden utilizar en JavaScript directamente tras utilizar el paquete que incluye MongoDB para su integración con el lenguaje. El proyecto no utiliza otras características como las réplicas ni el balance de carga, pues la aplicación no está pensada para tener muchos usuarios a la vez.

Node.js

Node.js es un ambiente de ejecución de JavaScript de código abierto, y permite la ejecución de código JavaScript fuera de un navegador. Permite que los desarrolladores usen el lenguaje JavaScript para escribir herramientas de línea de comandos y para correr scripts desde un servidor, para generar contenido dinámico. Permite que el desarrollo de una aplicación web use el mismo lenguaje tanto en el servidor como en el cliente [11].

El ambiente Node.js también tiene una arquitectura basada en eventos, capaz de recibir y enviar datos de manera asíncrona, por lo que se utiliza a menudo para aplicaciones web que requieren interacción en tiempo real.

En la aplicación desarrollada se utiliza Node.js en la parte del servidor, para utilizar una tecnología parecida a la del cliente. Node.js se usa en este caso para servirle al cliente los datos que se encuentran en MongoDB, y para correr comandos en la máquina que actúa como *host*.

Express.js

Express.js es una framework para aplicaciones web para Node.js. Está diseñada para crear aplicaciones web y APIs [10].

En el proyecto que describe este informe, Express.js se usa para recibir las peticiones enviadas al servidor, encontrar la información solicitada y devolverla al cliente.

Docker

El sistema descrito en el informe utiliza Docker y Docker Compose. Docker utliza virtualización a nivel de sistema operativo para entregar software, lo que significa que no tienen el costo de una máquina virtual convencional de correr el sistema operativo y el sistema de

archivos de la máquina virtualizada. El software que corre Docker viene en paquetes llamados *containers*, que están aislados entre sí, y vienen con su software, librerías y archivos de configuración [7].

Docker Compose, por otro lado, es una herramienta para definir aplicaciones que utlizan varios de los containers descritos anteriormente. Los servicios se configuran mediante archivos YAML, que son leídos por el software, para crear y iniciar los containers de manera automática, utilizando un solo comando. Además, esta herramienta permite definir conexions de red entre los containers configurados, entre otras cosas[15].

Estas herramientas fueron elegidas para el problema descrito pues es más fácil configurar laboratorios nuevos, pues cada máquina configurada puede reutilizarse en laboratorios posteriores. Además, dado que los contenedores son livianos al compararlos con una máquina virtual convencional, es más fácil que un estudiante monte el laboratorio en su propio computador, en caso de que quiera rehacerlo, o si no pudo asistir a la instancia original.

WebSockets

Los WebSockets son un protodoclo de comunicación en ambos sentidos, utilizando una única conexión TCP. Está diseñado para trabajar sobre los puertos 80 y 433, que se usan en HTTP y HTTPS, por lo que es compatible con el protocolo HTTP.

El protocolo WebSocket permite la interacción entre un navegador web (u otra aplicación como cliente) y un servidor web con un costo menor comparado a tener al cliente mandando peticiones constantemente (aunque no haya información nuevo). Esto porque define un estándar para que el servidor le envíe información al cliente sin que este le haga una petición, siempre que se mantenga la conexión abierta [21].

El protocolo es usado en este caso para entregar información al cliente en tiempo real sobre eventos que ocurren en el servidor. Se utiliza para entregar en tiempo real la salida de la consola cuando se montan las máquinas en el servidor, así como para los eventos nuevos tras iniciarse el monitoreo.

1.2. Metodologías de desarrollo

A continuación se describen las metodologías de desarrollo que se utilizaron en pos de la calidad de la solución, del código y del resultado final.

1.2.1. Desarrollo incremental

Según Roger Pressman [25], el desarrollo incremental es un método de desarrollo de software en el que el producto se diseña, implementa y testea de manera incremental, es decir, se va implementando la solución por partes, agregando algo nuevo cada vez, hasta que el producto se termina (cuando se cumplen todos los requisitos).

El producto se separa en componentes, cada uno de los cuales se diseña y se arma de forma separada. Cada componente se integra al producto cuando está terminado, lo que permite utilizar el producto antes de que esté completamente terminado.

Cada vez que se agrega algo nuevo al software se habla de "incrementos", y cada incremento agrega nuevas funcionalidades. El primer incremento entrega algo que ya es usable, y después de cada incremento se ven las prioridades para el siguiente, hasta que el producto se termina. Sus características principales son:

- El sistema se separa en proyectos de desarrollo más pequeños.
- Se arman sistemas parciales para producir el sistema final.
- Se priorizan los requerimientos de mayor prioridad.
- Los requisitos de una parte del sistema se congelan cuando se termina el desarrollo de la misma

Pressman [25] además indica que este modelo de desarrollo también se usa en procesos ágiles, ya que permite realizar entregas de manera más rápida, obteniendo así una retroalimentación más rápidamente desde los usuarios, y usando esta retroalimentación para mejorar la calidad del desarrollo.

La principal ventaja de este modelo de desarrollo es que, al estar diseñado para realizar entregas más seguidamente, permite ir probando el sistema a medida que se desarrolla, asegurando que cada una de las partes incrementales funciona bien, y que si hay regresiones, estas se deben a los cambios introducidos en el nuevo incremento.

Durante el desarrollo este proyecto se utilizó un desarrollo incremental porque, si bien las partes del sistema implementado tenían el mismo objetivo final, tienen baja interdependencia. Esto se aprovechó para ir probando el sistema por partes, e ir asegurando que cada una de ellas funcionaba de la forma que se esperaba antes de continuar con la siguiente. La forma en que se separó el sistema, y cómo se desarrolló cada una de las partes se describe mejor en los capítulos siguientes.

1.2.2. Sistema de control de versiones

Según lo descrito por Scott Chacon y Ben Straub [2], un sistema de control de versiones es un sistema que maneja y guarda los cambios en un archivo o conjunto de archivos a través del tiempo, permitiendo devolver a versiones específicas más tarde.

Típicamente la gente realiza un tipo de control de versiones, copiando sus archivos en otro directorio para respaldarlos. Para evitar los errores que suelen ocurrir con este sistema (por ejemplo, se pueden copiar archivos equivocados, o sobreescribir versiones sin querer), se desarrollaron inicialmente sistemas de control de versiones con una base de datos que iba archivando los cambios.

Chacon y Straub comentan que este sistema empieza a fallar cuando se empieza a colaborar con otras personas que también contribuyen sus cambios. Debido a esto, se desarrollaron los primeros sistemas de control de versiones centralizados (como CVS o Subversion) que tienen un único servidor que mantiene todos los archivos versionados, y clientes que pueden obtener estos archivos. La ventaja de estos sistemas es que cualquiera sabe hasta cierto punto qué está haciendo el resto de los contribuidores al proyecto, y es más fácil gestionar los permisos sobre los archivos.

La desventaja que describen para estos sistemas es que tienen un único punto de falla. El servidor se puede caer, y mientras permanezca así, nadie puede guardar versiones de los archivos en los que están trabajando. Si falla el disco del archivo, y no hay respaldos, se puede perder la historia completa del proyecto.

Por estas razones, según argumentan Chacon y Straub, se desarrollaron sistemas de control de versiones distribuidos. En estos sistemas los clientes no obtienen solo la última versión de los archivos, si no que obtienen todo el repositorio, con su historial completo. Así, si el servidor deja de funcionar, cualquiera de los clientes puede subir su repositorio a otro servidor, ya que posee un respaldo completo de los datos.

Git es un sistema de control de versiones que guarda versiones de los archivos, en lugar de diferencias entre las versiones [2]. Esto hace que los cambios entre versiones sean mucho más rápidos, y permite que la creación de ramas de desarrollo sea más eficiente, pues no se necesita recorrer una estructura de árbol para aplicar los cambios de otra rama. Además, la mayoría de las operaciones en Git necesitan solo recursos locales para funcionar, por lo que no hay latencia de red como en otros sistemas.

Este sistema de control de versiones asegura la integridad de los archivos que se usan, y cada cambio se guarda internamente agregando datos, por lo que cualquier operación se puede revertir.

El proyecto utiliza Git como su sistema de control de versiones. Este se utilizó en conjunto al modelo de desarrollo incremental para revertir fácilmente cualquier cambio si este introducía problemas en otro módulo del sistema. El sistema de control de versiones utilizado permite detectar fácilmente qué cambios se hizo en cada una de las versiones, y los comentarios que se usan ayudan a entender mejor las razones de cada cambio introducido.

1.3. Arquitecturas estándar

1.3.1. Cliente-Servidor

El modelo cliente-servidor es una arquitectura de aplicación distribuida que separa las tareas o trabajos entre quienes proveen un servicio, los que se llaman servidores, y quienes solicitan dichos recursos, llamados clientes [22].

Un *host* de servidor corre uno o más programas, los que comparten sus recursos con los clientes. Un cliente no comparte sus recursos, pero se encargan de iniciar la comunicación con el servidor, que está esperando solicitudes.

La aplicación desarrollada utiliza este modelo, pues se separa la responsabilidad en dos partes principales: el cliente, que es una aplicación web que muestra una interfaz a través de la cual un usuario interactúa con la aplicación, y el servidor, que recibe solicitudes del cliente y las procesa, y que es el que se encarga de guardar los datos y gestionar los laboratorios.

1.4. Soluciones existentes

Ya se han utilizado *honeypots* antes para enseñar ciberseguridad. Neil Eliot, David Kendall, y Michael Broc describen la arquitectura de un laboratorio que hicieron con componentes físicas para la enseñanza de ciberseguridad [8]. En su paper muestran cómo montaron una honeynet de tipo HILV de bajo costo, utilizando Raspberry Pis para poder cambiar fácilmente los escenarios de prueba para sus estudiantes. De sus resultados destaca la facilidad de reconfigurar sus escenarios para distintas sesiones, destacando la importancia de hacer laboratorios que sean altamente configurables y portables. En su caso, notaron también un aumento en el número de estudiantes que tomaban cursos de ciberseguridad, y su modelo les permitió integrar los conocimientos más fácilmente en cursos de redes. En su estudio, 89.66 % de los estudiantes que tomaron el curso encontraron el uso de honeypots como muy bueno o excelente, el 98 % encontró que eran fáciles de configurar, y el 97 % indicó que lo usarían dentro de sus métodos de estudio a futuro.

Aaron Lanoy y Gordon W. Romney [18] hicieron una comparación entre la enseñanza de ciberseguridad utilizando honeynets virtuales y honeynets físicas. La comparación se basó en cuatro puntos: instalación inicial, recolección de datos, análisis de datos y mantenimiento.

En cuando a la recolección y análisis de datos, no notaron grandes diferencias al comparar los dos tipos de honeynets. En las otras categorías, en cambio, sí hubo diferencias. En cuanto a la instalación inicial, habían previsto que sería más fácil para los estudiantes el caso de las honeynets virtuales. Sin embargo, muchos tuvieron problemas creando redes virtuales, debido a la dificultad que había para configurar correctamente las redes. La mantención, en cambio, era más fácil de hacer en las honeynets virtuales.

Finalmente, concluyen que las honeynets reales son un recurso valioso en la enseñanza, principalmente si parte de lo que se quiere enseñar es la experiencia de armar una honeynet. Por otro lado, concluyeron que si se quería hacer un laboratorio por más tiempo, era más fácil hacerlo con una honeynet virtual, porque la dificultad de la instalación inicial se puede aliviar entregándole a los estudiantes asistencia en el laboratorio.

Existen ya experiencias anteriores del uso de honeynets virtuales para enseñar ciberseguridad. En 2013 Iván Marsa-Maestre, Enrique de la Hoz, José Manuel Giménez-Guzmán y Miguel A. López-Carmona publicaron una descripción de la plataforma *NEMESIS*, que permite montar escenarios preconfigurados fácilmente, utilizando archivos de configuración XML [20]. Su experimento se realizó entre 2011 y 2012, en un curso de seguridad de internet que dictan en la Universidad de Alcalá. Para evaluar los resultados, evaluaron a sus estudiantes de la misma forma en que lo hacen tradicionalmente, siendo la única diferencia el uso de la plataforma NEMESIS. Según su estudio, los resultados de los estudiantes que usaron NEMESIS eran significativamente mejores que los que habían tomado el curso anteriormente, principalmente para la materia de seguridad de sistemas. Desde el punto de vista de quienes tomaron el curso, la percepción también fue muy positiva.

Para sus escenarios utilizan una distribución de GNU/Linux llamada *Damn Vulnerable Linux*, que se encuentra descontinuada [6], por lo que no puede ser utilizada para enseñar escenarios realistas o actuales.

Lamentablemente, a pesar de que en la publicación descrita anteriormente hacen una descripción detallada de lo que hace el programa y parte de su arquitectura, la implementación no está disponible en ningún lugar, y no ha sido publicada. Existen también videos que muestran cómo configuraban los escenarios, pero el software en sí nunca fue publicado, por razones que se desconocen.

Existen también herramientas que permiten montar honeynets fácilmente, siendo la más nueva y mejor mantenida la conocida como MHN (Modern Honey Network) [33]. Modern Honey Network permite montar honeynets pre-hechas y monitorear de manera general cómo las atacan. No obstante, las herramientas de monitoreo que vienen incluidas están pensadas para ataques a escala más grande, es decir, se pueden observar las IPs que están atacando la red, y qué puertos están intentando acceder, pero no a través de qué protocolos se intentan comunicar, o qué tipo de información están enviando.

Para poder estudiar y guiar a los estudiantes a los cuales se les está enseñando ciberseguridad, es necesario observar no sólo a través de qué protocolos se están conectando a las redes, si no que la información que están enviando a través de ellas, para lo que se pueden utilizar herramientas como Wireshark [12], que permiten observar de manera detallada el tráfico de una red de manera pasiva, sin afectar el tránsito en ella.

Buscando en la red se pueden encontrar varias soluciones que buscan enseñar a la gente habilidades de ciberseguridad de forma práctica. El nombre que reciben estas soluciones es "virtual penetration testing laboratories" y, lamentablemente, la mayoría son soluciones de pago y que corren en la nube [17, 24, 29], son soluciones de *una* máquina (sin redes) [27], o no están actualizadas [28].

Capítulo 2

Problema

2.1. Planteamiento y relevancia

Como se describió brevemente en la introducción, el problema que se busca abordar es la necesidad de contar con una forma rápida y dinámica para montar laboratorios de ciberseguridad, que permiten complementar la enseñanza de este tema a estudiantes universitarios de carreras relacionadas a la computación o informática.

El problema se puede dividir en dos dimensiones: la primera está relacionada con el montaje automático de honeynets de manera dinámica, y la segunda corresponde al uso de honeynets como herramientas de apoyo al aprendizaje de ciberseguridad. Ambas partes del problema se han abordado en trabajos previos, pero como se describió anteriormente, las soluciones existentes no son públicas, o se limitan a máquinas preconfiguradas y, por lo tanto, definidas anteriormente. Para asegurar que los escenarios montados puedan cambiarse en el tiempo, y con ello asegurar que se mantengan vigentes, es importante que las máquinas de los escenarios se puedan elegir dinámicamente.

Con la segunda dimensión existen problemas parecidos. Hay cursos que utilizan honeynets para enseñar sobre ciberseguridad, pero se dan en privado y son inaccesibles sin pagar.

En cuanto a la relevancia del problema, también se puede dividir en dos partes, dependiendo de quiénes se ven beneficiados por una solución: profesores o estudiantes.

Para un profesor de ciberseguridad, tener la habilidad de montar laboratorios prácticos para sus estudiantes puede ayudarlo a entregar de mejor manera conceptos, especialmente los que tienen aplicación directa.

En general montar laboratorios es una tarea que requiere bastante tiempo, sean los laboratorios hechos con máquinas virtuales o reales; hay que diseñar el laboratorio, configurar las máquinas, configurar las redes, y probar que todo esté funcionando correctamente. Tanto el diseño como el testing son tareas que se deben realizar de todas formas, pero sí hay una oportunidad de ahorrar tiempo tanto en la configuración de las máquinas como de las redes.

Además, se puede agregar valor proveyendo de feedback en tiempo real de lo que está pasando en el laboratorio, con la finalidad de entregarle mejores herramientas a los estudiantes, o identificar algún concepto que no haya quedado claro, sin tener que esperar a una instancia evaluativa para que esto sea evidente.

Para un estudiante el valor que se agrega tiene más relación con el último punto: obtener retroalimentación más rápido le permite identificar más fácilmente los conceptos o temas que tiene que reforzar. Junto a esto, la retroalimentación temprana puede ser un factor que motive su aprendizaje, ya que puede ir viendo sus avances más seguidamente.

Tener un apoyo práctico en la docencia es importante también para reforzar lo aprendido e internar los conceptos que son vistos en cátedra, y es una práctica que se utiliza en todos los niveles del aprendizaje.

2.2. Requisitos

Durante el desarrollo de la solución no se utilizó un proceso de desarrollo tipo cascada. Sin embargo, para entender mejor cuáles eran los objetivos específicos de cada módulo del sistema, a continuación se presentan los principales requisitos del sistema desarrollado. En ella se incluye su identificador, nombre, descripción y prioridad. Éstos están separados por cada uno de los módulos de la solución, para dar a entender mejor el funcionamiento de cada uno de ellos.

2.2.1. Gestión de máquinas

Identificador	RU001
Nombre	Agregar máquinas
Descripción	El sistema permite agregar máquinas nuevas con un nombre descriptivo, el nombre de la imagen, un tag, una descripción, y los puertos que utiliza, y si estos deben ser expuestos a una red pública o no.
Prioridad	Crítica

Identificador	RU002
Nombre	Información de máquinas
Descripción	El sistema permite revisar qué máquinas han sido agregadas anteriormente, con su nombre descriptivo, el nombre de la imagen, tag y descripción.
Prioridad	Crítica

Identificador	RU003
Nombre	Búsqueda de imágenes en Dockerhub
Descripción	Al momento de agregar una máquina, el sistema permite buscar una imagen preexistente en Dockerhub con su URL, para asegurar que ésta es válida, y se va a poder instalar en la máquina del servidor.
Prioridad	Crítica

Identificador	RU004
Nombre	Filtrado de máquinas por nombre
Descripción	El sistema permite filtrar máquinas por su nombre descriptivo o por el nombre de la imagen, para encontrarlas más fácilmente.
Prioridad	Deseable

Identificador	RU005
Nombre	Estadísticas de máquinas
Descripción	El sistema muestra un resumen del uso de cada máquina en escenarios anteriores.
Prioridad	Deseable

2.2.2. Configuración de escenarios

Identificador	RU006
Nombre	Selección de máquinas
Descripción	El sistema permite elegir qué imágenes se utilizarán en cada escenario.
Prioridad	Crítica

Identificador	RU007
Nombre	Configuración de conexiones
Descripción	Una vez seleccionadas las máquinas para un escenario, el usuario puede elegir las que estarán conectadas entre sí y que, por lo tanto, son visibles las unas desde las otras.
Prioridad	Crítica

Identificador	RU008
Nombre	Nombre de laboratorio
Descripción	El sistema debe permitir al usuario seleccionar un nombre para el laboratorio, con la finalidad de identificarlo y revisar sus estadísticas aún después de la sesión.
Prioridad	Deseable

Identificador	RU009
Nombre	Configuración de puertos reservados
Descripción	El sistema debe permitir al usuario indicar cuáles puertos están usados en la máquina que se usa como servidor, para que no haya conflictos con la asignación automática.
Prioridad	Deseable

Identificador	RU010
Nombre	Confirmación
Descripción	Antes de comenzar el laboratorio, el usuario puede ver un resumen de las máquinas configuradas y sus conexiones.
Prioridad	Deseable

Identificador	RU011
Nombre	Archivo de configuración
Descripción	Cuando se ha terminado de configurar el laboratorio, el sistema genera un archivo *docker-compose* con la configuración indicada.
Prioridad	Crítica

Identificador	RU012
Nombre	Descarga del archivo de configuración
Descripción	Tras generar el archivo de configuración, el sistema le permite al usuario descargarlo, por si lo quiere utilizar en otra máquina
Prioridad	Deseable

2.2.3. Montaje

Identificador	RU013
Nombre	Montaje automático
Descripción	Una vez generado el archivo de configuración, el sistema permite montar el laboratorio en la máquina que corre su servidor.
Prioridad	Crítica

Identificador	RU014
Nombre	Retroalimentación del montaje
Descripción	Si se monta el laboratorio en la máquina que corre el servidor, el usuario debe poder ver el progreso mientras se montan las máquinas.
Prioridad	Deseable

Identificador	RU015
Nombre	Retroalimentación del montaje
Descripción	Si se monta el laboratorio en la máquina que corre el servidor, el usuario debe poder ver el progreso mientras se montan las máquinas.
Prioridad	Deseable

2.2.4. Monitoreo

Identificador	RU015
Nombre	Vista general de monitoreo
Descripción	El sistema tiene una vista general para monitorear un laboratorio. En ella, se ve un resumen de las estadísticas de cada máquina, incluyendo las peticiones que han recibido, y los comandos que se han ejecutado en ellas.
Prioridad	Crítica

Identificador	RU016
Nombre	Actualizaciones en tiempo real
Descripción	Si se está viendo un laboratorio activo, se le indica al usuario cuando han recibido peticiones nuevas.
Prioridad	Deseable

Identificador	RU017
Nombre	Detalle por máquina
Descripción	Desde la vista general el usuario puede acceder a una vista más en detalle de la actividad de cada máquina, para revisar sus datos en tiempo real.
Prioridad	Crítica

Identificador	RU018
Nombre	Actualización de datos por máquina
Descripción	Desde la vista de cada máquina, el usuario puede apretar un botón para actualizar la información del monitoreo.
Prioridad	Crítica

Identificador	RU019
Nombre	Filtro de datos por máquina
Descripción	El sistema tiene filtros que permiten ver las peticiones más comunes, así como los comandos que se han corrido más veces en cada máquina.
Prioridad	Deseable

Identificador	RU020
Nombre	Archivado de monitoreo
Descripción	Cuando un laboratorio haya terminado, un usuario puede indicarlo. Cuando realiza esta acción, la información de monitoreo se guarda, por si quiere revisarla a futuro.
Prioridad	Deseable

Identificador	RU021
Nombre	Revisión de laboratorios anteriores
Descripción	Tras archivar un laboratorio, un usuario puede buscarlo por su ID y volver a revisar sus datos
Prioridad	Deseable

2.3. Calidad y criterios de aceptación

A continuación se describen las características de calidad deseadas para una solución al problema planteado, junto a los criterios considerados para que una solución sea válida:

2.3.1. Calidad

Debido a que el sistema no está pensado para recibir una gran cantidad de usuarios al mismo tiempo (solo uno se encarga de diseñar un laboratorio), hay criterios típicos de calidad que no se evalúan para una solución del problema, como lo son la eficiencia y escalabilidad. Esto no significa que la solución descrita en el capítulo siguiente sea ineficiente, o no pueda escalar, si no que simplemente que no es un criterio a tomar en cuenta para evaluar la solución como válida.

1. El sistema debe ser estable. En caso de que ocurra un error, debe poder recuperarse y mostrarle un mensaje al usuario.
2. El código debe ser extensible. Para ello, es necesario que esté modularizado, y que sus componentes tengan bajo acoplamiento.
3. El sistema debe tener buenos tiempos de respuesta mientras se está armando un laboratorio.

2.3.2. Criterios de aceptación

1. El sistema debe ser fácil de usar. Para ello, es necesario realizar estudios de usabilidad sobre una solución. De ser necesario, tras estos de estudios de usabilidad, la solución puede ser refinada.
2. El sistema debe servir como herramienta de apoyo a la docencia. Para ello, se realizará un estudio sobre percepción (subjetivo) con un grupo de usuarios estudiantes de prueba. No es posible hacer un estudio objetivo, pues en el contexto local no se han dictado cursos relacionados a ciberseguridad en un largo tiempo, por lo que no existen datos suficientes para hacer una comparación con un *grupo de control*.
3. El sistema debe ser probado en un escenario real, armando un laboratorio utilizando la herramienta, y utilizándolo con un grupo de estudiantes como acompañamiento a una clase o taller, en forma exitosa. Es decir, tanto el diseño del laboratorio como su aplicación deben poder realizarse satisfactoriamente.

Capítulo 3

Solución

3.1. Arquitectura

3.1.1. Arquitectura del software

Para la solución implementada se utiliza el *stack* de tecnologías conocido como MEVN. A continuación se muestra una imagen con la arquitectura básica del sistema, y luego se explica cómo se conectan estas componentes.

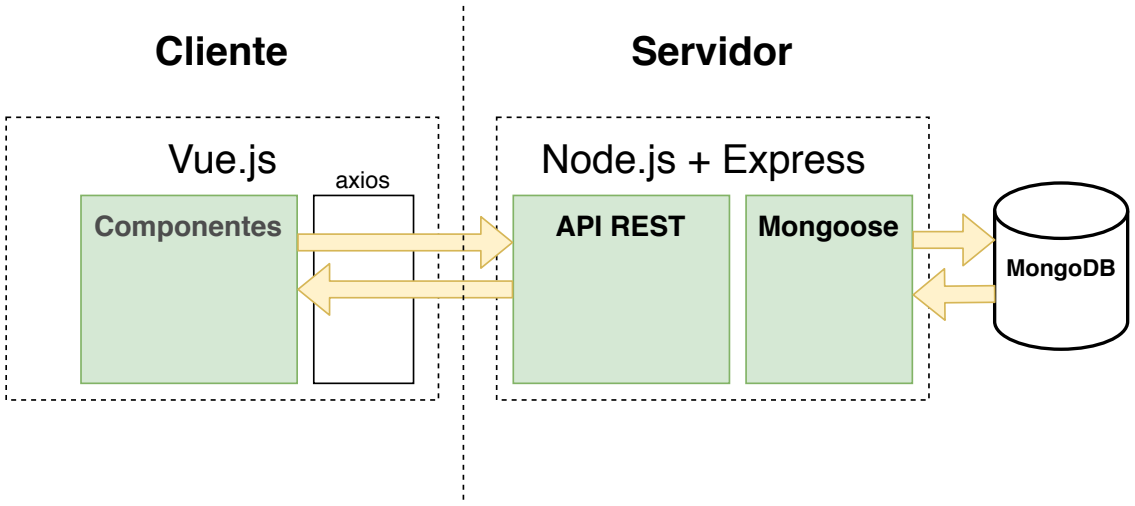

Figura 3.1: Arquitectura de la solución.

El stack MEVN obtiene su nombre de las cuatro tecnologías principales que utiliza: MongoDB, Express.js, Vue.js y Node.js. El orden en que aparecen las tecnologías no tiene sentido técnico, si no que proviene de un juego de palabras del nombre de otro stack: MEAN (que en vez de Vue.js, usa Angular).

En el caso de nuestra solución, tenemos una arquitectura típica cliente-servidor. En el cliente tenemos una aplicación que corre utilizando Vue.js. Las componentes corresponden a

módulos que representan elementos visibles de la interfaz de usuario (widgets). Cada uno de ellos maneja su estado, y tiene sus propios métodos.

La mayoría de los componentes de la aplicación obtiene los datos que muestra de una base de datos, para ello, se utiliza *axios*, que es un paquete de JavaScript que le permite a una aplicación realizar peticiones HTTP o HTTPS.

Las peticiones se envían a través de axios a un servidor que está corriendo Express, un paquete que corre sobre Node.js, y que actúa como servidor, es decir, recibe peticiones HTTP y las responde. Para las peticiones que requieren obtener información de la Base de Datos, se utiliza Mongoose, un middleware que hace las consultas a la base de datos, y las convierte a un objeto JavaScript, con el formato indicado en el esquema correspondiente. Luego, estos objetos son entregados al servidor Express para que se devuelvan por la API.

3.1.2. Arquitectura del Hardware

Para las pruebas descritas en este informe, no hay mucho que sea necesario destacar en cuanto al hardware utilizado. Por la forma en que está desarrollada la solución, debería correr en cualquier computador que tenga instaladas las dependencias de software.

No obstante, las pruebas en particular se realizaron corriendo las aplicaciones de cliente y servidor en servidores de Amazon, en instancias EC2. Esta decisión se tomó por tres razones principalmente:

1. **Facilidad de configuración:** montar un servidor es muy rápido. Como todo está en *la nube*, no es necesario preocuparse personalmente de la configuración del hardware. Además, Amazon permite limitar fácilmente desde qué IPs nuestro servidor recibe solicitudes. Esto se utiliza para aceptar solamente las IPs que corresponden al laboratorio que se está corriendo.
2. **Costos:** se aprovechó de usar créditos que Amazon regala a estudiantes mediante su programa *AWS Educate*. Esto significó que finalmente, las pruebas realizadas no tuvieran costos.
3. **Velocidad de conexión:** los servidores utilizados, si bien no tienen tantos recursos, sí tienen una conexión bastante rápida. Al momento de descargar las imágenes para montar los laboratorios, utilizar los servidores implica una descarga 10 veces más rápida.

Si bien se pierden algunas cosas al utilizar servidores externos, ya que se depende de otro ente, que no necesariamente es confiable, para este caso, y ya que es una aplicación que no está abierta al público, se considera que las tres ventajas descritas anteriormente superan a las posibles desventajas de utilizar este servicio.

3.2. Base de datos

Para el funcionamiento de la aplicación no es necesario guardar una gran cantidad de datos. Por eso, el modelo de la base de datos no es muy complejo. A continuación se muestra un diagrama de la base de datos:

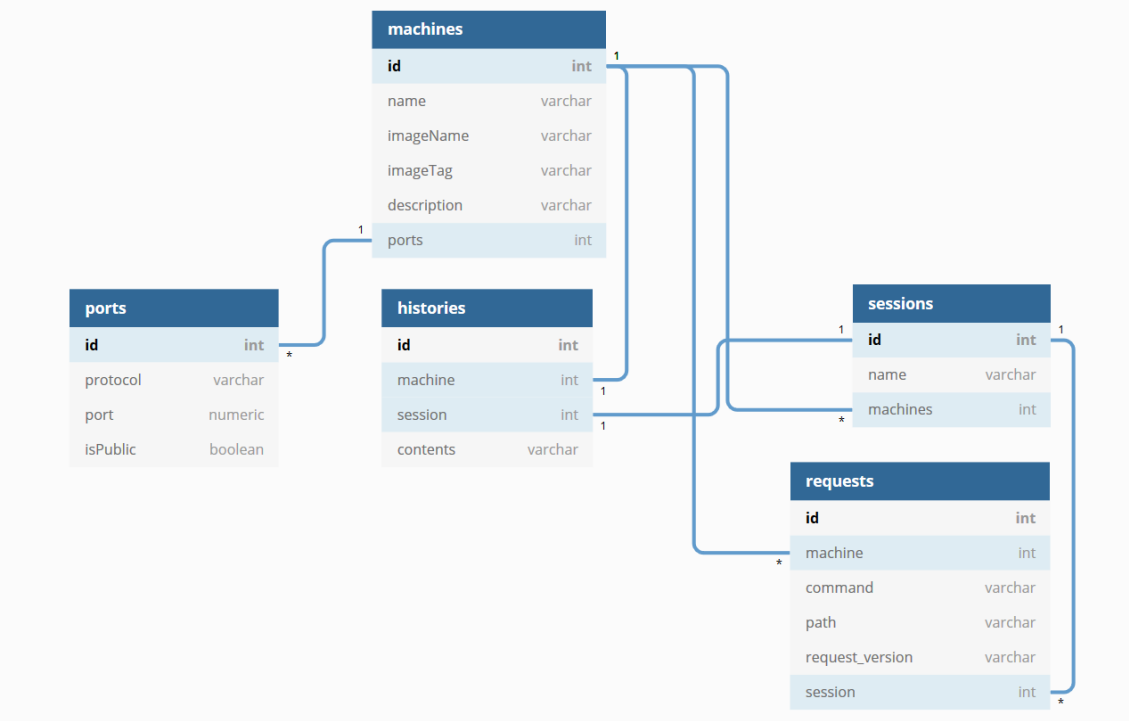

Figura 3.2: Diagrama de la Base de Datos.

3.2.1. Máquinas

Las **máquinas** son la unidad más fundamental para el uso de la aplicación. Además de su ID, tienen un nombre para que los usuarios de la aplicación puedan identificarlas. Junto a esta identificación de cara al usuario, se les permite también guardar una descripción, por si necesitan más texto para recordar qué corre cada máquina.

Cada máquina tiene además un nombre de imagen y un tag de imagen. El nombre de imagen corresponde al nombre que tiene cada máquina subida a Docker Hub, lo que es necesario para descargarlas posteriormente, e identificarlas en esa plataforma. El tag de la imagen sirve en caso de que hayan varias versiones de una máquina.

Finalmente, en una máquina se pueden configurar sus **puertos**. Los puertos son utilizados por la máquina de *host* para identificar a qué contenedor le pasará los datos. Luego, cada contenedor identifica a cuál de sus puertos internos corresponde el paquete, para que sea recibido por la aplicación que lo usa.

3.2.2. Puertos

A nivel de la base de datos, los puertos son los que cada contenedor asigna a sus aplicaciones internamente. Esta información es utilizada para que la máquina *host* pueda dirigir los paquetes entrantes correctamente, ya que la asignación interna no es necesariamente la misma. Se guarda también información extra, como el protocolo para el que utiliza cada contenedor el puerto en específico, para configurar el monitoreo de los protocolos particulares. Finalmente, cada puerto se puede dejar abierto para aceptar cualquier conexión, o solo desde los contenedores configurados al montar un laboratorio.

3.2.3. Historial de laboratorios

Con la finalidad de monitorear las sesiones llevadas a cabo con la herramienta, se guardan también las solicitudes a las máquinas, y el historial de sus *shells*. En ambas se guarda tanto la máquina como la sesión, para poder revisar la información de cada sesión posteriormente.

Las **historias** corresponden al log de shell de cada máquina, y sus contenidos se guardan directamente. Posteriormente, son procesados para que el usuario que esté monitoreando un laboratorio pueda ver sus contenidos más fácilmente.

Para las **solicitudes**, además, se guardan el comando (como GET o POST), la ruta a la que se hizo la solicitud, y la versión de HTTP que se utilizó.

Finalmente, para las sesiones no se guarda mucha información más (además de la información de las relaciones), ya que no es necesario para las interfaces de usuario. Se guardan las máquinas utilizadas para acceder a la información más fácilmente, y un nombre, para que los usuarios de la plataforma puedan identificar sus laboratorios, y revisarlos más tarde.

Cabe destacar que la aplicación no tiene usuarios u algún otro método de autenticación. Esto se debe a que la aplicación corre de manera local, por lo que se asume que no es necesario tener este tipo de control.

3.3. Interfaces de usuario y funcionamiento de la aplicación

A continuación se describe de forma general cómo funciona la aplicación, y el detalle de cada interfaz de usuario. Para cada una de ellas, se muestra y explica su funcionalidad.

3.3.1. Interfaz principal

En la figura se puede ver cómo la interfaz principal de la aplicación se divide en tres partes:

- A la izquierda hay un menú que le permite al usuario moverse por las tres vistas principales: la primera es la de *deployment*, en la que se puede configurar y montar un escenario para un laboratorio. Cuando se hace click en algún elemento del menú, se cambia la URL actual en la que está el usuario, y el enrutador de la aplicación les muestra la interfaz correcta.

- En la parte superior hay una barra con el título de la aplicación, y un botón que permite ocultar o mostrar el menú de la izquierda
- El resto de la interfaz, y lo que ocupa más espacio, es el contenido, que depende del ítem seleccionado en el menú de la izquierda, y en qué estado se encuentre. Cada uno de estos elementos se explicará en más detalle más adelante.

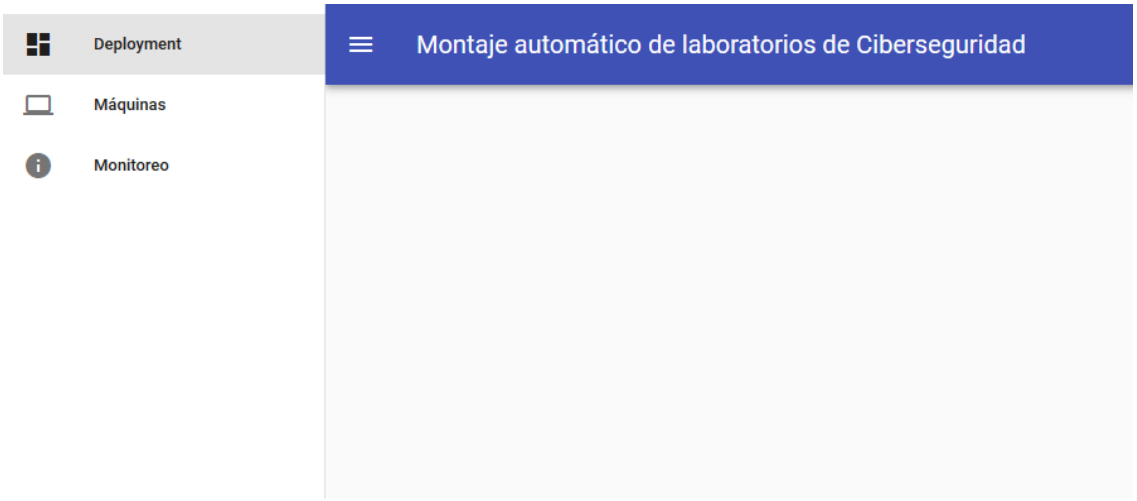

Figura 3.3: Interfaz principal de la aplicación.

Como también se puede ver en la figura, las interfaces de usuario de la aplicación están basadas en las directrices de diseño de las aplicaciones de Android, *Material Design*. Esto es porque se utilizó un paquete que funciona junto a Vue.js, llamado Vuetify. Este paquete se utilizó para armar las interfaces más fácilmente, con elementos prediseñados.

Estado y funcionamiento general de la aplicación

Sea cual sea la URL en la que un usuario está en la aplicación, se muestra la interfaz general mostrada más arriba. El contenido mostrado en el espacio sobrante depende de cuál elemento está seleccionado en la barra de la izquierda. Este comportamiento es controlado por el enrutador que viene incluído con la librería Vue.js.

Además, la aplicación guarda en su estado general ciertas variables, de forma de pasarlas más fácilmente a las interfaces que lo necesiten. Las variables que se comparten son las siguientes:

- Compose: el sistema guarda en su estado una copia del archivo *docker compose* generado para montar el laboratorio.
- Machines: una copia con el arreglo de las máquinas del laboratorio actual (si es que lo hay)
- SessionName: el nombre de la sesión del laboratorio en curso
- MonitoringSession: un booleano para que la aplicación recuerde si hay o no una sesión en curso.

Junto a facilitar compartir información entre las distintas componentes del sistema, estas variables se utilizan con un plugin de Vue.js llamado "vuex-persistedstate", que permite guardar el estado general de la aplicación en el LocalStorage del navegador del usuario. De esta forma, si un usuario refresca o cierra accidentalmente la aplicación mientras está configurando o monitoreando un laboratorio, el estado de este no se pierde.

Finalmente, a este nivel de la aplicación se configuran algunos *plug-ins* necesarios para su funcionamiento general. Los principales son *axios*, que permite hacer peticiones HTTP o HTTPs desde los componentes de la aplicación, y VueSocket, que se utiliza para obtener información en tiempo real de los procesos que corren en la máquina host.

3.3.2. Gestión de máquinas

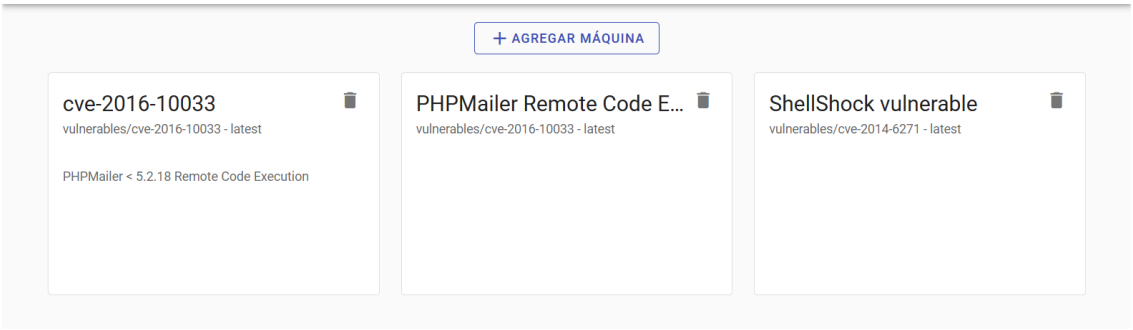

Figura 3.4: Interfaz de gestión de máquinas.

Desde la interfaz de gestión de máquinas, se puede ver un resumen de las que hay actualmente, junto a su información más relevante. En el título de cada elemento se ve el nombre que el usuario de la aplicación le puso a la máquina. Bajo este, se pueden ver el nombre interno de la imagen, el *tag* o versión de la misma y, más abajo, la descripción que el usuario ingresó para la máquina (si es que lo hizo).

Desde la misma interfaz se puede acceder a la interfaz para agregar máquinas, así como eliminar las máquinas si ya no se utilizarán más. En este último caso, se le pide al usuario que confirme su acción a través de un diálogo.

La componente que muestra la información de cada máquina tiene una propiedad definida a través de su constructor que define si se está en la interfaz de deployment o no. Esto sirve para mostrar u ocultar el botón para remover las máquinas, y además, para permitir que sean seleccionadas.

3.3.3. Agregar máquinas

Para agregar una máquina al sistema, primero es necesario que el usuario obtenga un link a la máquina que quiere agregar desde Docker Hub. En caso de que el usuario no sepa de qué manera hacerlo, el sistema incluye instrucciones paso a paso.

Cabe destacar que Docker Hub es sólo un repositorio de imágenes de Docker, pero la API es abierta. Es decir, cualquiera podría hacer su propio repositorio, y el sistema desarrollado

debería funcionar de igual manera. Sin embargo, por ahora el soporte de la aplicación sólo incluye ese repositorio, pues es el más utilizado, y con mayor cantidad de máquinas disponibles. Además, para usarlo no es necesario hacer configuraciones extras.

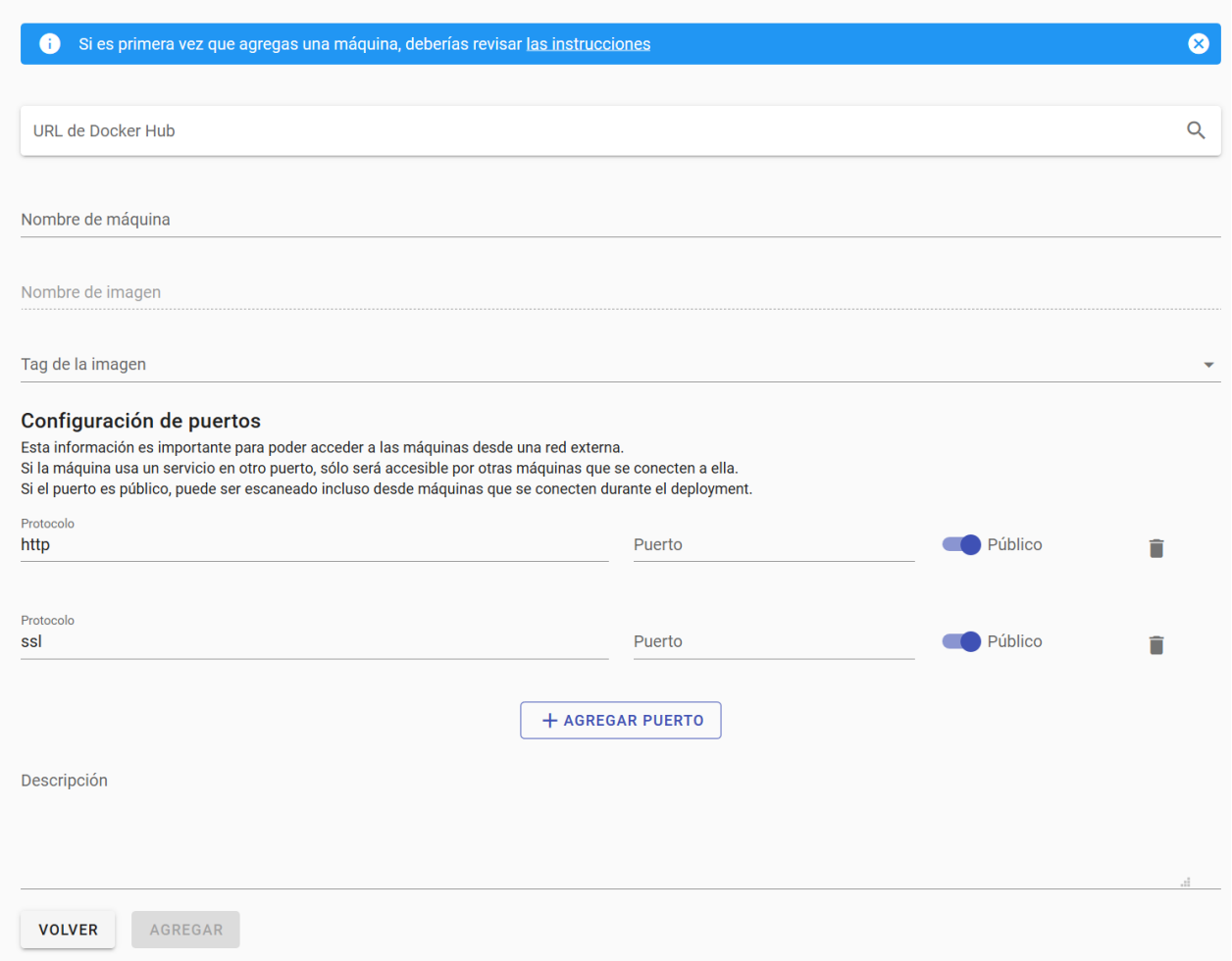

Figura 3.5: Interfaz para agregar máquinas.

Una vez ingresada la URL de la imagen que se quiere agregar al sistema, este realiza una consulta al backend para encontrar la información necesaria para agregarlo al sistema. Este funcionamiento se explica en más detalle en 3.4.3.

El paso anterior rellena automáticamente el campo *Nombre de la imagen*, y puebla el menú de selección para el tag,

Además, el usuario puede agregar qué puertos utilizan las aplicaciones que corren en la máquina. Esto es necesario para poder crear el archivo *Docker Compose* posteriormente, sin que haya conflictos si es que se utilizan muchas máquinas. La configuración de puertos también sirve para algunos protocolos especiales durante el monitoreo, como el puerto que se utiliza para las conexiones HTTP y SSL.

Finalmente, los usuarios pueden agregar una descripción para las máquinas, por si quieren tener más información al momento de seleccionarlas para laboratorios futuros.

Es importante destacar que los usuarios sólo pueden agregar o quitar máquinas. Esto es una decisión que se tomó de forma consciente, y se debe a que cambiar una máquina después de que esta se haya utilizado podría hacer que los historiales de monitoreos dejen de tener sentido. La eliminación de máquinas, en cambio, es algo solamente cosmético (más que eliminar las máquinas, éstas se ocultan), por lo que no afecta los historiales.

3.3.4. Selección de máquinas

Si bien todas las interfaces correspondientes al montaje de un laboratorio están relacionadas, cada una se describe en su propia subsección, con la finalidad de facilitar la lectura.

Todas las interfaces de configuración tienen en su parte superior un elemento que muestra en qué paso de la configuración se está. Cada una de las interfaces tiene su propio componente y, por lo tanto, su propio estado. No obstante, hay también un estado para la configuración en general, para ir acumulando los datos de cada paso para el momento de generar el archivo de configuración final. Para ello cada vez que hay un cambio en una de los componentes, estas se encargan de notificar esto a su componente superior mediante el método *emit* que es proporcionado por Vue.js.

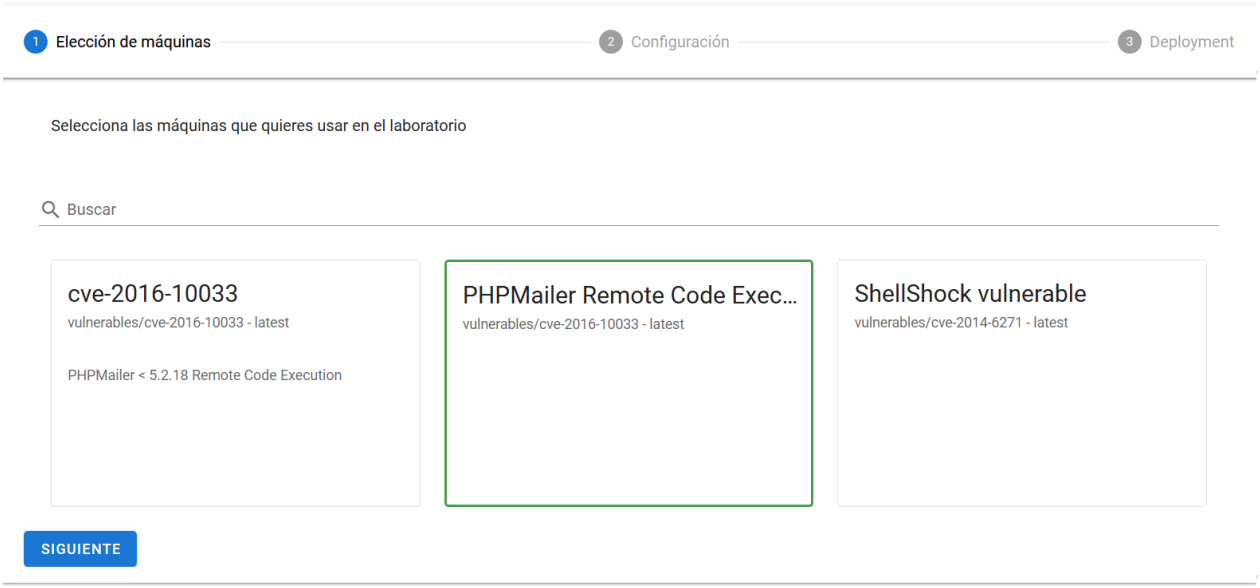

Figura 3.6: Interfaz de selección de máquinas.

La interfaz de selección de máquinas es bastante parecida a la interfaz principal de gestión de máquinas, pero las máquinas se pueden seleccionar para utilizarlas en el laboratorio. En la parte superior se ve en qué paso del montaje de laboratorio se encuentra el usuario. Más abajo está la barra de búsqueda, la que permite encontrar máquinas más fácilmente, en caso de que se hayan agregado muchas al sistema.

Cada máquina guarda internamente si está o no seleccionada, y este estado es también guardado por el componente que mantiene la configuración completa. Esto permite filtrar máquinas no cambie su estado de seleccionadas.

3.3.5. Configuración del laboratorio

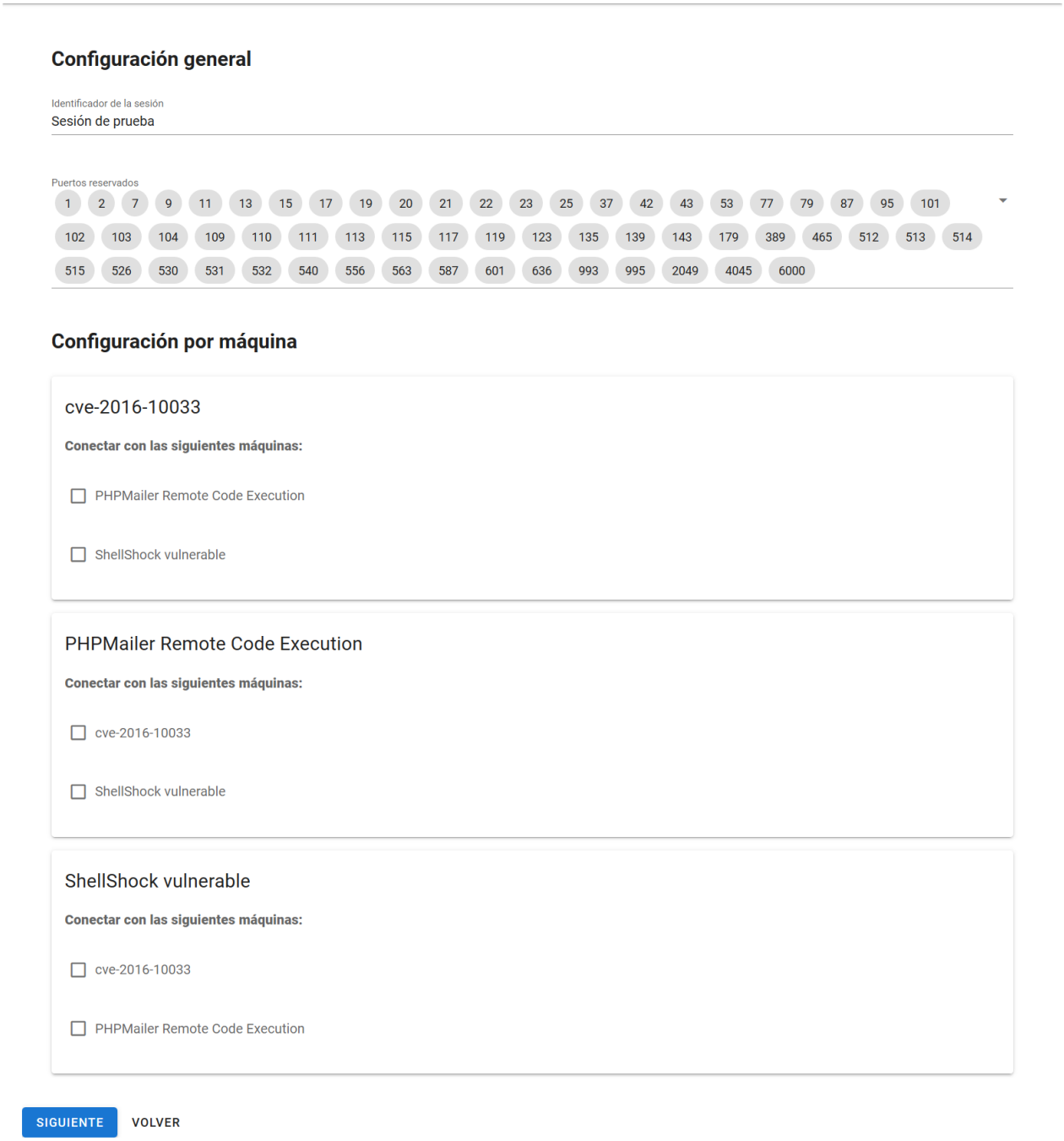

Figura 3.7: Interfaz de configuración de laboratorio.

Tras haber seleccionado las máquinas que quiere utilizar en el laboratorio, y cuando el usuario aprieta el botón "Siguiente", las máquinas seleccionadas se pasan al estado general, y son pasadas como propiedad para el componente que controla el segundo paso, de configuración del laboratorio.

La interfaz de configuración se divide en dos secciones. La superior corresponde a la configuración general del laboratorio. El usuario puede escribir el nombre para identificar la sesión, lo que le permite revisar la actividad aún después de que haya terminado.

También se pueden configurar los *puertos reservados*. Estos corresponden a puertos que, en general, son usados por ciertos protocolos comunes, y que no se pueden acceder con cierto software que no está diseñado para trabajar con esos protocolos. Por ejemplo, los navegadores no se conectan al puerto 22, porque generalmente se utiliza para el protocolo *SSH*. Esto no significa que una máquina que utiliza esos puertos no pueda funcionar, si no que estos se asignarán a otro puerto al momento de generar el archivo de configuración. Algunos puertos vienen pre-configurados, pero el usuario puede agregar más puertos, y también quitarlos, tomando en cuenta que puede dificultar su acceso.

Anteriormente se habían agregado opciones para activar o desactivar las opciones de monitoreo según el usuario. Sin embargo, esto se quitó, pues se determinó que la acción de estos switches no era intuitiva; los usuarios no veían directamente la consecuencia de esta acción y, por lo tanto, se podían quedar sin la habilidad monitorear algo por error.

La otra sección de configuración corresponde a la configuración por máquina. En ella, se pueden configurar las conexiones entre máquinas. De esta forma, las máquinas pueden comunicarse entre sí, aunque sus puertos no sean públicos.

Se consideró también agregar otras opciones de configuración. Por ejemplo, para permitirle al usuario agregar ciertos paquetes, o correr scripts automáticamente al montar las máquinas. Esta opción se descartó, porque complejizaba mucho el proceso de configuración, y la instalación de paquetes y su configuración depende de la distribución que corre la máquina, y el resto del software que utiliza. La distribución en general se puede detectar, pero hacer lo mismo con el resto del software que corre la máquina es algo mucho más complejo, por lo que esta opción se descartó.

3.3.6. Confirmación y montaje

Tras haber seleccionado y configurado las máquinas que se utilizarán en el laboratorio, el sistema muestra un resumen de la configuración del usuario, y genera el archivo Docker Compose correspondiente.

En el resumen de cada máquina, el usuario puede ver fácilmente su identificación, sus conexiones, y los puertos que utilizará en la máquina del servidor. Esto sirve para identificar fácilmente si algo se configuró mal, ya que desde este paso aún es fácil que un usuario modifique la configuración de ser necesario.

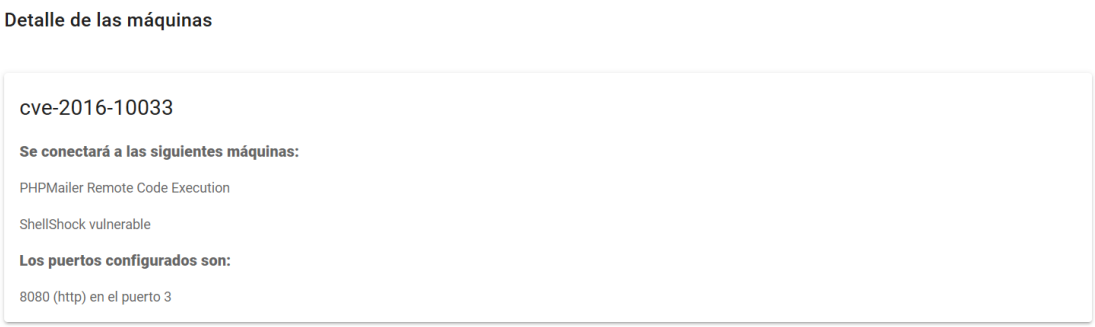

Figura 3.8: Resumen de configuración por máquina.

Bajo el resumen de las máquinas, el usuario puede ver y descargar el archivo *Docker Compose*. La forma en que se genera este archivo se puede ver en la subsección 3.4.3.

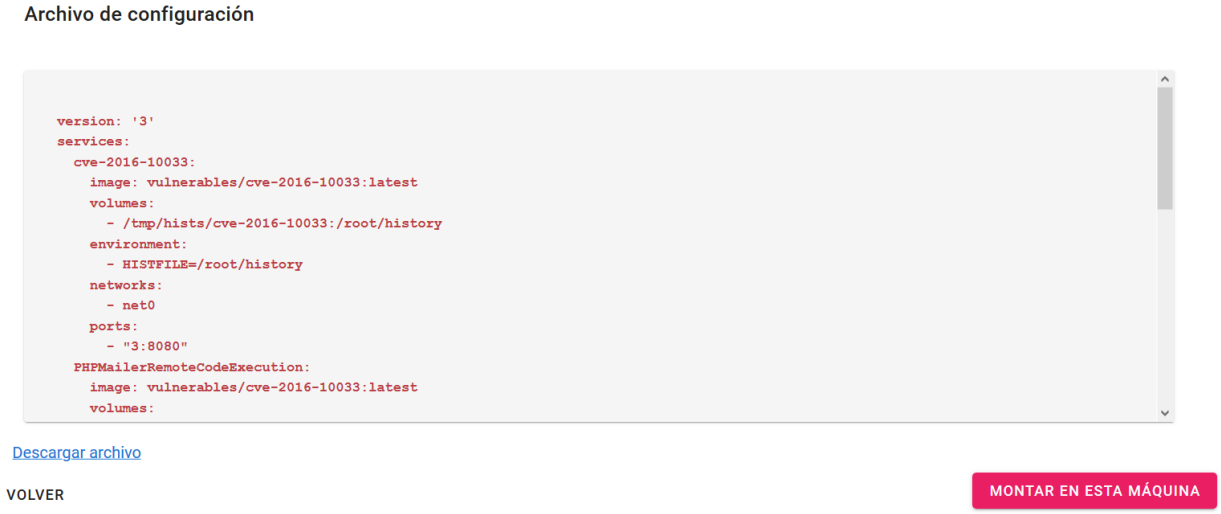

Figura 3.9: Archivo de configuración generado.

Junto al archivo de configuración, el usuario tiene un botón que le permite montar el laboratorio configurado en el servidor.

3.3.7. Estado de montaje

Si el usuario decide montar la aplicación en el servidor que se está usando, se le envía una petición al servidor, y el cliente y el servidor se conectan mediante un socket, el que se utiliza para que el servidor le envíe al cliente el estado actual del montaje en la máquina.

Para mostrarle la información al usuario se utiliza un emulador de consola en JavaScript, llamado *xterm*. Esto permite mostrar los mensajes fácilmente, y como si el usuario los estu-

viera corriendo en su máquina local, sin necesidad de escribir un parser (que fue una de las opciones que se había considerado inicialmente).

El montaje puede fallar por distintos motivos, siendo el más común que haya un puerto en uso en el servidor y que no se haya configurado como *puerto reservado* durante la configuración general del laboratorio.

Si el montaje no falla, en cambio, se le da la opción al usuario de pasar directamente al monitoreo. Para revisar que esté listo, simplemente se revisa la información que llega a través del socket. Cuando todas las máquinas se montan correctamente, significa que el proceso de montaje ha finalizado, y se pude monitorear el estado de las máquinas.

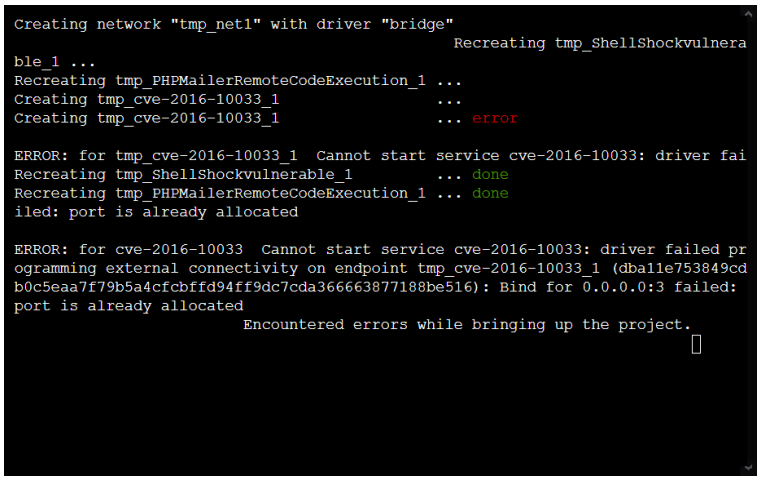

Figura 3.10: Terminal cuando el montaje falla.

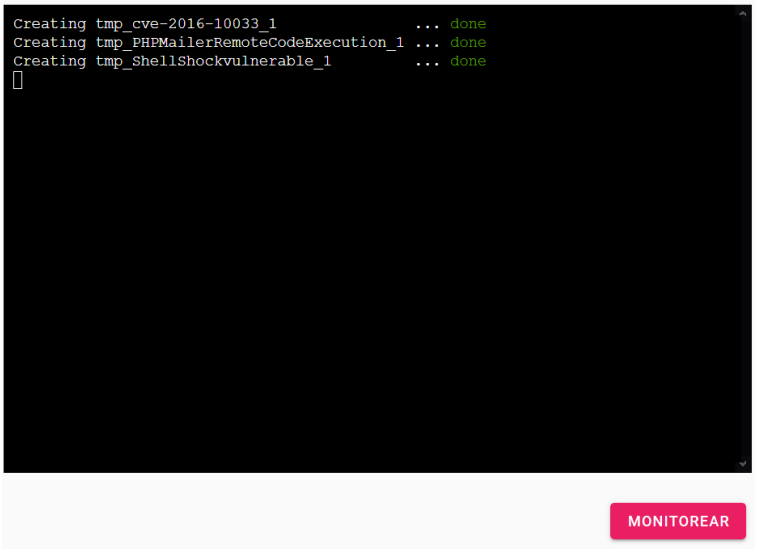

Figura 3.11: Terminal cuando el montaje termina correctamente, con el botón para monitorear el laboratorio.

3.3.8. Monitoreo de laboratorios

Tras haberse montado el laboratorio, el usuario puede ir a la interfaz de monitoreo. Esta interfaz se puede acceder en cualquier momento desde el menú lateral, y muestra automáticamente el estado del laboratorio activo. En caso de que no haya un laboratorio activo, le permite al usuario seleccionar uno de los anteriores. El elemento para eso es un select con autocompletado, por lo que el usuario puede escribir parte del nombre de la sesión para filtrarlas y encontrarlas más fácilmente.

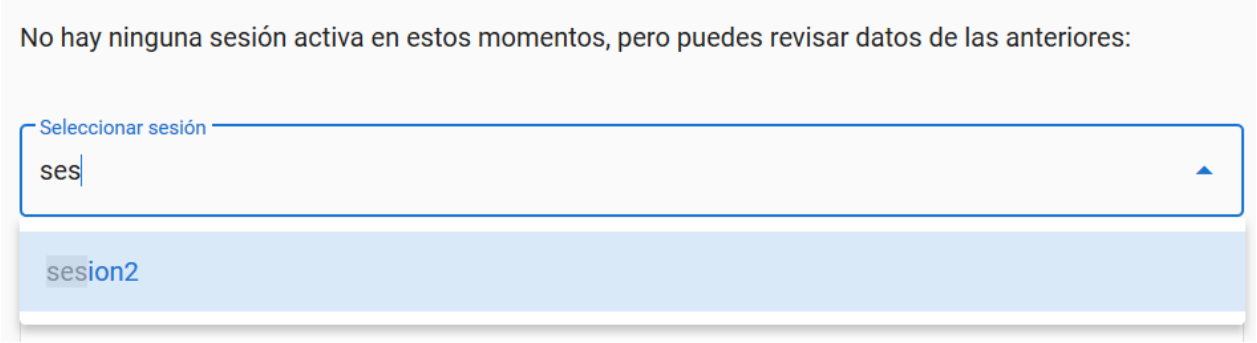

Figura 3.12: Selección de laboratorios anteriores para revisar su historial.

Una vez seleccionada una sesión (o si hay una en curso), el usuario puede ver un resumen general de las máquinas que se encuentran en la sesión. Cuando hay una sesión activa, el usuario es notificado cuando una máquina recibe nuevas peticiones o comandos, por si quiere revisarlas en detalle. Para ello, se utiliza un websocket para notificar al usuario de los nuevos comandos y peticiones. En el caso de los comandos, estos son registrados en un archivo en la máquina del servidor, por lo que el sistema de archivos es el encargado de notificarle a la aplicación que hay algo nuevo.

Para las peticiones, en cambio, se utiliza un mecanismo de la base de datos, ya que las peticiones se van registrando en ella. Al realizarse una nueva petición, esta se guarda, y se activa un gatillo que permite avisarle a la aplicación de este cambio. Esto se explica en más detalle en la sección 3.4.3.

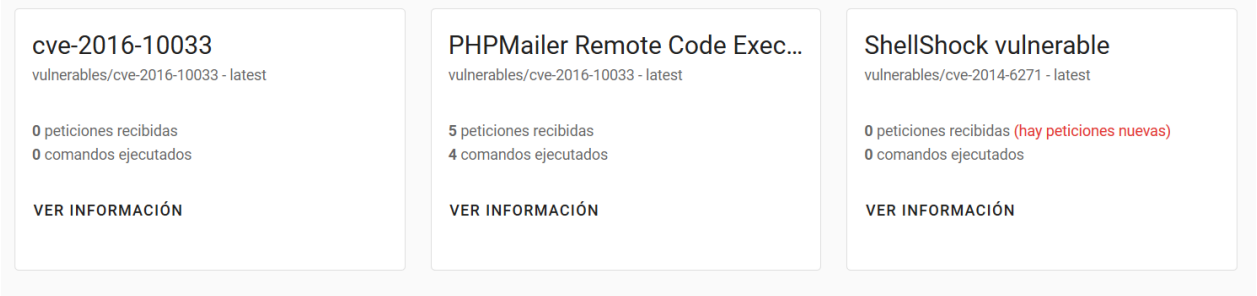

Figura 3.13: Ejemplo de la vista de las máquinas en una sesión en curso.

Además de la vista general, un usuario puede acceder a ver en mayor detalle las peticiones y comandos que ha recibido la máquina, apretando en el texto que dice *Ver información*. Desde ahí, el usuario puede ver cada petición y comando que han recibido las máquinas, y filtrarlas fácilmente para ocultar las que sean menos importantes (por ejemplo, un GET al directorio raíz), o revisar las que le llamen más la atención.

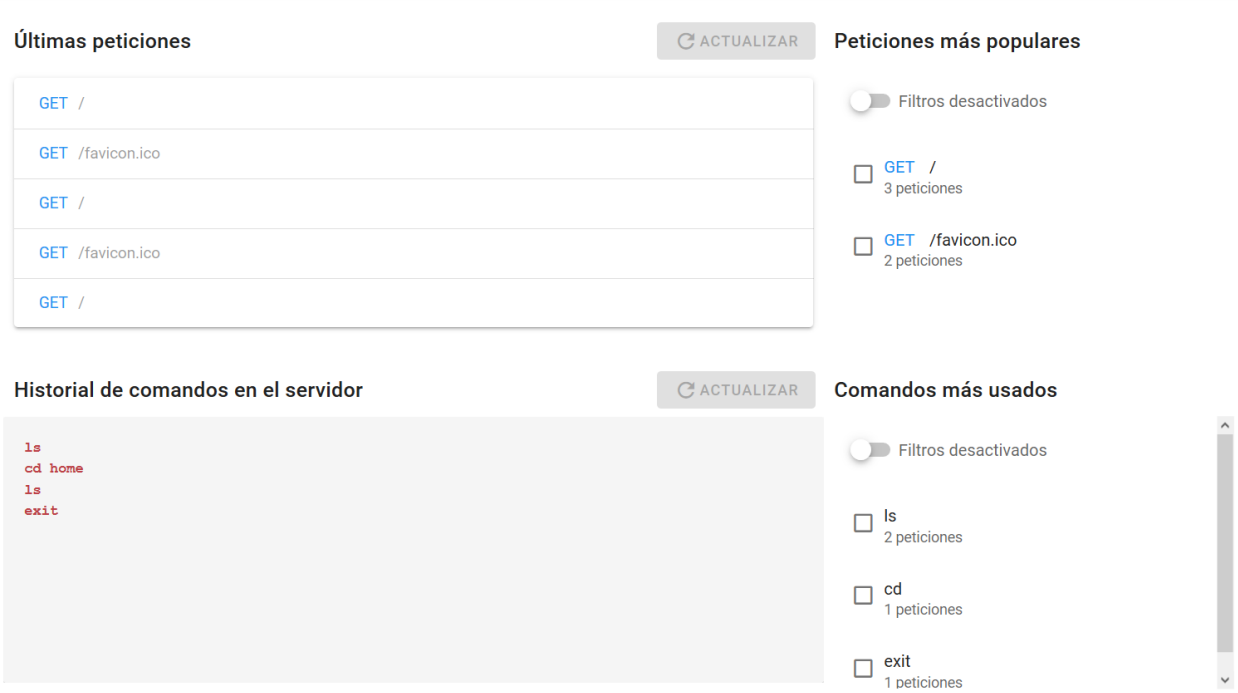

Figura 3.14: Ejemplo de la vista de las máquinas en una sesión en curso.

Cuando se han recibido nuevas peticiones o comandos desde la última vez que el usuario actualizó, el botón para actualizar se activa. Se consideró entregar las actualizaciones en tiempo real, pero esto podría entorpecer al usuario si es que se encuentra revisando algo en detalle. Para las peticiones POST, el usuario puede también revisar el contenido de la petición en más detalle.

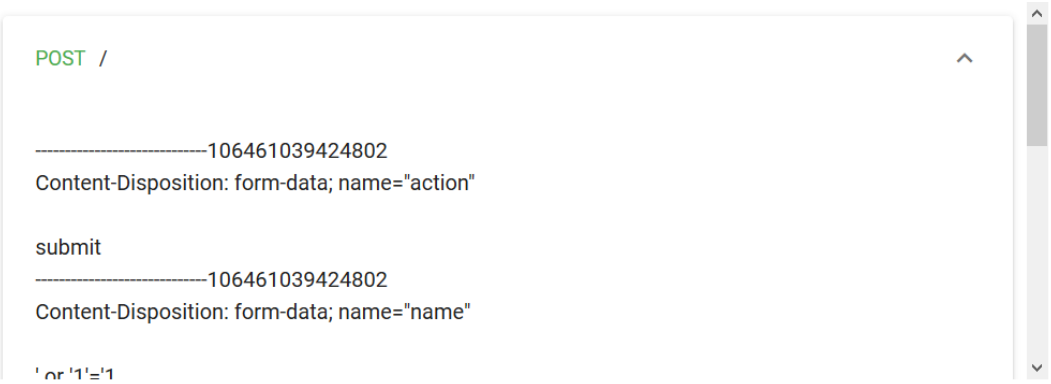

Figura 3.15: Contenido de un mensaje POST captado por la aplicación.

Inicialmente se había propuesto monitorear cualquier tipo de tráfico utilizando WireShark. Sin embargo, tras haber probado esto, se comprobó que no era tan viable, pues ese programa entrega los paquetes *RAW* del tráfico de las redes. Por lo tanto, para mostrarlo y analizarlo es necesario detectar el protocolo que se está utilizando, y procesar cada protocolo de forma especial. A esto se le suma otro problema: muchos de los protocolos usados para comunicación utilizan encriptación, y la finalidad de la herramienta no es romper esa encriptación. Por lo mismo, el monitoreo se realiza en las mismas máquinas, después de que estas desencriptan el tráfico entrante.

Se optó por monitorear este tipo de eventos (peticiones y comandos) porque son los más comunes para el tipo de máquinas utilizadas: en general tienen una aplicación web, y las vulnerabilidades se explotan a través de una petición maliciosa, o hay algún servicio de comunicación abierto, en cuyo caso, el usuario manda comandos para intentar obtener acceso. No obstante lo anterior, se podrían agregar nuevos protocolos al monitoreo sin tanto trabajo, dependiendo de cómo el protocolo registra sus intercambios.

3.4. Funcionamiento del servidor

Hasta ahora se ha descrito cómo funciona la aplicación de cara al usuario. Sin embargo, la mayor parte del trabajo de procesamiento se realiza en una aplicación que corre en el servidor. A continuación se describen las funcionalidades que se implementaron para esta aplicación, y de qué manera se utilizan para entregarle información al usuario. Para ello, se describirá el funcionamiento general del servidor, y cómo funciona cada método implementado en particular.

3.4.1. Funcionamiento general

El servidor utiliza Express.js para recibir las peticiones que manda el cliente. Para que los datos lleguen correctamente se utilizan las librerías *cors* y *body-parser* como middlewares.

La primera librería permite hacer *Cross-origin resource sharing*, es decir, permite que el servidor responda a peticiones de recursos aunque provengan de un dominio distinto. Esto permite que las peticiones se procesen correctamente, a pesar de que pasan por el servidor de la aplicación cliente, lo que hace que el encabezado *Origin* no coincida.

La segunda librería permite parsear automáticamente el cuerpo de las peticiones entrantes, para que la aplicación del servidor no lo tenga que hacer por su cuenta. En el caso de la aplicación desarrollada, las peticiones se pasan automáticamente a un JSON, que tiene soporte nativo para varios lenguajes, entre ellos JavaScript, que es el que se utiliza en la aplicación.

Al correrse la aplicación, lo primero que hace es conectarse a la base de datos MongoDB. La configuración inicial deja también a la aplicación esperando en el puerto especificado (en este caso, el puerto 8083), y con esperando una conexión entrante a través de un WebSocket.

3.4.2. Métodos de gestión

La configuración de la aplicación no permite que los usuarios accedan directamente a la base de datos, ya que esto podría usarse maliciosamente si es que el acceso a la red se configura mal, y una persona externa al laboratorio logra conectarse. Por ello, todos los métodos de gestión de la base de datos tienen que pasar por la aplicación del servidor. Debido a que la mayoría de estos métodos no tiene algo en especial, primero se describirán en general y, luego, se nombrarán con una breve descripción. Todos los métodos pueden fallar, en cuyo caso se devuelve un mensaje de error con un código de respuesta `40X`, y la aplicación sigue funcionando.

Conexión a la base de datos

La conexión inicial a la base de datos se realiza cuando la aplicación se corre por primera vez. Para utilizar correctamente los gatillos, que se utilizan para notificarle al usuario cuando ha llegado una nueva petición, la base de datos se corre con un *Replica Set* pre-configurado.

Un set de réplica no es más que un mirror automático que hace MongoDB de una base de datos, y está hecho para casos que requieran redundancia, o mantener los datos en más de un servidor para obtener mejores tiempos de respuesta, y su uso en este caso responde a una limitación con los triggers de MongoDB, más que a una decisión técnica.

POST /machine

Este método se encarga de agregar nuevas máquinas a la base de datos. Por la forma en que estas se hacen a nivel de usuario, se agregan de una a la vez. El esquema definido es el mismo que se describe en la sección 3.2.

GET /machines

Este método hace la consulta a la base de datos para obtener las máquinas guardadas, que se devuelven en orden inverso de inserción, es decir, las más nuevas se devuelven primero. De esta forma, si el usuario agrega una máquina, la puede encontrar más fácilmente, ya que se asume que si la agrega, es porque la quiere utilizar pronto.

DELETE /machine

Este método se llama con la ID de la máquina que se quiere eliminar en el cuerpo de la petición, y marca la máquina como eliminada, lo que deja de devolverla si se consulta por las máquinas. No necesita confirmación, porque se asume que eso está a cargo de la aplicación del cliente.

POST /getRequests

Este método recibe la ID de una máquina, y el nombre de una sesión, y devuelve las peticiones que ha recibido o recibió la máquina y que están asociados a esa sesión. Es un método POST y no GET simplemente por si es necesario evitar el largo máximo de una URL que podría encontrarse al hacerse una petición por GET.

POST /getRequestsCount

Hace lo mismo que el método anterior, pero en lugar de devolver las peticiones con toda su información, se devuelve el número recibido. La única finalidad es evitar procesamiento innecesario del lado del cliente.

POST /getPopularRequests

Nuevamente, es similar a los métodos anteriores, pero en este caso se utiliza *map reduce*, para agrupar las peticiones y contarlas, con la finalidad de entregarlas ordenadas. El mapeo se hace utilizando la máquina, sesión, ruta de la petición y el método de la petición.

GET /sessions

Método que devuelve la lista de sesiones anteriores. Se utiliza para poblar el menú de selección descrito en la sección 3.3.8.

3.4.3. Métodos y flujos complejos

Además de los métodos descritos anteriormente, hay otros métodos que se llaman dentro de flujos más complejos de la aplicación y, por lo tanto, se explicarán con el contexto de lo que está ocurriendo en la aplicación de cliente.

Búsqueda de tags para agregar máquinas

Cuando un usuario va a agregar una máquina nueva al sistema, como se describe en 3.3.3, el primer paso es ingresar un link de Docker Hub correspondiente a la imagen que se desea añadir. Cuando presiona el botón para buscar, se hace una petición al servidor, con el comando `GET /imageTags`.

Cuando se recibe esta petición, se llama a un método estático de una clase que parsea la URL. Esto es porque las URLs se diferencian si son "oficiales", y soportadas directamente por la compañía que mantiene Docker, o si han sido agregadas por otro usuario. Tras haber parseado la URL, se llama a otro método que se encarga de obtener los tags válidos.

Para ello, se hace otra petición GET a `/v2/repositories/${library}/${image}/tags`, en el servidor de Docher Hub. Library e image son variables que dependen del resultado del parser anterior. Tras completarse la solicitud, se devuelven las tags válidas para la imagen que se está consultando.

Generación de archivo Docker Compose

Cuando el usuario termina la configuración del laboratorio, el servidor genera un archivo Docker Compose para montar el laboratorio. Para ello, se hace una petición `POST /generateDocker`, con los puertos reservados, las máquinas y sus conexiones como parámetros.

La creación del archivo también está a cargo de otra clase, que genera el archivo por partes. El archivo tiene tres partes: encabezado, configuración de máquinas y configuración

de redes.

El encabezado es el mismo para cualquier archivo, y tiene solamente una línea para indicar la versión del archivo, según la especificación de Docker Compose que se use. Esto es `version: '3'`. Luego, se incluye la primera línea de configuración de máquinas, con una línea que dice "`services:`".

La siguiente sección es la de configuración de máquinas. Lo primero que se rellena es una línea que tiene el nombre de la máquina ingresado por el usuario, seguido del nombre de la imagen obtenido de la URL, y el tag correspondiente. Dentro del archivo también se incluye la configuración para montar un archivo local en el archivo `/root/history` de la máquina, con la finalidad de poder monitorear los comandos ingresados a la máquina. Junto a eso, se configura la variable de entorno `HISTFILE` para que efectivamente se use el archivo compartido.

Hay dos partes más de la configuración de cada máquina que se procesan por separado debido a su complejidad: la configuración de redes y de puertos.

La configuración de redes se hace a partir de una lista de pares de redes. Si la máquina actual está en la lista, se agrega la red a la configuración de la máquina actual. En caso de que la máquina no se conecte a ninguna otra, esta parte se omite. En este paso solo se declara a qué redes se va a conectar cada máquina, pero estas redes aún no se han configurado.

Luego se configuran los puertos, según lo especificado por el usuario al agregar la máquina al sistema. Para asignar los puertos, se va incrementando un contador que comienza en el último puerto asignado (o en 1, si es el primero por asignar), y termina al llegar a un puerto que no esté en la lista de puertos reservados. Esta última verificación se hace con una estructura de datos tipo HashSet, por lo que la búsqueda de puertos es lineal. En este paso también se configuran los puertos privados para que no sean accesibles por máquinas que no están conectadas.

Finalmente se especifican las redes. Para ello, simplemente se nombran numéricamente y de forma creciente, y se agregan al final del archivo, para que sean utilizables por las máquinas del laboratorio.

Montaje de máquinas

Para montar las máquinas, se crea una conexión mediante un WebSocket a la aplicación cliente. Luego, se crea un archivo temporal con la configuración del laboratorio, y se inicia un hilo de ejecución con el comando que monta todas la máquinas del archivo de configuración.

La salida de este hilo se escribe directamente al WebSocket, de forma de mostrarle al usuario el estado del montaje.

Inicio de monitoreo

Cuando se inicia el monitoreo, pasa lo siguiente:

1. Se envía un mensaje por un WebSocket al servidor, indicando que comenzará un mo-

nitoreo. Con ello, el servidor configura un observador de la base de datos, que avisa cuando se ha recibido una nueva petición. Cuando se recibe una nueva petición, se envía un WebSocket a modo de notificación a la aplicación del cliente. Además, se configura un observador para el archivo que guarda el historial de cada una de las máquinas.

2. Se manda una petición al servidor, a `POST /startRequestLoggingThreads`. Se toman los datos de las máquinas que se han montado, y se inicia un thread que corre la utilidad *tcpflow*, para monitorear la actividad de cada uno de los puertos que se están usando en el laboratorio. Otro thread corre un programa en python que procesa la salida de tcpflow, y parsea y guarda los requests en la base de datos. Esta llamada devuelve los PIDs de los procesos creados, para que se puedan detener cuando se termina de monitorear el laboratorio.

Finalización del monitoreo

Cuando la sesión termina, y el usuario lo indica, se termina de monitorear la actividad en tiempo real. Cuando esto ocurre, se manda un mensaje a través del WebSocket que se abrió al iniciar el monitoreo, con el mensaje `stopMonitoring`,

Cuando este mensaje es recibido por el servidor, se guarda el contenido de los archivos de monitoreo en la base de datos, y se matan los procesos creados para hacer el *logging*. Con ello, la aplicación vuelve a su estado inicial.

Capítulo 4

Validación

La validación se realizó mediante una sesión práctica con estudiantes del Departamento de Ciencias de Computación de la Universidad de Chile. Debido a que las condiciones materiales al momento de realizarse no eran las adecuadas, ya que había cuarentena por el Coronavirus, la sesión práctica se llevó a cabo de forma no presencial. Por lo tanto, los resultados obtenidos no necesariamente representan lo que ocurriría en una sesión presencial, pero sí son el mayor acercamiento que se pudo obtener.

A continuación se describe de forma general el instrumento utilizado para la validación, la descripción de la clase, y los resultados obtenidos de ella.

4.1. Preparación de la Clase

Para la clase práctica se consideran estudiantes que estudian computación o carreras relacionadas a la informática, con la finalidad de asegurar ciertos conocimientos básicos, que no vale la pena explicar en una sesión cortas. En particular, se asume algo de manejo de Bases de Datos y consultas SQL, así como también el uso de terminales.

La clase cuenta con dos partes: la primera es una clase expositiva, acompañada de una presentación de diapositivas, en la que se muestran algunos conceptos que pueden ayudar a los participantes a "resolver" las máquinas del laboratorio. No se les da respuestas explícitas, pero la idea tampoco es que dependan sólo de sus conocimientos previos. El material de apoyo de esta parte se puede encontrar en el Anexo 5.3.

Después de la clase expositiva, se le da a los participantes una URL desde la que deben acceder a las máquinas. La estructura del laboratorio es semi lineal. Hay dos máquinas que pueden encontrar al inicio, aunque hay una más fácil de resolver que la otra, y hay una *oculta*, a la que solo se puede acceder desde una red que conecta la conecta a la segunda máquina, como se puede ver en la sección 4.1.

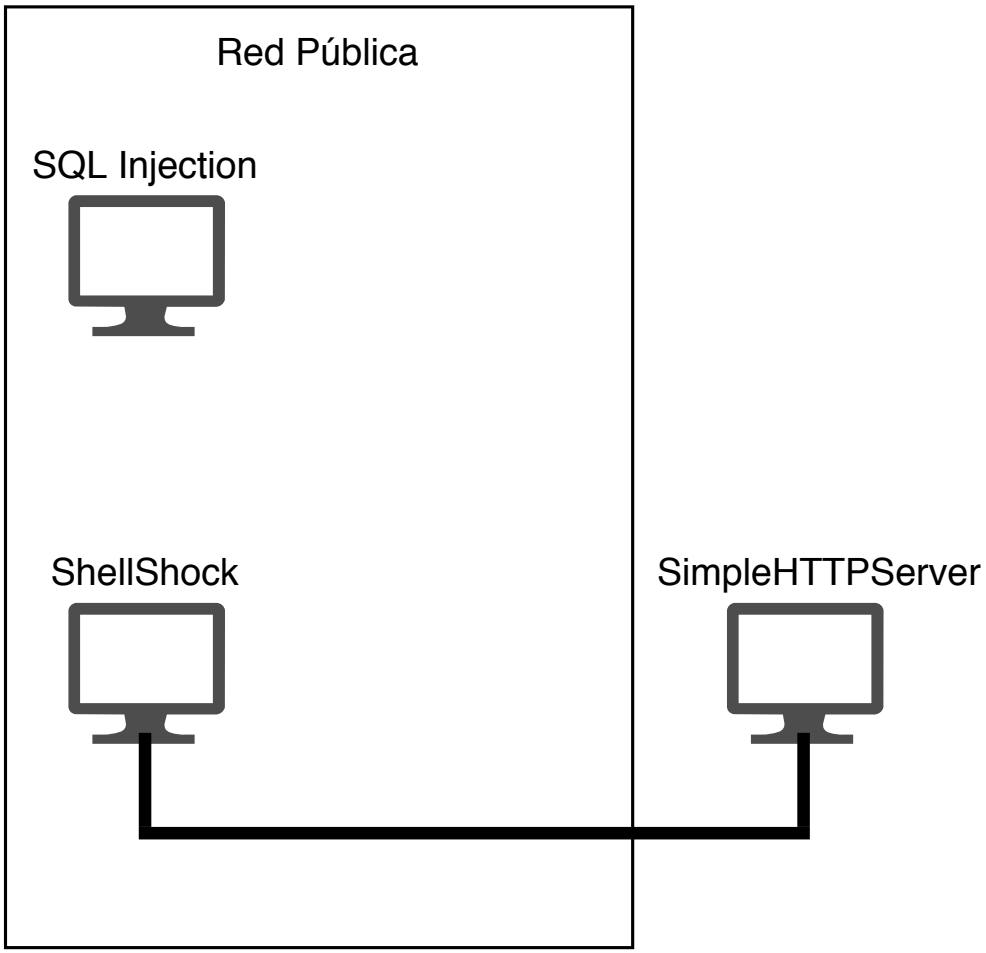

Figura 4.1: Diagrama del laboratorio de prueba

Las dos primeras máquinas tienen vulnerabilidades fáciles de aprovechar, y que son conocidas, pero que también son de las vulnerabilidades que más se repiten en las máquinas en producción. La máquina "SQL Injection" tiene una vulnerabilidad de inyecciones SQL, y la máquina "ShellShock" tiene una vulnerabilidad por usar componentes de software que se sabe que son vulnerables, y que no han sido actualizados. Ambas vulnerabilidades están en los puestos 1 y 9 del *top ten* de la OWASP[1], respectivamente [23]. Las máquinas utilizadas y sus vulnerabilidades se describen a continuación.

4.1.1. Máquina 1: SQL Injectable

La primera máquina sirve una aplicación web simple. El usuario puede ingresar un número, y la aplicación devuelve un elemento de una tabla en su base de datos que corresponde a ese número.

[1]OWASP es el Proyecto de la Seguridad en aplicaciones de la Web abierta, la fundación más importante que busca mejorar la seguridad del software. Su Top Ten es un estándar que representa los riesgos de seguridad más comunes en las aplicaciones web

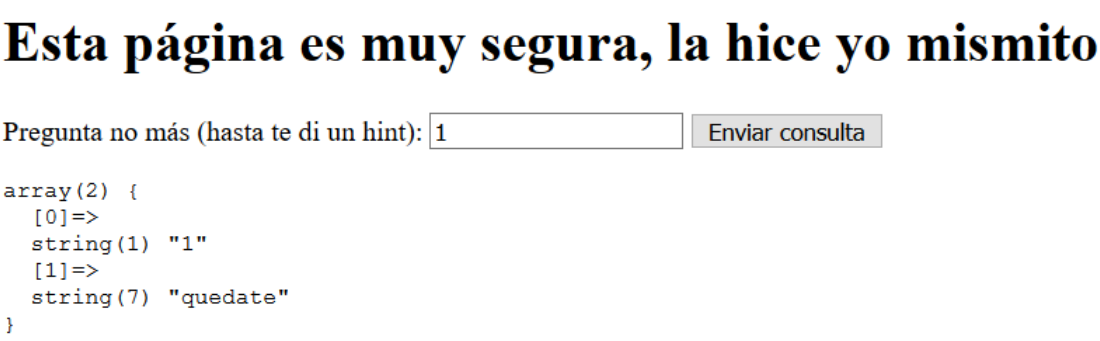

Figura 4.2: Funcionamiento de la primera máquina del laboratorio

El parámetro de la consulta se manda tal cual es ingresado por el usuario mediante un parámetro GET, por lo que lo más natural es que empiecen a probar cambiando los contenidos de la consulta ahí.

Como el nombre de la máquina lo indica, esta tiene una vulnerabilidad por inyección SQL. Esta vulnerabilidad se produce porque en el código de la aplicación, la consulta se hace con las siguientes líneas:

```
1 $sql = "select * from ‘words‘ where id = ’$id’;";
2 $res = $mysqli->multi_query($sql);
```

En el código hay dos problemas. El primero es el más común, y el que probablemente conocen los participantes, aunque no sepan mucho sobre inyecciones SQL. La consulta del usuario no está bien sanitizada (más adelante se explica que sí se está protegiendo para ciertas consultas), por lo que pueden, por ejemplo, ingresar `’ or ’1’=’1`. De esta forma, pueden obtener todas las entradas de la tabla *words*, ya que en todas se cumple que `’1’ = ’1’`.

Sin embargo, aún reconociendo esta vulnerabilidad, no acceden a toda la información contenida en la base de datos. Para hacerlo no les queda otra que probar algo más.

La siguiente consulta que se espera que prueben es una de tipo `0’;show tables;`. Con esto ya pueden darse cuenta que hay más de una tabla en la máquina, que pueden revisar sus columnas, e intentar cambiar la consulta para obtener datos de otra tabla.

Pero hay dos trabas más para hacer las consultas. Por ejemplo, si alguien intenta con `0’;select * from ‘noooo‘;`, obtendrá un mensaje que le indica que su consulta con Select fue bloqueada. Esto es porque la consulta se compara con la expresión regular `/select|update | delete|drop|insert|where|/i"`, es decir, cualquier consulta que tenga select, update, delete, drop, insert o where son bloqueadas, independiente de si son mayúsculas o minúsculas. El mensaje de error que se entrega, eso sí, tiene una pista para los participantes, pues se les entrega un mensaje que dice: “Te bloqueé la consulta, estoy más PREPARADO que tú ;)”. Esto hace referencia a las consultas preparadas que se pueden hacer en SQL.

Como la comprobación de la expresión regular es textual, se puede hacer la siguiente consulta: `0’;set @sql=concat(’selec’,’t * from ‘noooo‘’);prepare presql from @sql;`

`execute presql; deallocate prepare presql;`. Lo anterior hace que el chequeo falle, porque no está el texto *select* explícitamente. En este caso, se entrega un mensaje que dice que la consulta se bloqueó por contener el substring *prepare*. Lamentablemente, esta última medida de protección no revisa si son mayúsculas o minúsculas, por lo que se puede sobrepasar fácilmente.

Tras haber superado esta última medida de seguridad, los participantes obtienen una URL, que les confirma que completaron la primera máquina, y les da una pista para la segunda.

4.1.2. Máquina 2: ShellShockeable

La segunda máquina recibe a los participantes con la siguiente información:

Figura 4.3: Funcionamiento de la segunda máquina del laboratorio

Esta máquina tiene una vulnerabilidad conocida como Shellshock. En este caso, la dificultad no está en ver cómo funciona la aplicación, si no que en reconocer que es susceptible a este ataque. Lo que se busca con esta máquina, por lo tanto, no es que ocupen tiempo

intentando solucionarla, si no que reconocer que se utiliza CGI (Common Gateway Interface), relacionarlo a la versión de Apache que corre el servidor, e investigar vulnerabilidades conocidas de esa versión.

Al saber lo fácil que es explotar una vulnerabilidad de este tipo, la idea es que se den cuenta del peligro de correr software que no esté actualizado y, a la vez, de lo fácil que es solucionar una vulnerabilidad de este tipo cuando ya ha sido parchada por versiones más nuevas.

El directorio en que corre el script vulnerable lo pueden encontrar en un comentario oculto en el código fuente de la página que sirve la máquina. Tras encontrar el directorio, no es difícil darse cuenta que el sistema corre una versión obsoleta de Apache con mod-cgi, por lo que es vulnerable a ShellShock. Al encontrar la vulnerabilidad, basta con una línea para correr un script (con permisos de usuario):

```
curl -H ‘‘user-agent: ()  :; ; echo; echo; /bin/bash -c ’cat /etc/passwd’’
http://localhost:8080/cgi-bin/vulnerable
```

Cambiando el comando, pueden empezar a enumerar los directorios del sistema, hasta encontrar un archivo en el directorio `/home/` que les da la pista para entrar a la última máquina del sistema.

La última pista dice solamente “nmap”, que es una herramienta para hacer mapeo de redes. Corriendo el comando `ip a` pueden reconocer en qué subred se encuentra la máquina, para luego escanearlas con `nmap`. Con ello, pueden descubrir que hay una subred oculta con un servidor HTTP. Pueden usar `curl` para mandarle una petición, y con eso, son notificados de que completaron el laboratorio.

En resumen, el laboratorio busca que sus participantes aprendan lo siguiente:

1. Aprovechar una vulnerabilidad conocida, pero analizando por qué no pueden llegar al resultado tan fácilmente, y de qué manera pueden ajustar lo que ya saben para un nuevo escenario.
2. Investigar y buscar una vulnerabilidad que, si bien es conocida, es menos “popular”. Analizar los pocos datos que obtienen del laboratorio y relacionarlos para encontrar la vulnerabilidad en cuestión.
3. Reconocer que además de los servicios públicos que corren en un sistema, pueden haber otros servicios que se pueden acceder solo por las redes internas. Esto no significa que los servicios sean seguros, y se pueden acceder a través de los servicios públicos, si estos se ven comprometidos.

Debido a las limitaciones al hacer esta clase online, no se incluyó tiempo para incluir una parte en que los participantes se metían al código a solucionar sus problemas. Esta decisión se tomó porque darles acceso remoto para cambiar la configuración del servidor podría haber tenido un riesgo mayor que en un escenario sólo con acceso local.

4.2. Evaluación de la clase

Para la clase online se liberó un formulario de inscripción a estudiantes del DCC. El formulario fue contestado por 88 personas, de las cuales, 45 pudieron participar de la sesión en vivo.

A las personas que participaron del taller, se les hizo un formulario basado en los resultados de la plataforma NEMESIS [20]. Las respuestas usan una escalada de tipo Likert, es decir, van de *muy en desacuerdo* (1) a *muy de acuerdo* (5). Las preguntas realizadas son las siguientes:

1. Los contenidos de la clase abordan tópicos interesantes.
2. La parte práctica ayuda a consolidar conceptos y técnicas.
3. Las vulnerabilidades usadas podrían encontrarse en máquinas reales.
4. Hacer un laboratorio práctico aporta en tu aprendizaje.
5. Aplicar los contenidos disminuye las dudas posteriores sobre este.
6. La modalidad de la clase es preferible a una sólo presencial.
7. Participarías de otra sesión similar.

Los resultados de la encuesta posterior se pueden ver en la siguiente tabla:

Pregunta	Evaluación Promedio
Los contenidos de la clase abordan tópicos interesantes.	4.53
La parte práctica ayuda a consolidar conceptos y técnicas.	4.12
Las vulnerabilidades usadas podrían encontrarse en máquinas reales.	3.82
Hacer un laboratorio práctico aporta en tu aprendizaje.	4.65
Aplicar los contenidos disminuye las dudas posteriores sobre este.	4.35
La modalidad de la clase es preferible a una sólo presencial.	3.76
Participarías de otra sesión similar	4.65

Tabla 4.1: Resultados de la encuesta a participantes en el laboratorio

El desarrollo del laboratorio se llevó a cabo sin mayores complicaciones, y la plataforma se utilizó sin problemas. Las imágenes de la plataforma en uso para el monitoreo del taller se pueden encontrar en los anexos 5.1 y 5.2

Capítulo 5

Conclusión

En cuanto a la solución descrita en este informe, se puede decir que la plataforma desarrollada cumple con los objetivos descritos en este mismo.

La plataforma es capaz de configurar y montar automáticamente redes de imágenes de Docker, en particular, imágenes con vulnerabilidades conocidas, lo que permite que estudiantes aprendan de forma práctica habilidades relacionadas a la ciberseguridad.

En cuanto a los objetivos específicos planteados, se considera que estos también se cumplen con la solución propuesta. Tanto las máquinas como las redes que conforman se pueden configurar utilizando la plataforma desarrollada. Lo anterior permite que los laboratorios puedan ser armados de forma dinámica: tanto sus componentes como la topología de las redes se puede definir utilizando la misma interfaz.

Tras la configuración de los escenarios, el sistema genera un archivo Docker Compose. Este permite que los usuarios compartan el archivo, que puede ser utilizado por estudiantes que no hayan asistido al laboratorio.

La plataforma permite efectivamente monitorear fácilmente la actividad principal de quienes participan en el laboratorio, por lo que el objetivo relacionado a eso también se cumple.

Sobre los resultados de la validación, se puede decir que los tópicos abordados son interesantes para los estudiantes que participaron de la sesión. Esto probablemente se ve favorecido porque la inscripción al taller era voluntaria, aunque los temas específicos no fueron comunicados antes de partir el taller. Además, los ramos de estos tópicos en general también son electivos.

Acompañar los conocimientos teóricos con una sesión práctica fue bien evaluado por quienes participaron, como se puede ver en las preguntas 2 y 4 (ver sección 4.2). Según los resultados de la pregunta 5, también ayuda a disminuir sus dudas posteriores sobre el tema. Esto valida la aplicación desde el punto de vista de los estudiantes, que no ven directamente cómo funciona por detrás.

Las preguntas 3 y 6 también obtuvieron respuestas positivas, pero no tan fuertemente

como las demás. Las vulnerabilidades encontradas en las máquinas preparadas parecen ser realistas (considerando sistemas en producción). La clase es preferible a una solo presencial, pero tampoco tan fuertemente, probablemente debido a que las condiciones de esto dependen del tema, y si bien en este caso funcionó, no siempre tiene que ser así.

Finalmente, según lo que dijeron los estudiantes, participarían nuevamente en una sesión similar, lo que significa que, considerando el taller en su totalidad, están satisfechos con la experiencia.

Lamentablemente, dadas las condiciones de la validación, no se puede decir que los resultados son replicables bajo las condiciones de un semestre normal. De todas formas, los resultados son alentadores, ya que el taller le gustó a los estudiantes, y la plataforma actuó como un buen apoyo. Sirvió para monitorear lo que estaban haciendo, aún cuando no había tanta interacción directa como en una clase presencial.

Es posible, por lo anterior, que los resultados tengan que ser validados nuevamente bajo condiciones presenciales. De todas formas, esto no los hace menos válidos, y al menos son valiosos para las condiciones bajo las cuales se midieron (clases no presenciales).

La plataforma desarrollada cumple con todos los requisitos que se habían planteado inicialmente, y durante la sesión de prueba funcionó bien. No obstante, hay cosas que podrían mejorarse de la solución presentada. Monitorear la actividad de los participantes a la actividad fue algo fundamental para observar de qué manera estaban intentando acceder a las máquinas, y fue especialmente importante debido a la naturaleza de una sesión remota; no es posible observar las reacciones de los estudiantes, ni saber si están atascados en algo sólo viendo sus caras. Por lo mismo, lo primero que aparece como posible mejora del sistema es agregar otro tipo de actividad de los estudiantes. Si bien más del 90 % de la actividad en las máquinas corresponde a solicitudes HTTP, el resto también puede ser interesante de analizar, aunque no sea tan importante hacerlo en tiempo real.

Como trabajo futuro, también sería interesante estudiar de nuevo la posibilidad de agregar el *origen* de las solicitudes a la información de monitoreo. Esta se descartó por ahora, porque no es fácil determinar el origen correcto de un paquete cuando puede pasar por otra máquina intermedia antes de ser procesada. En este caso, el paquete tendría como origen esta máquina intermedia.

Otra posible mejora es agregarle al monitoreo un registro de toda la actividad en las redes de las máquinas, incluyendo los puertos utilizados. Esto se puede hacer utilizando herramientas como Wireshark. Si bien esta información no es tan fácil de procesar como las que se pueden acceder de la interfaz, puede dar información general de la actividad y se puede analizar de otras maneras.

Las tecnologías utilizadas cumplen bien con su propósito principal, que es hacer el desarrollo más fácil. Esto sin lugar a dudas se cumplió, porque pese a que la plataforma montada es compleja, se pudo desarrollar sin mayores problemas. Utilizar Vue.js con Vuetify fue lo que más aporto a desarrollar la plataforma más rápidamente.

Por otro lado, quizás la utilización de MongoDB como base de datos no fue la más acertada.

Si bien cumple con todo lo necesario para la plataforma, no tiene ventajas para este caso de uso con respecto a otras bases de datos más "tradicionales", y su utilización, por lo tanto, no se justifica completamente. Debido a la naturaleza de la aplicación, cambiar el sistema de manejo de bases de datos no es difícil, por lo que esto es una posible mejora a futuro para el sistema, aunque no es algo crítico.

Otra mejora posible sería añadir una opción para repetir una sesión anterior. Esto no sería un gran esfuerzo, ya que se puede reconstruir esto desde los mismos historiales guardados, y podría servir en caso de que un laboratorio se quiera repetir para secciones distintas, o en semestres diferentes.

Salvo los tres últimos puntos nombrados, a modo de cerrar el informe, y dado todo lo descrito en este, se puede decir que la aplicación desarrollada, si bien es mejorable, cumple con los objetivos que se plantearon en un principio de manera satisfactoria.

Bibliografía

[1] Tim Bray. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 8259, December 2017.

[2] Scott Chacon and Ben Straub. *Pro Git*. Apress, USA, 2nd edition, 2014.

[3] Eric Cole and Stephen Northcutt. Honeypots: A security manager's guide to honeypots. `https://web.archive.org/web/20171201034709/https://www.sans.edu/cyber-research/security-laboratory/article/honeypots-guide`, 2017. [Online, archivado el 1 de diciembre de 2017].

[4] World Wide Web Consortium. Web services architecture. `https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest`, 2004. [Online, accedido el 12 de noviembre de 2019].

[5] World Wide Web Consortium. Document object model. `https://www.w3.org/DOM/`, 2012. [Online, accedido el 1 de diciembre de 2019].

[6] Distrowatch. Damn vulnerable linux. `https://distrowatch.com/table.php?distribution=dvl`, 2019. [Online, accedido el 9 de abril de 2019].

[7] Inc. Docker. What is a container? `https://www.docker.com/resources/what-container`, 2019. [Online, accedido el 2 de diciembre de 2019].

[8] Neil Eliot, David Kendall, and Michael Brockway. A flexible laboratory environment supporting honeypot deployment for teaching real-world cybersecurity skills. *IEEE Access*, 6:34884–34895, 2018.

[9] Roy T. Fielding and Julian Reschke. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230, June 2014.

[10] Node.js Foundation. Express. `https://expressjs.com/`, 2018. [Online, accedido el 2 de diciembre de 2019].

[11] Node.js Foundation. About node.js. `https://nodejs.org/en/about/`, 2019. [Online, accedido el 2 de diciembre de 2019].

[12] The Wireshark Foundation. Wireshark. `https://www.wireshark.org/`, 2019. [Online, accedido el 9 de abril de 2019].

[13] Jill Gordon. One to one teaching and feedback. *BMJ*, 326(7388):543–545, 2003.

[14] TJohn Gossman. Introduction to model/view/viewmodel pattern for building wpf apps. `http://web.archive.org/web/20080130093754/http://blogs.msdn.com/johngossman/archive/2005/10/08/478683.aspx`, 2005. [Online, archivado el 30 de enero de 2008].

[15] Docker Inc. Compose file version 3 reference. `https://docs.docker.com/compose/compose-file/`, 2019. [Online, accesido el 14 de abril de 2019].

[16] Davis Kerby. Why mongodb is the way to go. `https://dzone.com/articles/why-mongodb-is-worth-choosing-find-reasons`, 2015. [Online, accedido el 1 de diciembre de 2019].

[17] Virtual Hacking Labs. Penetration testing lab. `https://www.virtualhackinglabs.com/labs/penetration-testing-lab/`, 2019. [Online, accedido el 14 de abril de 2019].

[18] A. Lanoy and G. W. Romney. A virtual honey net as a teaching resource. In *2006 7th International Conference on Information Technology Based Higher Education and Training*, pages 666–669, July 2006.

[19] Abhishek Mairh, Debabrat Barik, Kanchan Verma, and Debasish Jena. Honeypot in network security: a survey. In Sanjay Kumar Jena, Rajeev Kumar, Ashok Kumar Turuk, and Manoranjan Dash, editors, *Proceedings of the 2011 International Conference on Communication, Computing & Security, ICCCS 2011, Odisha, India, February 12-14, 2011*, pages 600–605. ACM, 2011.

[20] Ivan Marsá-Maestre, Enrique de la Hoz, José Manuel Giménez-Guzmán, and Miguel A. López-Carmona. Design and evaluation of a learning environment to effectively provide network security skills. *Computers & Education*, 69:225–236, 2013.

[21] Alexey Melnikov and Ian Fette. The WebSocket Protocol. RFC 6455, December 2011.

[22] Sun Microsystem. Distributed application architecture. `https://web.archive.org/web/20171201034709/https://www.sans.edu/cyber-research/security-laboratory/article/honeypots-guide`, 2003. [Online, archivado el 6 de abril de 2011].

[23] OWASP. Owasp top ten. `https://owasp.org/www-project-top-ten/`, 2017. [Online, accedido el 21 de marzo de 2020].

[24] PracticalPentestLabs. Practical pentesting lab. `https://practicalpentestlabs.com/`, 2019. [Online, accedido el 14 de abril de 2019].

[25] Roger Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, Inc., USA, 7th edition, 2009.

[26] Leonard Richardson and Sam Ruby. *RESTful Web Services*. O'Reilly, Beijing, 2007.

[27] Dewhurst Security. Damn vulnerable web application. http://www.dvwa.co.uk/, 2019. [Online, accedido el 14 de abril de 2019].

[28] Maven Security. Web security dojo. https://sourceforge.net/projects/websecuritydojo/files/, 2017. [Online, accesido el 14 de abril de 2019].

[29] Offensive Security. Virtual penetration and security testing hosting labs. https://www.offensive-security.com/offensive-security-solutions/virtual-penetration-testing-labs/, 2019. [Online, accedido el 14 de abril de 2019].

[30] Amazon Web Services. Aws customer agreement. ttps://aws.amazon.com/agreement/, 2018. [Online, accesido el 9 de abril de 2019].

[31] Shodan. Honeypots: A security manager's guide to honeypots. https://honeyscore.shodan.io/, 2017. [Online, accedido el 9 de abril de 2019].

[32] Pavol Sokol and Maros Andrejko. Deploying honeypots and honeynets: Issues of liability. In Piotr Gaj, Andrzej Kwiecien, and Piotr Stera, editors, *Computer Networks - 22nd International Conference, CN 2015, Brunów, Poland, June 16-19, 2015. Proceedings*, volume 522 of *Communications in Computer and Information Science*, pages 92–101. Springer, 2015.

[33] ThreatStream. Modern honey network. https://threatstream.github.io/mhn/, 2019. [Online, accedido el 9 de abril de 2019].

[34] Vue.js. Components basics. https://vuejs.org/v2/guide/components.html, 2016. [Online, accedido el 1 de diciembre de 2019].

[35] Vue.js. Reactivity in depth. https://vuejs.org/v2/guide/reactivity.html, 2016. [Online, accedido el 1 de diciembre de 2019].

[36] Vue.js. Template syntax. https://vuejs.org/v2/guide/syntax.html, 2016. [Online, accedido el 1 de diciembre de 2019].

Anexos

Anexo A

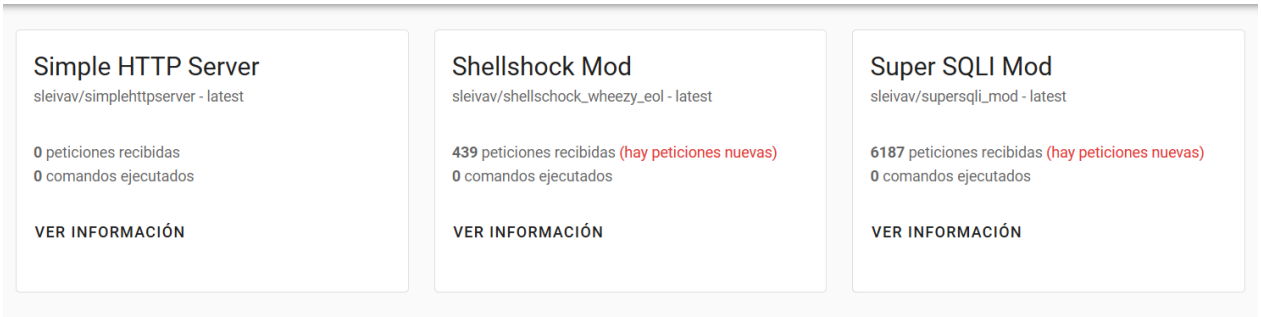

Figura 5.1: Estadísticas generales de accesos durante el taller

Anexo B

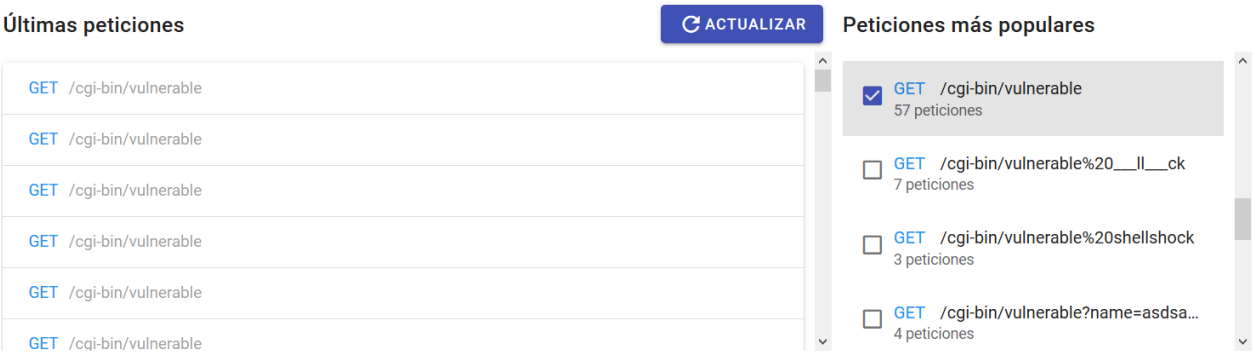

Figura 5.2: Detalles de solicitudes a una máquina durante el taller

Anexo C

slides_taller.pdf [1]

Taller (básico) de CiberSeguridad

slides_taller.pdf [2]

¡Hola!

Me llamo Sergio

Me pueden decir Checho

(Les pido perdón, pero esto tendrá harto texto :(es para la gente que no esté en vivo)

2

slides_taller.pdf [3]

1.

¿Qué aprenderán hoy?

O qué me gustaría que aprendan

3

slides_taller.pdf [4]

Aprendizajes esperados

- Reconocer una vulnerabilidad simple y aprovecharla
- Utilizar pocas "pistas" disponibles para reconocer qué servicios usa un sistema
- Reconocer diferencias entre servicios públicos y privados
- Y ojalá aprender a cuidar nuestros servicios

4

slides_taller.pdf [5]

¿Qué haremos?

- Una clase cortita (ojalá ~20 minutos)
  - ¿Por qué es importante saber de seguridad?
  - Inyecciones
  - Mapeo de redes y servicios
  - **Todo** (o casi) les servirá para el laboratorio
- Laboratorio con 3 máquinas (ojalá <40 minutos, pero no hay límite si están entretenides)

5

slides_taller.pdf [6]

2.

¿Por qué es importante saber de CiberSeguridad?

6

slides_taller.pdf [7]

Aprender sobre CiberSeguridad

- Es entretenido (quizás tengo sesgo)
- Nos permite **proteger** los datos de quienes usen nuestros sistemas

7

slides_taller.pdf [8]

3.

Inyecciones de código

Y por qué no hay que confiar en la gente uwu

8

slides_taller.pdf [9]

Inyecciones de código

- La mayoría de las aplicaciones son interactivas, y tienen contenido dinámico
- Alguien con malas intenciones puede aprovechar esto (si su consulta no es *sanitizada*)
- Las más comunes son *inyecciones SQL*

9

slides_taller.pdf [10]

Inyecciones SQL

- Una consulta SQL común se ve así:

```
"select * from `words` where id = '$id';"
```

- ¿Qué pasa si en la consulta `id = 1' or '1'='1`?
- ¿Qué pasa si `id = 1'; show tables;'`?

10

slides_taller.pdf [11]

Inyecciones SQL

- En el primer caso podíamos obtener información que no nos correspondía
- Si hubiera sido una consulta para iniciar sesión, podríamos haber iniciado sin tener cuenta
- Si el mecanismo de consulta hace varias *queries*, podemos también obtener información de otras tablas

11

slides_taller.pdf [12]

¿Cómo nos protegemos?

- En general, cualquier lenguaje decente tiene formas de procesar las consultas, y evitar este tipo de ataques.
- Esto es mejor que intentar filtrar las consultas con palabras clave (por ejemplo). Esto no sirve si el usuario hace una consulta en que concatena strings, hay que cuidar las mayúsculas, etc.
- Si la consulta se hace con un *prepared statement*, se procesa antes de ingresar la variable.

Ejemplo pseudocódigo

```
statement = db.prepare(
    "select *
    from `words`
    where id = (?)"
)
statement.execute(id='3')
```

12

slides_taller.pdf [13]

4. Mapeo de redes y servicios

13

slides_taller.pdf [14]

Reconocer servicios

- En general, no sabemos qué está corriendo un servidor. Pero si está público, podemos averiguarlo.
- Si apretamos F12 en el navegador, y vamos a la pestaña de conexiones, algunas páginas nos muestran información en las cabeceras

Server: Apache/2.2.22 (Debian)

14

slides_taller.pdf [15]

Reconocer servicios

- Por ejemplo, si sabemos que una versión de Apache con `mod_cgi` es vulnerable a cierto ataque, podríamos aprovechar esta vulnerabilidad.
- Si tenemos la IP de un servidor, podemos escanear todo lo que está abierto, revisando IP por IP
- Hay herramientas como `nmap` que pueden ayudarnos a revisar automáticamente

15

slides_taller.pdf [16]

Reconocer servicios

- Si alguien tiene acceso a uno de mis sistemas, puede usarlo para mapear redes internas
- No hay que asumir que por tener algo "oculto" esto es seguro. Incluso servicios internos deben tener permisos configurados

16

Figura 5.3: Diapositivas de apoyo usadas durante el taller