



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA TELEMÁTICA

TRABAJO DE FIN DE GRADO

SEGURIDAD EN SISTEMAS Y REDES:
PLATAFORMA E-LEARNING LIBRE PARA
CIBERSEGURIDAD BASADA EN MECÁNICAS DE JUEGO

Autor: José Enrique Narváez Gago

Tutor: Enrique Soriano Salvador

Curso académico 2015/2016



©2016 José Enrique Narváez Gago

Esta obra está bajo una licencia de “Creative Commons Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional”.

To see a copy of this licence, visit
<http://creativecommons.org/licenses/by-nd/4.0/legalcode> or send a letter to Creative Commons, 171 Second Street, Suite 300,
San Francisco, California 94105, USA.

Agradecimientos

Quiero agradecer a toda la gente que me ha ayudado a lo largo de mi experiencia en este Grado.

En primer lugar, al tutor de este proyecto (y el profesor que más clases me ha impartido), Enrique Soriano Salvador, por guiarme en el desarrollo de este proyecto y alejarme del camino del mal en la Ciberseguridad.

También me gustaría agradecer a todos los responsables de los laboratorios Linux, los cuales a lo largo de estos cuatro años han sido una herramienta invaluable para mi aprendizaje en estos entornos.

Además, a mi familia por apoyarme todo este tiempo. A mi padre por guiarme y convertirme en la persona que soy actualmente. A Beatriz, mi prima Ana Cristina y Miguel por animarme y preocuparse por mí y servir de mediadores entre mi padre y yo. Y por último a Ana por estar a mi lado.

A mis compañeros de BTC fácil por ayudarme a escribir esta memoria y animarme a probar nuevas tecnologías.

Y en último lugar, agradecer a todos los amigos que he hecho a lo largo de estos años en la universidad, especialmente a Antonio por probar todos los desafíos que he desarrollado a lo largo de este proyecto y hacerme tele-compañía todos los veranos.

¡Gracias!

Resumen

La ciberseguridad es un campo de las ciencias de la información en auge, debido al creciente número de amenazas y ataques en la industria. Aun así, generalmente es necesario que cualquier persona interesada necesite estudiar y practicar por su cuenta para obtener los conocimientos necesarios en dicho campo.

Hoy en día no es suficiente con tener conocimientos acerca de medidas defensivas, cifrado, comunicaciones seguras, etc. Es necesario saber cómo piensa y actúa un atacante, para entender mejor sus técnicas y, por consiguiente, poder defendernos y reaccionar con mayor velocidad y eficacia.

Debido a estos dos factores, y con el fin de evitar incurrir en una ilegalidad, los interesados en la seguridad ofensiva recurren a entornos vulnerables, que permiten su explotación de manera expresa.

Gracias a estos entornos vulnerables aparecen los CTF (Capture The Flag), competiciones diseñadas para servir de ejercicio educacional. Hay muchos tipos de CTF: por equipos, de defensa y ataque, individuales, etc.

Este proyecto propone una forma de organizar dichas competiciones, orientada a un entorno académico, proporcionando un entorno de trabajo (Framework), el cual permite a un programador crear y compartir pruebas para realizar competiciones de manera independiente.

En el anexo I se puede encontrar la URL del repositorio donde se encuentra el código fuente de la plataforma y un enlace para descargar las pruebas desarrolladas en este proyecto.

Índice general

1. Introducción	1
1.1. Ciberseguridad	2
1.2. Seguridad Ofensiva	2
1.3. Pentesting	3
1.4. Kali Linux	6
1.5. Metasploit	6
1.6. Formación en Ciberseguridad Ofensiva	7
1.7. CTF	8
2. Objetivos	11
2.1. Descripción del Problema	11
2.2. Objetivos	11
3. Descripción del trabajo desarrollado	13
3.1. Metodología	13
3.2. Iteraciones	14
3.3. Infraestructura	15
3.3.1. Go	15
3.3.2. net/http	16
3.3.3. html/template	18
3.3.4. MySQL	18
3.3.5. Vagrant	19

3.3.6. Docker	19
3.4. Desarrollo de la plataforma	20
3.4.1. Código Fuente	21
3.4.2. Esquemas de base de Datos	23
3.5. Desarrollo de Retos	26
3.5.1. Prueba de enunciado: Le xorffre indéchiffrable	27
3.5.2. Prueba de enunciado: Para el cesar lo que es del cesar	28
3.5.3. Prueba de enunciado: Such challenge	28
3.5.4. Prueba de entorno compartido: Wargame1, level01	29
3.5.5. Prueba de entorno compartido: Wargame1, level02	30
3.5.6. Prueba de entorno compartido: Wargame1, level03	31
3.5.7. Prueba de entorno compartido: Wargame1, level04	32
3.5.8. Prueba de enunciado: Fear The EXIF	33
3.5.9. Prueba de enunciado: La Venganza De Atahualpa	35
3.5.10. Prueba de enunciado: Telegram PoC	35
3.5.11. Prueba de entorno individual: Deface Me	36
3.5.12. Pruebas de enunciado: Retos Criptored	38
3.5.13. Prueba de entorno compartido: Metasploitable 2	38
3.6. Resultado Final	39
3.6.1. Uso Como Participante	39
3.6.2. Uso Como Diseñador de Pruebas	42
3.6.3. Fichero .ctff	44
3.6.4. Uso Como Administrador	45
4. Conclusiones	46
4.1. Conclusiones	46
4.2. Competencias Utilizadas y Adquiridas	47
4.3. Trabajo Futuro	48

<i>ÍNDICE GENERAL</i>	1
Bibliografía	50
Anexo I: URL de descarga	54

Capítulo 1

Introducción

La seguridad informática, o ciberseguridad, es un campo que históricamente se ha dejado de lado. En las empresas ha primado la cantidad a la calidad y a los desarrolladores se les ha presionado para conseguir resultados visibles en el menor tiempo posible. Esto puede ocasionar errores, ya sea por las prisas del desarrollador, la falta de experiencia o, directamente la falta de interés del mismo. Esos errores pueden poner en riesgo la integridad del sistema que han desarrollado de distintas maneras.

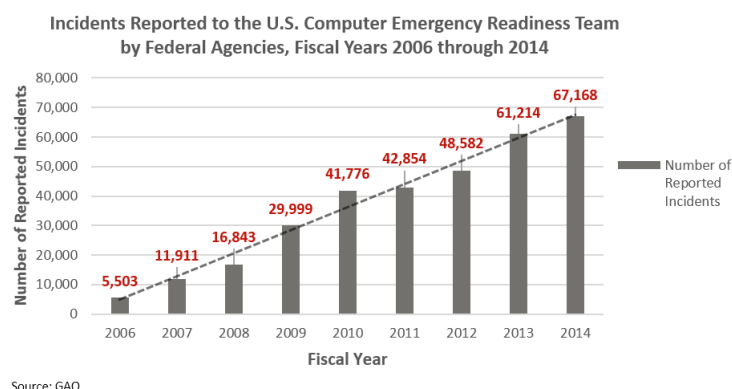


Figura 1.1: Incidentes reportados al Equipo de Preparación de Emergencias en Computadores de USA por Agencias Federales, Años Fiscales 2006 al 2014

A lo largo de los años, el número de incidentes de ciberseguridad sólo ha ido en aumento, y se espera que siga así en los años venideros. La necesidad de profesionales especializados en ciberseguridad es innegable y mercado el es consciente, la demanda de técnicos e ingenieros con conocimientos en este sector es mucho mayor que las personas capacitadas a desempeñar dichos trabajos.

Además, en el sector de la ciberseguridad se le está dando cada vez más importancia a profesionales capaces de realizar pruebas a un sistema, utilizando las mismas técnicas que

utilizaría un atacante externo. Esto es conocido como *tests* de penetración (*pentesting*) o “hacking ético” (no confundir con “hacktivismo”).

Si a este aumento de demanda le añadimos el hecho de que, generalmente, los estudios de Grado no profundizan en este tipo de conocimiento, nos encontramos con una necesidad de formación en la ciberseguridad. Por ejemplo, cuando alguien quiere aprender a programar, puede buscar información en libros y practicar, obteniendo resultados tangibles a lo largo del tiempo. En cambio, en el campo de la seguridad informática, es complicado poner en práctica tus conocimientos sin incurrir en una ilegalidad.

La alternativa que se ha generado en el mundo de la ciberseguridad, ha sido el desarrollo de entornos vulnerables (aplicaciones web, programas ejecutables, máquinas virtuales), cuyo objetivo es ser atacados para poner en práctica dichos conocimientos. Aunque todavía son escasos, la cantidad de este tipo de entornos va en aumento.

En este capítulo, se intentará dar un poco más de contexto en los términos, técnicas y tecnologías relacionadas, como también algunos ejemplos de su uso.

1.1. Ciberseguridad

La ciberseguridad, es la protección de los sistemas de la información de robo o daño del *hardware*, *software* y los datos. Esto incluye controlar el acceso físico al *hardware* y proteger de ataques, externos o internos, ya sea por un descuido de un operador, por un accidente provocado por catástrofes naturales, o porque se convenza a un usuario para realizar acciones que puedan beneficiar al atacante. En resumen, la ciberseguridad consiste en proteger activos frente a amenazas.

Históricamente, la ciberseguridad se ha basado en proteger las fronteras de nuestro entorno y entorpecer a los atacantes, ya sea con técnicas de seguridad perimetral (*Firewalls* y zonas desmilitarizadas), división de redes con **VLANs**, sistemas de detección de intrusos (*IDS*) entre muchas otras. Aún los ataques han evolucionado a una velocidad que los sistemas defensivos no han sido capaces de replicar.

1.2. Seguridad Ofensiva

Debido al aumento de ataques y su creciente complejidad, ha surgido la necesidad de aprender las técnicas más comunes realizadas por los atacantes, para entender mejor

al enemigo y reaccionar en consecuencia. Se suele decir que la ciberseguridad nunca es absoluta, pero al entender las técnicas de los atacantes, utilizarlas contra nosotros mismos y estudiar los resultados, podemos protegernos de una manera más eficiente y efectiva.

El trabajo de un auditor de seguridad consiste principalmente en la verificación de estándares, análisis de servicios y aplicaciones, detección y comprobación de vulnerabilidades, enumeración de medidas para la corrección y de recomendaciones de medidas preventivas para el futuro.

Los medios de comunicación han dado una mala fama al colectivo *hacker* ya que se suelen enfocar en los miembros del mismo que rompen la ley y realizan actos vandálicos. Un *hacker* es todo individuo que se dedica a programar de forma entusiasta o un experto entusiasta de cualquier tipo, que considera que poner la información al alcance de todos constituye un extraordinario bien [1]. Dentro de este colectivo se encuentran varios perfiles diferentes, como por ejemplo:

- **Hacker de sombrero blanco.** Motivación legítima, *hacking* ético. Prueba su propio sistema o el de otro, por el que está contratado, con autorización previa explícita.
- **Hacker de sombrero gris.** Invade un sistema sin autorización del responsable, notifica posteriormente que ha podido burlar la seguridad.
- **Hacker de sombrero negro (Cracker).** Es un delincuente que realiza actividades de vandalismo, fraude, robo de identidad o piratería.

Debido a esta mala fama, este tipo de puestos de trabajo (*hacker* de sombrero blanco) siempre ha estado estigmatizado. Con el paso de los años, se ha ido aceptando en la industria hasta el punto de ofrecerse un gran abanico de certificaciones para poder demostrar conocimientos (CEH, CPHE, OSCP, etc.)

1.3. Pentesting

La palabra *pentesting* es una palabra compuesta que proviene de las palabras *penetration* y *testing*, es decir; cuando hablamos de *pentesting* nos referimos a “pruebas de penetración”.

Una prueba de penetración es un ataque controlado y deseado, realizado generalmente por un auditor de seguridad, con el fin de descubrir que resultados podría obtener un atacante a la hora de realizar un ataque.

El *pentesting* está formado por varias etapas, que pueden ser agrupadas de diferentes formas. Una forma de distribuir las etapas sería la siguiente:

1. **Recolección de información:** en esta etapa, se realiza un perfil de la organización que se está auditando. Es común buscar información en buscadores (Google, Bing, Shodan [2]), sitios que brindan información sobre dominios, utilizar herramientas específicas como Maltego [3], intentar extraer información (nombres de usuario, direcciones IP, nombres de máquina) de metadatos en ficheros disponibles online, e incluso, realizar pruebas de Ingeniería Social. La Ingeniería Social consiste en aprovecharse de las vulnerabilidades del ser humano (apelar a la bondad, deseo de ayudar, lástima o credibilidad) para obtener información que no podríamos conseguir de otra forma. Por ejemplo, realizando una llamada a un empleado de la organización haciéndonos pasar por un administrador para pedir información específica acerca de un equipo (versión de antivirus, dirección IP, etc). Esta fase se suele considerar no intrusiva, ya que no estamos atacando a ningún sistema para conseguir esta información. Las próximas etapas se consideran intrusivas y no deben ser realizadas sin el consentimiento de la organización auditada.
2. **Escaneo de Puertos o Enumeración de Servicios:** esta etapa consiste en el uso de herramientas para obtener información acerca de los equipos de la organización: puertos abiertos, servicios que corren en esos puertos, *firewalls*, puntos de acceso, sistemas operativos. Lo más común es utilizar herramientas como Nmap[4] para esta etapa. Nmap es una herramienta que permite realizar analizar los puertos y servicios de direcciones IP específicas o rangos de direcciones. Además, intenta obtener información acerca de la versión de los servicios encontrados, sistema operativo, etc.
3. **Análisis de Vulnerabilidades:** esta etapa y la anterior pueden considerarse como una misma. Aquí nuestro objetivo es investigar si algún servicio encontrado anteriormente contiene una vulnerabilidad conocida. Las vulnerabilidades se dividen por su severidad, teniendo en cuenta su impacto en el sistema y la facilidad con la que se puede explotar. Esta etapa es muy importante, ya que muchos ataques suelen ser realizados a vulnerabilidades conocidas. Generalmente este análisis se realiza con escáneres de vulnerabilidades como por ejemplo Nessus [5] u OpenVAS [6].
4. **Explotación:** una vez obtenida la información necesaria, se procede a atacar de verdad al sistema. Estos ataques pueden realizarse de manera manual o

automatizada. Al atacar de manera automatizada se suelen utilizar *exploits*: programas que aprovechan cierta vulnerabilidad para conseguir un comportamiento no deseado del mismo. Por ejemplo, se puede obligar a que se detenga un proceso, conseguir acceso remoto al equipo, etc. Hay herramientas como **Metasploit** [7] que permiten automatizar el proceso de explotación de un equipo, hablaremos acerca de esta herramienta más adelante. Hay sitios como exploit-db.com (mantenido por **Offensive Security** [8]) donde podemos encontrar *exploits* para un amplio abanico de vulnerabilidades y, además, la base de datos de *exploits* se mantiene actualizada gracias a la colaboración de la comunidad.

5. **Post-Explotación:** si hemos conseguido acceso a un sistema, dependiendo del acuerdo realizado con la organización auditada, en esta fase se suelen realizar acciones de eliminación de *logs*, instalación de *backdoors* (puertas traseras), etc. En caso de que hubiese nuevas redes accesibles desde la máquina vulnerada, se debe realizar un nuevo proceso de *pentesting* a todas las nuevas máquinas descubiertas. Esto es vital ya que, generalmente las organizaciones enfocan su seguridad en la parte externa como se ve en la siguiente figura. Si lográsemos vulnerar el *router* interior podríamos acceder a una red interna que, generalmente, no está preparada para recibir ataques.

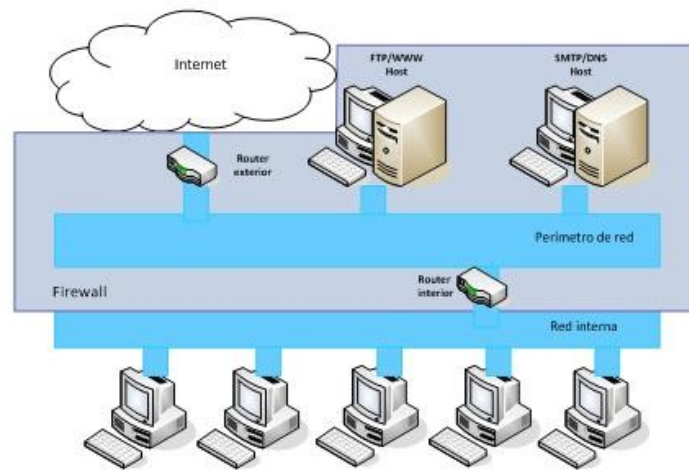


Figura 1.2: Esquema de red de una organización

6. **Reporte:** una vez realizada la auditoría, se realiza un informe que contiene un resumen ejecutivo, dirigido a personas que no necesariamente tienen conocimientos técnicos, con el fin de informar acerca de los resultados obtenidos. Además de este

resumen, se escribe un reporte técnico en el cual se explican las técnicas utilizadas, la información obtenida, las vulnerabilidades encontradas, y los activos implicados en la explotación. Este reporte además incluirá consejos cuyo fin es arreglar los fallos de seguridad encontrados.

1.4. Kali Linux

Kali Linux [9] es un proyecto código abierto mantenido y fundado por **Offensive Security**, que consiste en una distribución Linux con diferentes herramientas útiles para auditores de seguridad preinstaladas.

Dicha distribución se mantiene constantemente en desarrollo, encontrándose actualmente en su versión 2.1. Anteriormente, **Offensive Security** trabajaba en las distribuciones **Backtrack** [10], que eran similares a lo que es Kali Linux hoy en día. Es importante destacar que existen muchas distribuciones que tienen como objetivo ser de utilidad para auditores de seguridad, aunque generalmente tienen funciones más específicas: **Wifislax** [11] para auditorías *WiFi*, **Santoku** [12] para auditorías forense y móviles, etc.

Esta distribución es muy conocida y utilizada por su gran abanico de herramientas y su activa comunidad.

1.5. Metasploit

Metasploit [7] es un *framework*, actualmente mantenido por **Rapid7** [13], utilizado ampliamente en el mundo del *pentesting*. Fue desarrollado originalmente en el lenguaje **Perl**, pero actualmente está escrito en **Ruby**.

Esta herramienta incluye un amplio abanico de *exploits*, divididos por el servicio objetivo del ataque, y permite la integración con otras herramientas como **Nmap** y **Nessus**, exportación de datos en varios formatos e incluso, integrar *exploits* que no se encuentren actualmente en su base de datos.

Los pasos básicos para la explotación de un sistema con **Metasploit** consisten en:

1. Seleccionar y configurar un *exploit*.
2. Seleccionar y configurar un *payload* (código que será ejecutado en el sistema objetivo, como, por ejemplo, una **Shell** remota).

3. Seleccionar una técnica de codificación de con el fin de que un sistema de prevención de intrusos (IPS) ignore el *payload* codificado (opcional).
4. Ejecutar el *exploit*.

La facilidad de uso de **Metasploit**, y el hecho de que está disponible de manera abierta y gratuita, hace que sea frecuente su uso, tanto por auditores de seguridad, como por atacantes reales.

1.6. Formación en Ciberseguridad Ofensiva

Empresas como **Cisco** han ofrecido diferentes certificaciones y cursos en el área [14], pero se suelen enfocar en la parte defensiva de la seguridad, no dándole tanta importancia a la ofensiva. Otras empresas sí ofrecen certificaciones enfocadas en la seguridad ofensiva, como **Offensive Security**, que ofrece varios certificados para expertos en explotación de sistemas, explotación y auditorías *Wi-Fi* y auditorías a aplicaciones web. **EC-Council** [15] ofrece certificaciones en *Hacking* Ético, Investigación Forense, Desarrollo seguro, etc. El inconveniente de estas certificaciones es su elevado precio y la necesidad de tener experiencia profesional demostrable en el sector durante un periodo de tiempo que depende de la certificación.

También han surgido proyectos enfocados en la formación y concienciación en la seguridad ofensiva, como por ejemplo **OWASP** (Open Web Application Security Project) [16], un proyecto especialmente interesado en la seguridad de las aplicaciones web. **OWASP** Ofrece diferentes herramientas para el *testing* de aplicaciones, guías de desarrollo seguro y para la realización de pruebas de penetración, listas de vulnerabilidades comunes, etc.

Uno de los recursos más utilizados en la formación en ciberseguridad es el de entornos creados con la intención de ser vulnerables y atacados de alguna forma. Con esto, es posible probar técnicas de ataque y afianzar conocimientos de manera práctica, sin incurrir en ilegalidades.

Estos entornos pueden incluir vulnerabilidades de diferentes tipos: vulnerabilidades web, desbordamientos de *buffer*, vulnerabilidades de componentes que pueden ser atacados de manera automática con herramientas como **Metasploit**, etc.

OWASP ha desarrollado un conjunto de entornos vulnerables llamado *OWASP Broken Web Applications Project* (**BWA**) [17]. Este proyecto permite descargar una máquina

virtual que incluye varias aplicaciones web con vulnerabilidades conocidas. Una de las aplicaciones incluidas en este proyecto es **WebGoat**, que, en lugar de ser un entorno vulnerable, ofrece un listado de pequeñas pruebas. Estas pruebas permiten entender vulnerabilidades de manera individual, en lugar de tener que enfrentarte a un sistema sin saber por dónde empezar.

En smashthestack.org, overthewire.org y varios sitios más se ofrecen varios entornos llamados *wargames*. Un *wargame* es un entorno en el cual hay varias pruebas, divididas por niveles; cada vez que se supera una prueba, obtenemos acceso al siguiente nivel, donde tendremos que superar otra prueba, y así sucesivamente. Generalmente los *wargame* suelen centrarse en vulnerabilidades debido a errores en programas escritos en el lenguaje de programación C o vulnerabilidades de un sistema UNIX.

1.7. CTF

La evolución de estos entornos vulnerables ha sido lo que se denomina **CTF** (Capture the Flag), concursos en los que, ya sea de manera individual o en equipos, se debe demostrar habilidad tanto práctica como teórica en este campo del conocimiento.

Hay tres tipos comunes de **CTF**:

- **Ataque-Defensa:** hay que atacar y/o defender un sistema con el fin de obtener una bandera (o *flag*) y evitar que los contrincantes consigan la nuestra.
- **Pregunta-Respuesta (*Jeopardy*):** no es necesario vulnerar ni defender sistemas. Generalmente se presentan enunciados que pueden venir acompañados con ficheros ejecutables, fuente, imágenes, etc. Las pruebas de estos **CTF** también se pueden dividir en varios grupos:
 - **Web:** se presenta una URL que dirige a una página web con vulnerabilidades típicas de entornos web (*cookies* inseguras, vulnerabilidades de inyección, etc.)
 - **Explotación:** se entrega un fichero ejecutable y, a veces, el fuente que ha generado ese fichero. Este fichero ejecutable suele contener una función que puede parecer inaccesible a simple vista, pero atacando vulnerabilidades como desbordamientos de *buffer* se puede acceder a la misma.

- **Criptografía:** este tipo de pruebas no suele presentar ficheros extra. Se presenta un enunciado el cual incluye un texto cifrado y se dan pistas acerca de como romper el cifrado utilizado.
 - **Forense:** en este tipo de pruebas se suelen presentar ficheros como *dumps* de memoria, capturas de tráfico o ficheros ejecutables que deben ser analizados para obtener la *flag* en cuestión.
 - **Esteganografía:** la esteganografía consiste en técnicas que permiten ocultar mensajes u objetos, dentro de otros, de modo que no se perciba su existencia. Las pruebas de este tipo suelen presentarle a los participantes imágenes, ficheros de audio, etc.
- **Mixto:** se presentan tanto entornos que simulan redes y equipos, como enunciados a responder.

Varias empresas y organizaciones están empezando a realizar pruebas de este tipo para incentivar el crecimiento del sector y captar talento. Por ejemplo, **Google** ha realizado su primer **CTF** este año (2016). El Instituto Nacional de Ciberseguridad (**INCIBE**)[18] ha organizado un **CTF** de varias etapas este año para el evento Cybercamp. Este evento constaba de una fase individual que se realizaba online y era del tipo **pregunta-respuesta** e incluía pruebas de criptografía, web, ingeniería inversa/forense y explotación, cambiando cada semana la temática de las pruebas. La segunda fase fue presencial en el evento Cybercamp y ofrecía entornos más complicados que simulaban redes empresariales, es decir, era de tipo **mixto**.

Si se quiere participar en uno de estos eventos, hay varias formas de empezar y practicar. En ctftime.org hay un *ranking* con las próximas competiciones de este tipo, un *ranking* global de los equipos que participan en las mismas. También contiene un archivo con un histórico de **CTF** realizados anteriormente. El sitio web root-me.org incluye varios desafíos para resolver y “salas” donde usuarios pueden instanciar máquinas virtuales y realizar pruebas del tipo ataque-defensa; hack.me ofrece un servicio de *hosting* de *sandbox* que permite crear pruebas con contenido estático, aisladas unas de otras, generalmente de tipo pregunta-respuesta enfocado en pruebas de entornos web.

El problema viene cuando, por ejemplo, una organización de alumnos como **IEEE** (o un profesor o una organización externa al ámbito académico) quiere realizar un **CTF**, desarrollado por ellos mismos, o con una recopilación de pruebas. Sus alternativas son:

desarrollar desde cero una plataforma para la organización de pruebas y el ranking de los equipos, utilizar servicios online externos (como los vistos anteriormente) o utilizar otros *frameworks* ya existentes, como CTFD del *ISIS Lab* de la *NYU School of Engineering* [19]. El problema encontrado con este *framework* es la poca libertad disponible para un diseñador de pruebas, ya que sólomente ofrece soporte para **CTF** del tipo pregunta-respuesta.

Ademas de CTFD, existen diversos *framework* para el desarrollo de **CTF** tipo pregunta-respuesta como: FBCTF [20] (Facebook CTF), Mellivora [21], scorebot [22] o NightShade [23]. Los *framework* para **CTF** tipo ataque-defensa son más escasos (entre los que destaca *shuriken-framework*) [24] pero nos fue imposible encontrar un *framework* para **CTF** de tipo mixto.

Capítulo 2

Objetivos

En este capítulo, vamos a describir los principales objetivos de este proyecto.

2.1. Descripción del Problema

Como se describió al final del capítulo anterior, el problema que habíamos encontrado en el entorno de los **CTF** era la necesidad de recurrir a recursos externos o de mantener arrancados servicios y programas a lo largo de la duración del mismo. Además, los entornos eran generalmente compartidos, por lo que un usuario podía realizar una denegación de servicio de una prueba y perjudicar la experiencia del resto de los participantes.

2.2. Objetivos

Sabiendo todo esto, decidimos enumerar los siguientes objetivos para el proyecto:

1. Desarrollar una plataforma electrónica educativa para el desarrollo de las competencias necesarias para comprender, desarrollar y contrarrestar los distintos tipos de ataques a sistemas informáticos y redes a través del empleo de mecánicas de juego (gamificación).
2. Crear un “juego de guerra” (*wargame*) multinivel en el que los estudiantes pueden atacar distintos sistemas aplicando diversas técnicas existentes dentro de un entorno controlado.
3. Aplicar diversas técnicas de gamificación para influir y motivar a los estudiantes en el estudio de la seguridad de los sistemas operativos y los sistemas distribuidos.

4. Crear una plataforma genérica y libre, que esté disponible a través de Internet para su uso en otras organizaciones. Al consistir en niveles, este proyecto será continuado en el tiempo de forma natural, añadiendo nuevos niveles, distintas características de gamificación y potenciando su uso mediante la interacción con distintas redes sociales.

Siendo estos los objetivos iniciales, a lo largo del proyecto, llegaron a aparecer diferentes objetivos, y otros se decidieron dejar en un segundo plano. Por ejemplo:

- Proporcionar facilidad de uso y la uniformidad del formato de las pruebas, convirtiendo el proyecto más en un *framework* o entorno de trabajo, en lugar de una simple plataforma.
- Crear pruebas independientes entre sí, para cubrir más ámbitos dentro de la ciberseguridad (criptografía, metadatos, ataques web, etc.).

Además de todo esto, la plataforma a desarrollar, obviamente, debe ser resistente a ataques y, en la medida de lo posible, no contener vulnerabilidades típicas de entornos web. Entre estas medidas destacan:

- Se utilizan consultas **SQL** parametrizadas para evitar Inyecciones **SQL**.
- El **HTML** servido proviene de plantillas que escapan todo el contenido dinámico, evitando **XSS**.
- No se guarda información en texto claro en las *cookies*.
- No se realizan funciones de administración directamente en la página web, por lo que los ataques **CSRF** no pueden realizar ninguna acción trascendental.

Capítulo 3

Descripción del trabajo desarrollado

3.1. Metodología

El modelo del ciclo de vida usado durante el desarrollo de este proyecto es el de “*Desarrollo en espiral*” [25]. Este modelo nos permite trabajar en el software deseado de manera incremental, aumentando la complejidad del problema de manera progresiva y proporcionando prototipos funcionales al final de cada fase.

El uso de este modelo es muy útil debido a que permite dividir el trabajo a realizar en pequeños sub-objetivos, los cuales se pueden probar y perfeccionar antes de avanzar a otros objetivos en el desarrollo. El modelo en espiral se desarrolla en ciclos y en cada ciclo podemos diferenciar cuatro partes principales. Una vez hemos avanzado a través de estas partes, podemos avanzar hacia la siguiente iteración. En la siguiente imagen podemos distinguir las 4 partes principales de cada ciclo de desarrollo.



Figura 3.1: Desarrollo en espiral

1. **Determinar objetivos.** Encontramos qué objetivos deben ser satisfechos al final de

cada iteración, teniendo en consideración el objetivo final deseado.

2. **Análisis de riesgo.** Estudiamos los procesos que pueden ser seguidos con el fin de cumplir los objetivos determinados y analizamos las ventajas y desventajas de cada uno. Además determinamos los riesgos que se presentarán en este ciclo y proponemos posibles formas de mitigarlos.
3. **Desarrollar y probar.** Una vez decidido el proceso y las herramientas a utilizar, desarrollamos el producto deseado. Una vez el desarrollo se finaliza, procedemos a realizar una serie de pruebas con el fin de asegurar su conformidad con los objetivos que teníamos para esa iteración.
4. **Planificación.** Después de ver los resultados obtenidos en esta iteración, se empieza a planear la siguiente fase, teniendo en cuenta los problemas encontrados en la fase presente, con el fin de evitarlos en las próximas.

Los ciclos que han tomado lugar en este proyecto se describirán con detalle en la próxima sección.

3.2. Iteraciones

Las iteraciones que se describirán a continuación, fueron generalmente ideadas una a continuación de otra, por lo que no se siguió un plan específico a la hora de escogerlas.

- **Desarrollo de una aplicación web en Go.** El objetivo principal de esta fase inicial era la familiarización con el desarrollo web en el lenguaje de programación Go, el cual se basa principalmente en la librería `net/http`. Esto serviría como base para la plataforma.
- **Desarrollo de un “juego de guerra” (*wargame*).** En esta fase se creó basado en UNIX, en una máquina virtual, el cual incluía varios niveles, con una prueba por nivel, con el objetivo de demostrar errores y vulnerabilidades típicos en entornos UNIX y programas escritos en el lenguaje C.
- **Mejora de la aplicación web, incluyendo en ella el juego de guerra realizado anteriormente.** Se implementaron funciones que permitiesen el encendido y apagado dinámico del wargame, de forma que un usuario pudiese encender la máquina

virtual cuando quisiese utilizarla. De esta manera, no era necesario tenerla encendida permanentemente para dar servicio

- **Implementación de nuevas pruebas.** Decidimos implementar nuevas pruebas de diferentes tipos con dos fines. El primero, empezar a poblar la plataforma con pruebas que se enfocasen en diferentes ataques y fallos; el segundo, entender mejor como desarrollar pruebas, con el fin de que el *framework* pudiese trabajar con el mayor número de pruebas diferentes posible.
- **Mejora de interfaz de usuario.** Al llegar a este punto, decidimos mejorar el aspecto visual del sitio web generado por la plataforma.
- **Implementación de aún más pruebas.** De nuevo decidimos implementar más pruebas. En esta etapa las pruebas eran muy diferentes entre sí y nos permitieron ver algunas deficiencias en el *framework*, las cuales pudieron ser solventadas. Algunas pruebas que se desarrollaron en esta etapa estuvieron más enfocadas a servir de prueba de concepto, implementando pequeñas funciones que podrían ser implementadas en pruebas más interesantes, como por ejemplo, la utilización de los *bots* de Telegram para interacción con un usuario.
- **Implementación de funciones de administración.** En último lugar, se implementaron ciertas acciones de administración a la plataforma, con el fin de poder exportar o importar pruebas, así como eliminar pruebas que se encontrasen actualmente en el entorno.

3.3. Infraestructura

A continuación, se describirán los diferentes elementos externos que se han utilizado en este proyecto, como lenguajes de programación, principales librerías utilizadas y *softwares*.

3.3.1. Go

Go [26] es un lenguaje de programación desarrollado por Google, de código abierto que hace sencillo construir software simple, confiable y eficiente. Ha sido diseñado para aplicaciones *cloud* escalables.

Go consigue su simplicidad y confiabilidad basándose en, entre otras cosas, la legibilidad de su código. Generalmente no hay muchas formas de realizar una acción en Go, por lo que



Figura 3.2: Tiempos de *benchmark* relativos a C, mientras más pequeño mejor (la eficiencia de C = 1.0).

entender un fragmento de código es más sencillo. Al entender mejor un fragmento de código, es más sencillo trabajar con él y, si es necesario, arreglarlo.

Además, Go incluye diferentes herramientas de concurrencia, una de ellas son las llamadas “gorutinas”. Las *gorutinas* son corutinas colaborativas, las cuales son ejecutadas por diferentes hilos, planificados por colas de planificación. A pesar de esto, las *gorutinas* son muy ligeras y eficientes.

3.3.2. net/http

A la hora de programar un servidor web, lo recomendable en Go es utilizar esta librería, ya que aun habiendo diferentes *frameworks* para el desarrollo web (como `Gin` [27], `Revel` [28] o `Beego` [29]), lo recomendable y lo que cumple mejor la filosofía de Go, es utilizar su librería estándar.

El paquete `net/http` [30] implementa un eficiente servidor HTTP y su uso es muy sencillo: llamando a la función `ListenAndServe` podemos servir HTTP en la dirección y puerto que proporcionemos. `ListenAndServeTLS` nos permite servir HTTPS, siempre y cuando

proporcionemos un certificado y una clave privada. La función `HandleFunc` nos permite enlazar funciones a patrones en la petición, por ejemplo:

```
http.HandleFunc("/", handlerRoot)
http.ListenAndServe(":9090", nil)
```

En este ejemplo, el programa aceptará peticiones en la dirección `localhost` y el puerto 9090 y, las que pidan el recurso `"/"`, serán procesadas por la función `handlerRoot`.

Es importante destacar que la función `http.ListenAndServe` (y `http.ListenAndServeTLS`) reciben como último parámetro un *Handler*, permitiéndonos implementar uno, en caso de que el *Handler* básico de `net/http` no nos baste. En el caso de este proyecto, se decidió utilizar el paquete github.com/gorilla/mux, el cual daba pequeñas facilidades, muy útiles a la hora del desarrollo de la plataforma. Para el uso de este *Handler*, hay que cambiar muy poco en el código del ejemplo anterior:

```
import github.com/gorilla/mux
...
r := mux.NewRouter()
r.HandleFunc("/", handlerRoot)
http.ListenAndServe(":9090", r)
```

Otro paquete que se ha utilizado, además de github.com/gorilla/mux es github.com/gorilla/sessions, el cual implementa un sistema de sesiones con *cookies*, guardando información en ellas (como nombre de usuario) y posteriormente cifrándolas, con el fin de evitar su lectura y modificación por usuarios o atacantes. Este paquete permite el uso de varios pares de claves para su rotación. La primera clave de cada par se utiliza para autenticación y la segunda para cifrar. Las claves de cifrado deben ser de 16, 24 o 32 bytes para escoger entre AES-128, AES-192 o AES-256.

Todas las funciones que se utilicen como manejadores en `net/http` deben aceptar dos argumentos:

1. `http.ResponseWriter`. Es el canal de comunicación para responder al cliente. Con esta estructura de datos podemos enviar cabeceras HTTP, redirecciones, etc.
2. `*http.Request`. Un puntero a un `http.Request`, en el cual podemos encontrar toda la información referente a la petición realizada (por ejemplo *cookies*, cabeceras o el método HTTP recibido)

3.3.3. html/template

Este paquete [31] implementa plantillas basadas en datos para la generación de HTML seguro contra inyección de código. Más adelante se hablará más acerca de la inyección de código.

Las plantillas se ejecutan aplicándolas a una estructura de datos, permitiendo generar diferentes salidas de acuerdo a los datos dentro de la estructura. Dentro de una plantilla se pueden realizar operaciones como definición de bloques, e inclusión de otras plantillas, permiten el uso de variables en ella, incluir estructuras de selección, iteraciones, etc.

Esto permite, por ejemplo, generar diferentes páginas HTML que dependan de una lista de elementos en una base de datos. Podemos obtener un *array* de la base de datos e iterar sobre él dentro de la plantilla, en lugar de tener que generar el HTML directamente dentro del código del programa.

3.3.4. MySQL

Para almacenar los datos que deben perdurar entre ejecuciones decidimos utilizar una base de datos y optamos por MySQL.

MySQL es uno de los sistemas gestores de base de datos a larga escala mas populares. Ofrece múltiples funcionalidades y es código abierto. Además, empezar a utilizar MySQL es relativamente sencillo y es trivial encontrar información al respecto. Esta puede trabajar con muchos datos de manera eficiente.

Si bien una base de datos PostgreSQL es más potente que una MySQL, nos pareció que la pérdida de velocidad y simpleza a la hora de configurar la base de datos era suficiente motivo para escoger MySQL. En cambio, una base de datos SQLite no nos proporcionaba la robustez a la hora de trabajar concurrentemente como una MySQL. Para la interacción con la base de datos desde el programa, se utilizó la librería github.com/go-sql-driver/mysql. Su uso es muy sencillo:

```
import "database/sql"
import _ "github.com/go-sql-driver/mysql"

db, err := sql.Open("mysql", "user:password@/dbname")
```

Con esto, tenemos en la variable `db` una instancia `sql` con la cual podemos realizar consultas a la base de datos. Además, se permite el uso de peticiones parametrizadas, para evitar la inyección SQL. Más adelante se explicarán en detalle los esquemas utilizados.

3.3.5. Vagrant

Vagrant [32] es una herramienta para la construcción de entornos de desarrollo, los cuales son fáciles de configurar, reproducir y exportar. Vagrant trabaja encima de proveedores como VirtualBox(el usado en este proyecto), VMWare[33], AWS[34]. Para la configuración e instalación automática se utilizan **scripts** de **Shell**, **Chef**[35] o **Puppet**[36].

Esta herramienta nos permite crear entornos virtualizados fáciles de exportar, con el fin de poder desarrollar pruebas y luego exportarlas sin preocuparnos por las dependencias.

En este proyecto se utiliza **Vagrant** para dos pruebas. Una es de confección propia, mientras que otra es una versión de la famosa máquina virtual **Metasploitable 2** [37].

3.3.6. Docker

Los contenedores de **Docker** [38] envuelven software en un sistema de ficheros que contiene todo lo que necesita para ejecutarse: código, herramientas del sistema, librerías, etc. Esto garantiza que siempre se ejecutará de la misma forma, sin importar el entorno en el cual esté ejecutándose. Todos los contenedores que corren en una máquina comparten el *kernel* del sistema, haciéndolos más eficientes. Las imágenes de **Docker** construyen un sistema de ficheros en entorno de usuario, haciendo que compartir ficheros, utilizar el disco y descargar imágenes sea mucho más eficiente respecto a máquinas virtuales convencionales. La ventaja principal es el tiempo que se tarda en instanciar: un contenedor tarda decenas de milisegundos en arrancar y una **VM** tarda decenas de segundos. Lo mismo pasa para pararlos. Esta es la principal ventaja de los contenedores [39].

Estos contenedores se basan en estándares abiertos, permitiendo que se puedan ejecutar en todas las grandes distribuciones de **Linux** y **Microsoft** con soporte para cada infraestructura. Además, los contenedores aíslan aplicaciones entre sí y la infraestructura inferior, añadiendo una capa de seguridad al entorno.

Una de las ventajas que vimos en **Docker** fue la capacidad de lanzar dinámicamente pruebas en las que los usuarios no comparten entorno, de esta forma podemos buscar objetivos para pruebas que generalmente no son posibles, como por ejemplo denegaciones de servicio o *defacement* de páginas web.

El *defacement* consiste en la deformación o cambio producido en una página web por un atacante. Puede consistir en reemplazar imágenes, textos, redirigir al sitio web del atacante, etc.

Start and stop times

	Start Time	Stop Time
Docker Containers	<50ms	<50ms
VMs	30-45 seconds	5-10 seconds

Flux7

Figura 3.3: Comparación entre tiempos de arranque y parada de máquinas virtuales y contenedores

3.4. Desarrollo de la plataforma

Para el desarrollo, como se explicó anteriormente, se ha utilizado el lenguaje de programación Go. La plataforma ha recibido el nombre **CTFF** (*Capture The Flag Framework*). Esta consta de varios ficheros fuente, que forman parte del mismo paquete “main”. Este se puede ejecutar en varios modos:

- **run:** lanza el servidor web con la configuración actual.
- **setup:** busca cambios en el directorio de desafíos y añade las pruebas nuevas.
- **list:** imprime el alias de cada una de las pruebas actualmente instaladas en el servidor.
- **export ALIAS...:** genera n ficheros **.ctff**, con los contenidos de cada una de las pruebas que se haya pedido exportar, por separado, los cuales pueden ser importados posteriormente. El fichero **.ctff** se explica en el apartado **Fichero .ctff**.
- **install CTFF_FILE...:** recibe n ficheros **.ctff**, los cuales extrae y utiliza para instalar en la plataforma.
- **remove ALIAS...:** recibe n alias de pruebas instaladas en la plataforma y elimina tanto sus ficheros como sus entradas en la base de datos.

Posteriormente se darán mas detalles acerca del uso de estos comandos.

3.4.1. Código Fuente

La plataforma está formada por los siguientes ficheros fuente.

main.go

El código en `main.go` se encarga de parsear los argumentos recibidos, comprobando que se ha introducido el número de argumentos necesario dependiendo del modo de operación elegido. Además, se registran aquí los manejadores de los recursos `HTML` del servidor. Debido a su complejidad, este código también se encarga de listar los alias de las pruebas instaladas en la plataforma, en caso de que ese sea el modo de operación escogido.

remove_challenge.go

Este código se encarga de eliminar todas las pruebas que se indiquen, de manera concurrente. Para eliminar las pruebas, primero se eliminan de la base de datos y, posteriormente, se eliminan sus ficheros.

add_challenge.go

En este código se realizan las funciones encargadas de añadir pruebas a la plataforma. Primero lee los datos de la prueba, hallados en su fichero `info.json` (es un fichero en el cual se codifican los datos necesarios para almacenar una prueba en la base de datos, más adelante se explicará en que consisten los campos de dicho fichero) a continuación, se añade esta prueba a la base de datos. Además, incluye la funcionalidad necesaria para extraer los ficheros `.ctff`.

export_challenge.go

El código en `export_challenge.go` genera ficheros `.ctff` a partir de las pruebas de manera concurrente.

config.go

En `config.go` se encuentra un conjunto de constantes, las cuales deberán ser configuradas por el administrador de la plataforma, antes de su uso. Actualmente, dichas constantes son:

- **CTF2Path:** *path* absoluto del directorio donde se encuentre el ejecutable de la plataforma.
- **ChallengesPath:** *path* absoluto del directorio donde se encuentren las pruebas.
- **MaxChallengeScore:** Actualmente en desuso, en un futuro puede ser el valor máximo de puntuación para una prueba.
- **DBLoginString:** String con el que se espera acceder a la base de datos, siguiendo el siguiente formato:

```
"[username[:password]@][protocol[(address)]]/dbname[?param1=value1&...&paramN=valueN]"
```

Posteriormente se dará más información acerca del uso básico para un administrador de la plataforma.

handlers.go

En este fichero se encuentran las funciones a realizar, dependiendo de que recurso HTML sea utilizado por cada usuario. Como se dijo anteriormente, estos handler tienen el formato `func(http.ResponseWriter, *http.Request))` y utilizan una librería externa para el uso de *cookies* y codificación de información cifrada dentro de las mismas.

En caso de que se quisiese añadir un manejador, es necesario implementar una función que se encargue de manejar la petición y añadir dicho manejador a las rutas en el fichero `main.go`.

dboperations.go

Tanto en este fichero fuente como en `handlers.go`, se encuentra el núcleo principal de la funcionalidad del entorno. En este fuente se encuentran los modelos de los tipos de datos utilizados en la base de datos y funciones para introducir o extraer estos datos en la base de datos.

Además, las estructuras de la base de datos se corresponden a una estructura en Go. Estas estructuras poseen varios métodos, utilizados en el resto de funciones de la plataforma. A continuación, describirán con detalle los modelos de la base de datos, y los métodos más importantes de las estructuras, como por ejemplo, cómo se lanzan las pruebas.

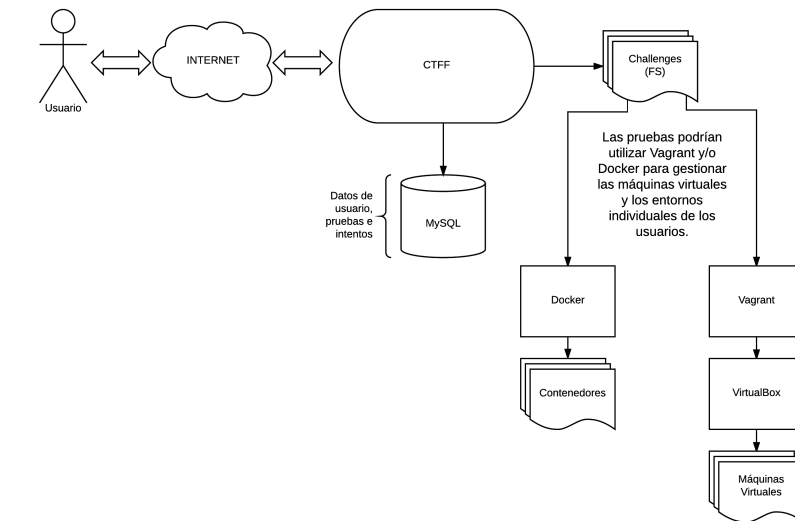


Figura 3.4: Arquitectura de la plataforma.

3.4.2. Esquemas de base de Datos

La base de datos necesita tener tres tablas: **challenges**, **attempts** y **userinfo**. La primera mantiene información acerca de las pruebas, la segunda acerca de los intentos de solución y, la tercera, sobre los usuarios.

Tabla challenges

Esta tabla contiene la información acerca de las pruebas de la plataforma. Una parte se genera a la hora de instalar la prueba mientras que la otra se va modificando según se van instanciando pruebas. Sus columnas son las siguientes.

- **UID:** clave primaria. Identificador único de una prueba, se genera realizando un hash y seleccionando los primeros 8 caracteres de su representación hexadecimal. Siendo cada uno de esos caracteres algo entre 0 y f y escogiendo 16 los primeros 8 caracteres tenemos $16^8 = 4294967296$ diferentes combinaciones de identificadores. Según el problema del cumpleaños [40], necesitaríamos 78643 pruebas diferentes en un mismo entorno para que la probabilidad de colisión de identificadores fuese mayor que $\frac{1}{2}$.
- **Title:** el título de la prueba, que debe ser diferente de **NULL**. Es una cadena de caracteres con una longitud máxima de 140 caracteres. Se carga a la hora de instalar la prueba.

- **Description:** la descripción de la prueba. Este texto no puede ser `NULL` y es una cadena de caracteres con una longitud máxima de 500 caracteres. Se carga a la hora de instalar la prueba. Esta descripción no debería ser muy detallada (un pequeño texto con el cual informar brevemente al usuario que tipo de prueba se va a realizar).
- **MaxScore:** un entero que no puede ser `NULL`, su función actual es indicar cuál es la puntuación que otorga la prueba al finalizarla. Se carga a la hora de instalar la prueba. En un futuro podría utilizarse para, como su nombre indica, indicar el valor máximo posible de la prueba, pudiendo reducir los puntos resultantes para el usuario en función de varios factores.
- **Alias:** una cadena de caracteres con una longitud máxima de 100 caracteres. Se utiliza para funciones de administración de la plataforma. Se genera a partir del nombre del directorio que contiene a la prueba, tomando como base el directorio donde se encuentren todas las pruebas.
- **Category:** cadena de caracteres, que no puede ser `NULL`. Sirve para discriminar las pruebas por su tipo. Se carga a la hora de instalar la prueba.
- **Creator:** cadena de caracteres que no puede ser `NULL`. Tampoco se utiliza para nada actualmente pero podría utilizarse para enseñar quien ha diseñado la prueba para, por ejemplo, contactar con él para solicitar información acerca de la prueba o informar acerca de *bugs*, etc. También se podrían implementar sistemas de premios o algo similar, con técnicas de gamificación, para fomentar la creación de pruebas.

Tabla attempts

Esta tabla contiene la información acerca de los intentos de solución de cada una de las pruebas y todos sus datos se generan de forma dinámica a lo largo del uso de la plataforma. Sus columnas son las siguientes.

- **AttemptId:** un entero único por intento que se incrementa de manera automática, es utilizado como clave primaria. Este identificador podría utilizarse para eliminar intentos específicos en la base de datos.
- **Date:** un campo de fecha, en el cual se registra la fecha en la que se realizó el intento.

- **U_email:** cadena de caracteres utilizada para enlazar un intento con el usuario que lo realizó. Cabe destacar que este campo es una clave externa enlazado con un usuario en la tabla `userinfo`.
- **Successful:** un valor booleano que indica si dicho intento ha sido exitoso. No debería haber nunca más de un intento exitoso por usuario y prueba, es decir, una vez solucionada la prueba, el usuario no podrá volver a realizarla.
- **Score:** puntuación que se obtendría si dicho intento fuese exitoso. Este campo actualmente siempre es igual a la puntuación máxima, pero, como se dijo anteriormente, podría modificarse dependiendo de diversos factores. La puntuación obtenida al superar la prueba se obtiene a partir de este campo y no a partir del campo `MaxScore` de la tabla `challenges`.
- **UID:** identificador único de la prueba en la cual se ha realizado el intento. Esto podría utilizarse en un futuro, entre otras cosas, para enseñar que usuarios han realizado una prueba específica.

Tabla `userinfo`

Esta tabla contiene la información acerca de los usuarios registrados en la plataforma, como por ejemplo correo electrónico y puntuación.

- **Email:** clave Primaria. Correo electrónico del usuario, proporcionado al momento de registro, es posible utilizarlo también para iniciar sesión. En el futuro se podría utilizar para enviar notificaciones a los usuarios por correo electrónico.
- **Password:** cadena de caracteres, es una `hash SHA-512` de la contraseña, de forma que no se guardan en claro en ningún momento. En un futuro podrían utilizarse funciones criptográficas más seguras para almacenarlas como KDF[41].
- **Created:** Fecha en la que se creó la cuenta del usuario, actualmente no tiene ningún uso real pero se podrían implementar diversas funcionalidades que utilizarasen este campo como, por ejemplo, permitir subir pruebas a la plataforma si el usuario tiene un cierto tiempo registrado en la plataforma.
- **Username:** Nombre de usuario, es una cadena de caracteres con una longitud máxima de 64 caracteres. Al igual que el correo electrónico, se tiene que proporcionar

a la hora del registro y es posible utilizarlo a la hora de iniciar sesión. Se utiliza junto al correo electrónico debido a que se utiliza para servir la página de usuario y un usuario puede preferir identificarse con un **username** en lugar de un correo electrónico (más información acerca de la página de usuario en la sección **Producto Final**).

- **Score:** Puntuación actual del usuario. Se actualiza según el usuario va superando pruebas.

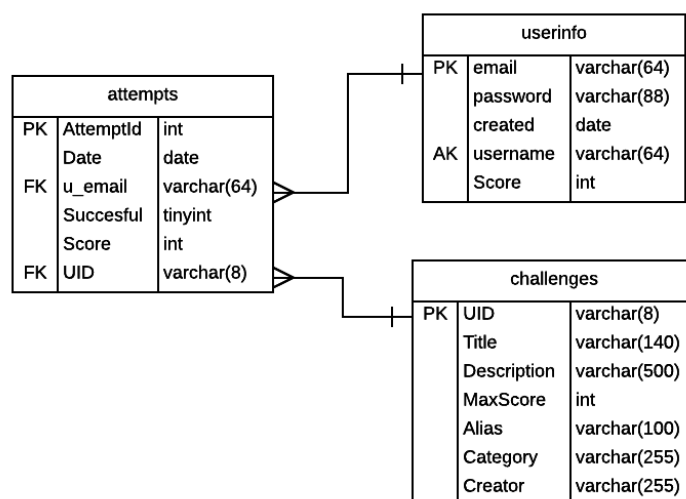


Figura 3.5: Diagrama Entidad-Relación de las tablas de la base de datos.
Notación *Crow's Foot* [42]

3.5. Desarrollo de Retos

A la hora de empezar con el desarrollo de la plataforma, realizamos una taxonomía de los tipos de pruebas con el fin de permitir la incorporación de pruebas totalmente diferentes entre si. Dividimos los tipos de prueba de la siguiente forma:

- **Pruebas de enunciado.** Este tipo de pruebas son las que aparecen en **CTF** de tipo pregunta-respuesta. Pueden incluir enlaces para descargar ficheros o imagenes, por ejemplo. Pueden además requerir lanzar un demonio o algo similar.
- **Pruebas de entorno compartido.** Este tipo de pruebas es similar a las que aparecen en **CTF** de tipo ataque-defensa o a los *wargame* convencionales. En este tipo de pruebas, los participantes comparten el entorno. Se utiliza **Vagrant** para la gestión de las máquinas virtuales. La ventaja de **Vagrant** radica en su facilidad

de uso. Para arrancar una máquina virtual únicamente hay que ir al directorio que contiene la prueba y ejecutar el comando `vagrant up` y, para apagar la máquina `vagrant halt`.

- **Pruebas de entorno individual.** Este último tipo consiste en entornos en los cuales es imprescindible que un usuario esté aislado del resto, debido a la naturaleza de la prueba. Por ejemplo: un escenario en el cual hay que realizar una denegación de servicio a un servidor o un *defacement*. Para generar entornos individuales, se recomienda el uso de **Docker**.

A continuación, se procederá a detallar el desarrollo de los retos, su contenido y los pasos estimados para su finalización. Ya que la mayoría de los títulos suelen ser pequeños chistes o referencias de cultura general, es importante no esperar mucha seriedad de los mismos. Este tipo de bromas tiene como finalidad hacer más atractivas a las pruebas y es típico en los entornos **CTF** y del colectivo *hacker* en general.

3.5.1. Prueba de enunciado: Le xorffre indéchiffrable

En esta prueba, se le presenta al usuario un texto y únicamente se le pide que lo descifre. Un fragmento del texto en cuestión es el siguiente:

```
FF73B1D132AF51326191D77EFFDD73A850325A90DD73F4C1738E5F6256DEE870F  
CD33DA21E7740DECF71B1DB3EBD5160479FD46BF49230AC5C7D139ADF3FFDD  
373AE5161479F9A7BF4DE738A517E55919A7BF4921EA...
```

Figura 3.6: Fragmento del texto cifrado

Este texto es un texto cifrado, escrito en su representación hexadecimal en el formato 02X. Este texto ha sido alterado con un cifrado *Vernam* (llamado cifrado **XOR** de ahora en adelante) utilizando una clave desconocida. Es importante tener en cuenta que se considera que el cifrado **XOR** es irrompible, incluso en teoría, siempre y cuando la clave sea totalmente aleatoria y sea de un tamaño mayor o igual al tamaño del mensaje. Si la clave se repite, es posible realizar ataques al cifrado.

El ataque utilizado por nosotros a la hora de idear que se espera para poder superar la prueba consistía en dos fases: la primera consiste en conseguir la longitud de la clave, utilizando el método *Kasiski*[43] y la segunda, obtener el texto en claro, ya sea a través de métodos estadísticos, o por lista blanca de caracteres (los caracteres **ASCII** imprimibles

empiezan en 0x20 y acaban en 0x7E, además se puede rechazar caracteres extraños como llaves, corchetes, etc).

Esta prueba es relativamente complicada, ya que suele requerir que un usuario escriba un programa para descifrar el código, por ejemplo, en el lenguaje C. Una solución de ejemplo, desarrollada en C se encuentra en el repositorio https://github.com/jenarvaezg/vigenere_breaker

3.5.2. Prueba de enunciado: Para el cesar lo que es del cesar

Esta prueba, al igual que la anterior, presenta al usuario un texto cifrado que tiene que descifrar. En este caso, el texto en cuestión es:

S dsk uzausk vw nwjvsv dwk ymkls wd hgddg xjalg

Figura 3.7: Fragmento del texto cifrado

Este desafío es mucho más sencillo que el anterior, ya que para descifrar el texto únicamente hay que realizar un cifrado cesar del texto, probando con todos los desplazamientos hasta conseguir el texto en claro. La pista clave para poder saber que hay que usar un cifrado César está en el título.

Este desafío está pensado para que los usuarios se familiaricen con el uso de la plataforma y con el funcionamiento de las pruebas, ya que la resolución de este desafío es trivial.

3.5.3. Prueba de enunciado: Such challenge

En esta prueba, se le presenta al usuario con el siguiente supuesto escenario: *“En la habitación del sospechoso hemos conseguido un post-it en el cual aparecía el mensaje ‘ohacvezs’ y la siguiente imagen: jenlacej”*. La imagen en cuestión es la siguiente.

Si el usuario intenta descifrar con César el mensaje, no obtendrá nada con sentido, pero, si realiza un descifrado *Vigenere* (similar al cifrado XOR), utilizando como clave el texto que aparece en la imagen (“WOW”), obtendrá el mensaje “steghide”.

Steghide [44] es una herramienta para esconder o extraer textos de imágenes, utilizando esteganografía. Con esta pista, un usuario puede intentar extraer la información oculta en esa imagen. En este caso, “steghide” es la clave para decodificar la información de la imagen.

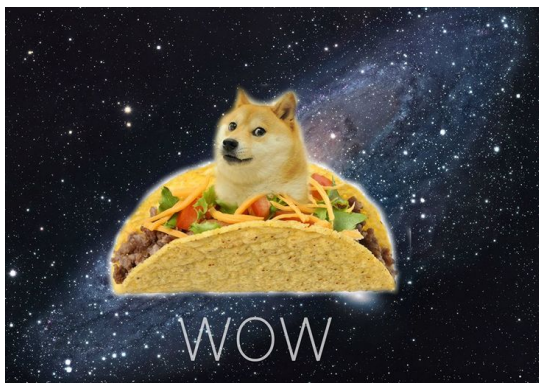


Figura 3.8: Imagen presentada disponible en el enlace proporcionado por la prueba.

Ésta fue una de las primeras pruebas desarrolladas y, permite demostrar el uso de imágenes estáticas servidas por la plataforma, más adelante se dará información en detalle acerca de cómo utilizar este tipo de contenido.

Aunque esta prueba puede parecer sencilla, es necesario que el participante tenga algunos conocimientos de esquemas de cifrado clásico y la existencia de la esteganografía, además, requiere varios pasos para conseguir la *flag*.

3.5.4. Prueba de entorno compartido: Wargame1, level01

Este es el primer nivel de cuatro del *wargame* realizado en este proyecto. En este apartado se aprovechará para explicar la arquitectura del entorno y las herramientas utilizadas para su desarrollo y configuración, posteriormente se describirá el desarrollo y solución de esta prueba en cuestión. En el resto de niveles de este *wargame*, se explicará únicamente el desarrollo y solución de la misma.

El *wargame* está basado en un entorno mínimo UNIX, con sesión sólo-texto. El objetivo de todos los niveles es conseguir atacar una vulnerabilidad para obtener la *flag* del siguiente nivel. Se espera que el usuario entre por `ssh` en el nivel en cuestión utilizando la *flag* obtenida en el nivel anterior como bandera.

Los niveles del *wargame* están separados por usuarios y directorios. En el nivel 01 entramos al sistema como el usuario “level00” y tenemos que conseguir la clave del usuario “level01”, en el nivel 02 entraríamos como usuario “level01” y tendríamos que obtener la clave del usuario “level02” y así sucesivamente.

En el primer nivel, como se explicó anteriormente, el usuario tiene que entrar al entorno por `ssh` con el usuario level00 y la contraseña “elNivelCeroEsEz”. En el home del usuario

`level100` encontramos un fichero `README` en el que se le explica al usuario la dinámica de las pruebas en el sistema y que puede utilizar el directorio `/tmp`, pero no podrá listar sus directorios. También es importante tener en cuenta que nadie tiene permiso de escritura en ninguno de los directorios `home`.

En este nivel, el usuario no tiene ningún programa que ejecutar, en el resto, suele aparecer un ejecutable perteneciente a participante usuario del siguiente nivel, con el bit `suid` activado, de esta forma, al ejecutarse, equivale a que lo ejecute el usuario dueño del fichero.

Para obtener la clave del usuario `level101`, el participante debe explorar el sistema de ficheros del entorno y descubrir que el usuario `level101` tiene algo particular en los permisos de su `$HOME`: tiene permiso de lectura para todos los usuarios. Este detalle permitiría listar todos los ficheros del `$HOME` de `level101` y encontrar el fichero oculto “`.password`”, el cual contiene la contraseña del usuario y, a su vez, sirve de *flag* para obtener los puntos de la prueba.

La premisa de esta prueba es bastante sencilla, pero un usuario necesita ciertos conocimientos de `UNIX` (bastante básicos) para poder superarla. Además, enseña que es muy importante tener vigilados los permisos de nuestros ficheros.

3.5.5. Prueba de entorno compartido: Wargame1, level02

En esta prueba, al conectarse el participante, se encontrará en un directorio con dos ficheros: `level101`, un ejecutable y `level101.c`, un código fuente, el cual se puede asumir que genera al ejecutable `level101` una vez compilado. Lo relevante del código sería lo siguiente:

```
void you_win(void){
    printf("Success!\n");
    seteuid(1003);
    execl("/bin/cat", "cat", "/home/level02/.password", NULL);
}

int main(int argc, char *argv[]){
    if(argc != 2){usage();}
    int pincode = strtol(argv[1], NULL, 10);
    printf("Checking if pincode is correct\n");
    sleep(3);
    if(pincode == SOMETHING){
        you_win();
        return 0;
    }
    printf("Nope\n");
}
```

```
    return 0;
}
```

En el código se ve que hay una función llamada `you_win`, la cual nos escribirá por la pantalla la contraseña del usuario del siguiente nivel. Para que se llame a esta función, es necesario que se le pase un argumento numérico al programa al ejecutarlo y que este sea igual a algo que no se nos indica. Además, antes de realizar la comprobación, el programa se dormirá durante tres segundos, por lo que la fuerza bruta podría tomar demasiado tiempo.

La forma que se utilizó para solucionarla, durante el desarrollo de la prueba, fue utilizar `gdb` [45] para analizar las instrucciones del ejecutable. Tras estudiar dichas instrucciones, hay una en la que realiza una comparación:

```
0x080485bf <+80>: cmpl  $0x1953,0x1c( %esp)
```

Si convertimos el número `0x1953` a decimal obtendremos 6483, el número que nos dará la clave para acceder al siguiente nivel.

La resolución de esta prueba es bastante sencilla si se cuenta con los conocimientos adecuados, pero sin conocer la existencia de herramientas de depuración como `gdb` puede ser imposible, por eso, en el enunciado de la prueba, se recomienda el uso de `gdb`, pero no se dan más pistas. Es posible intentar métodos alternativos como por ejemplo realizar ataques de fuerza bruta para intentar conseguir la *flag*.

3.5.6. Prueba de entorno compartido: Wargame1, level03

Al igual que en la prueba anterior, al participante se le presenta un fichero ejecutable y un código fuente, llamados `level02` y `level02.c` respectivamente. El código fuente es bastante corto:

```
int main(int argc, char *argv[]){
    char *args[] = {"whoami", NULL};
    seteuid(1004);
    execvp("whoami", args);
    return 0;
}
```

Al ejecutar el programa, se ejecuta el programa `whoami`, el cual nos dice el nombre del usuario en cuestión, pero el ejecutable `level02` en lugar de decirnos que somos el usuario “`level02`” no dice que somos el usuario “`level03`”.

Para solucionar esta prueba, es clave saber que la función `execvp` de `C` busca un ejecutable llamado “whoami” en los directorios descritos en la variable `$PATH`. Esta variable, para el usuario `level02` contiene: `"/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games"`

Teniendo en cuenta esto, hay que realizar dos acciones: primero, tenemos que crear un ejecutable que, si lo ejecutase el usuario `level03`, nos diría cuál es la contraseña que se encuentra en el fichero `.password`, dentro de su `$HOME`; el segundo, es modificar la variable `$PATH` para que incluya el directorio donde se encuentre este ejecutable al principio. El usuario puede utilizar el directorio `/tmp` para alojar este directorio y ejecutable. Una vez se hayan realizado estas dos acciones, al ejecutar el fichero `level02`, obtendremos la clave para el siguiente nivel.

Esta prueba requiere conocimientos generales de `UNIX` y `C`, aunque en caso de no conocer específicamente cómo funciona la función `execvp`, se puede leer su entrada del manual. Además, enseña de los peligros de utilizar la variable `$PATH` para encontrar ejecutables, en lugar de utilizar directamente la dirección del ejecutable deseado.

3.5.7. Prueba de entorno compartido: Wargame1, level04

En la última prueba de este *wargame*, el participante se encuentra de nuevo con un código fuente y un ejecutable. El código fuente es el siguiente:

```
int main(void){

    char buff [15];
    int pass = 0;

    printf("\n Enter the password : \n");
    gets(buff);

    if(strcmp(buff, SOMETHING)){
        printf ("\n Wrong Password \n");
    }else{
        printf ("\n Correct Password \n");
        pass = 1;
    }

    if(pass){
        seteuid(1005);
        execl("/bin/cat", "cat", "/home/level04/.password", NULL);
    }

    return 0;
```

}

A priori parece que el objetivo de esta prueba es, al igual que en nivel dos, comprobar con `gdb` cuál es la contraseña que utiliza el programa para comparar con la entrada suministrada, pero esa sería una solución más compleja que la esperada.

El problema de este programa, es que utiliza la función `gets` para leer de la entrada. La entrada de manual de `gets` [46] dice lo siguiente: *"gets() reads a line from stdin into the buffer pointed to by s until either a terminating newline or EOF, which it replaces with a null byte ('\0'). No check for buffer overrun is performed."*. Es decir, lee una línea de la entrada estándar y la introduce en la dirección que se le diga. El problema con esta función, es que no realiza comprobación de tamaño a la hora de introducir lo leído en el destino. Si un usuario introduce más bytes como entrada de los esperados, sobrescribiría segmentos de memoria que no debería.

En este caso, si un usuario introduce más de 15 caracteres por la entrada del programa, la función `gets` modificará el valor de la variable `pass` por lo que, al comprobarse posteriormente si es diferente de cero, obtendremos la clave del usuario `level104`.

En esta prueba se demuestra la importancia del tamaño de un *array* o *buffer* y el peligro de desbordamiento, ya que podríamos permitir comportamientos anómalos en nuestro programa.

Aunque esta es la última prueba de este *wargame*, es posible añadir más pruebas dentro de esa máquina virtual. Ningún usuario de los utilizados en las pruebas tiene permisos de superusuario, pero `Vagrant` nos permite utilizar un usuario, llamado `"vagrant"`, para elevar privilegios. Una vez con privilegios elevados, habría que crear un nuevo usuario, y una prueba en la que se permita obtener la contraseña del siguiente nivel.

3.5.8. Prueba de enunciado: Fear The EXIF

En esta prueba se le presenta al usuario únicamente con una imagen y se le dice que hay un mensaje oculto en la misma. Este mensaje será la solución de esta prueba. La imagen en cuestión es la siguiente.



Figura 3.9: Imagen presentada disponible en el enlace proporcionado por la prueba.

El título de la prueba está pensado para que el participante no piense que tiene que utilizar técnicas de esteganografía para encontrar la solución, como en pruebas anteriores, sino que tiene que buscar información en los metadatos de la imagen. El título, además, hace un guiño a un popular programa para el análisis de metadatos, llamado FOCA [47], cuyo eslogan es: “Fear the FOCA”

Los metadatos pueden definirse como datos que contienen información relativa a un documento o fichero concreto. Por ejemplo, un archivo de texto podría contener entre sus metadatos multitud de información relacionada con su procedencia, datos sobre su autor, su fecha de creación y modificación, qué otros usuarios han manipulado el documento o el software utilizado para su redacción, por ejemplo. Una fotografía puede incluir información en sus metadatos acerca de la marca y modelo de la cámara utilizada, profundidad de color, resolución o, incluso, coordenadas GPS desde las que se realizó dicha fotografía.

No obstante, los metadatos podrían convertirse en un riesgo potencial para el creador de la información si, al distribuir o publicar documentos en Internet, no son gestionados de forma adecuada. Para solucionar esta prueba, como se ha explicado anteriormente, hay que buscar información oculta entre los metadatos de la imagen. Específicamente, en el campo `\ImageDescription` del EXIF de la imagen, aparece la flag en cuestión.

3.5.9. Prueba de enunciado: La Venganza De Atahualpa

En esta prueba se le da al usuario un archivo de audio con un formato `.mp3`, el cual se recomienda que se descargue, en lugar de solo escucharlo. El nombre del archivo es “*What’s My Title.mp3*” Al reproducirse, podemos escuchar código morse. Dicho código morse, es un fragmento de la letra de una canción. Un usuario podría asumir que la letra de la canción es la flag a introducir, pero se equivocaría.

Si el usuario ha descargado el archivo, podrá ver que la carátula del archivo `.mp3` es la siguiente imagen.



Figura 3.10: Imagen presentada disponible en el enlace proporcionado por la prueba.

Esta imagen contiene, escondida con la herramienta `steghide`, la verdadera flag de esta prueba. Para poder descifrarla será necesario haber descubierto el título de la canción y utilizar éste como clave para la extracción del texto oculto en la imagen.

3.5.10. Prueba de enunciado: Telegram PoC

Este reto no necesita ningún tipo de conocimientos especiales para ser solucionado, en cambio, su objetivo es presentarle a futuros desarrolladores una de las posibilidades que proporciona esta plataforma.

Cuando un usuario decide iniciar la prueba, se lanza programa el cual, utilizando la API de *bots* de Telegram [48], permite que los usuarios se comuniquen con él a través de Telegram. Este *bot* responde a los mensajes de usuario y le indicará que si ejecuta el comando `/token` con un nombre de usuario, responderá con una *flag*, única para ese usuario.

Si bien el algoritmo utilizado para generar dicha flag no tiene ningún interés (simplemente se genera una `hash SHA-512` con el nombre de usuario y una `salt`), es importante destacar que la flag es única para ese usuario. Esto abre muchas posibilidades, ya que podemos generar pruebas en las cuales los usuarios están obligados a realizar la prueba, no les basta con utilizar una flag que les haya compartido otro usuario.

Otro detalle interesante es la utilización de *bots* de **Telegram** y las posibilidades que abren. Un escenario interesante sería el de simulación de un entorno empresarial en el cual una de las fases de la prueba sería realizar pruebas de ingeniería social simuladas utilizando al *bot*.

3.5.11. Prueba de entorno individual: Deface Me

Esta prueba es la última que se ha desarrollado a la fecha de la redacción de esta memoria. Se basa en el uso de **Docker** para poder ofrecerle a cada usuario un entorno *sandbox* único.

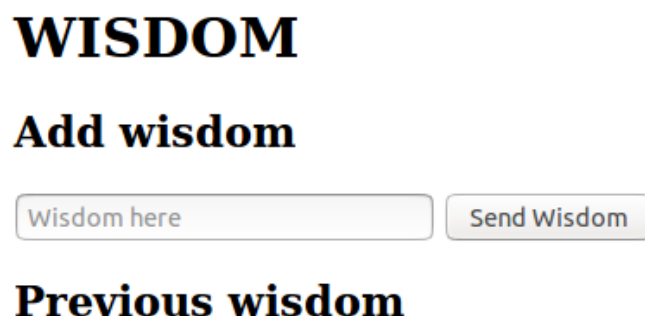


Figura 3.11: Estado inicial de la página web.

En esta prueba se le ofrece al usuario una pequeña página web, la cual se ha instanciado únicamente para él. Al usuario se le da una tarea: hacer que la página web aparezca completamente vacía.

Tras introducir datos en el formulario que aparece, se irán añadiendo entradas a la página web, como se ve reflejado en la siguiente figura.

WISDOM

Add wisdom

Previous wisdom

- Thing One
- Thing Two

Figura 3.12: Página web tras introducir algunos datos en el formulario

Para solucionar esta prueba es necesario que el participante realice un pequeño ejercicio de **XSS** (*Cross-Site Scripting*). Los ataques **XSS** son un tipo de ataque de inyección en el cual se inyectan *scripts* maliciosos en páginas web. Ocurren cuando un atacante utiliza una aplicación web para enviar código malicioso, en la forma de *scripts* para navegador. Esto se puede evitar si la aplicación web se encarga de “limpiar” las entradas que vengan de usuario (quitando etiquetas **HTML**, utilizando técnicas de *whitelisting*, etc).

En este caso, como se le dijo al usuario, el objetivo es vaciar de contenido a la página web, por lo que, en el campo del formulario, debemos introducir los siguiente:

```
<script>document.body.innerHTML=""</script>
```

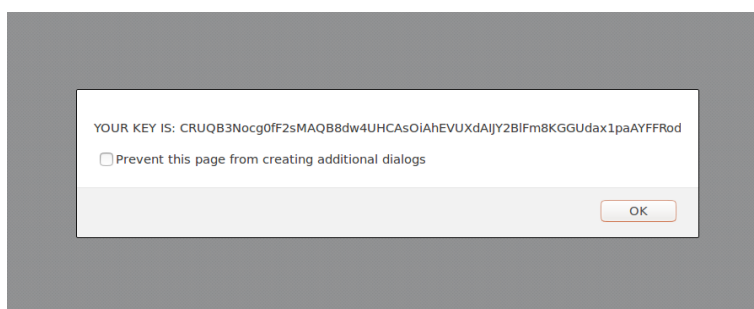


Figura 3.13: Mensaje de alerta enseñándole el flag al usuario.

Al introducir esto, el navegador nos dirá cuál es nuestro flag, el cual, al igual que en la prueba de **Telegram**, es único por cada usuario.

El código **javascript** que se utiliza para la generación de este *token* está ofuscado, de forma que sea mucho más complicado analizarlo sin haber realizado la prueba. En caso de que alguien obtenga el flag en cuestión analizando el código ofuscado, en lugar de realizando

el ataque XSS, no habría ningún problema, ya que la complejidad es mucho mayor y, es posible que ya supiese como realizar el ataque de todas formas.

Para la comprobación de la solución se utiliza un programa `javascript` en `NodeJS` [49], el cual realiza las mismas operaciones que el navegador para generar el flag.

También se han añadido a la plataforma pruebas desarrolladas por terceros, con el fin de demostrar la posibilidad y facilidad de integración de pruebas de diferentes tipos.

3.5.12. Pruebas de enunciado: Retos Criptored

Se han añadido a la plataforma cuatro desafíos que han sido creados por `criptored`[50] y se encuentran disponibles en <http://www.criptored.upm.es/paginas/criptoretos.htm>. Estos retos están orientados principalmente a la criptografía y esteganografía. Estos retos ya han sido resueltos y su solución se encuentra en la página citada anteriormente.

3.5.13. Prueba de entorno compartido: Metasploitable 2

En este reto se le presenta al usuario la famosa máquina virtual “Metasploitable 2”. Hemos dejado accesibles todos los puertos que estén abiertos en la máquina virtual, permitiéndole a un usuario realizar un escaneo de puertos y servicios.

El objetivo de la prueba es realizar una explotación al sistema y conseguir permisos de superusuario en el sistema, para poder leer la flag que está oculta en el fichero `/etc/shadow`, en el cual se encuentran los hashes de las contraseñas de los usuarios (aunque ese no es el objetivo).

Para completar la prueba, hay varios métodos, ya que ésta máquina virtual incluye multiples servicios vulnerables. Una ruta sería, después de escánear todos los puertos y servicios del sistema, utilizar `Metasploit` para realizar el ataque. El `exploit` `exploit/unix/irc/unreal_ircd.3281_backdoor` nos permite atacar una versión vulnerable de `ircd`[51]. Una vez obtenido el acceso, podemos leer el contenido de `/etc/shadow` y, al final del fichero, entraríamos el *flag* de esta prueba.

Con esta prueba se demuestra que es posible incorporar otras máquinas virtuales a la plataforma, necesitando muy poca configuración adicional.

3.6. Resultado Final

El código resultante y las pruebas anteriormente descritas conforman la plataforma que se presenta aquí. A continuación, se procederá a explicar el uso normal de la plataforma, por un lado, como participante, por otro lado, como diseñador de pruebas y, por último, como administrador.

3.6.1. Uso Como Participante

A continuación, se detallarán las páginas más relevantes que se puede encontrar un usuario a la hora de utilizar la plataforma y se darán explicaciones acerca de las decisiones tomadas cuando sea necesario.

La pantalla de inicio puede parecer bastante vacía, debido a que, como el objetivo de la plataforma es que cualquier usuario pueda alojar una propia, puede también personalizar el aspecto visual de éste. En esta página el usuario podrá iniciar sesión o, en caso de no tener una cuenta todavía, registrarse. La barra de navegación se transforma, una vez hayamos iniciado sesión, permitiéndonos avanzar a la sección de la página que más nos interese.

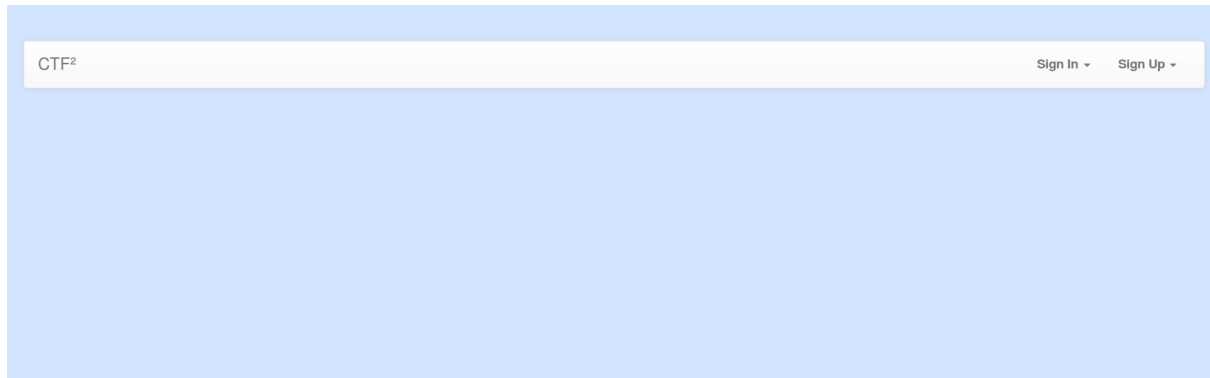


Figura 3.14: Página de inicio

Tras iniciar sesión, se nos dirige a nuestra página de usuario. El *path* para acceder a una página de cualquier usuario es `/user/[username]`. La plataforma distingue cuando hemos accedido a nuestra página de usuario y cuando estamos accediendo a la página de otro usuario, permitiendo en un futuro que se pueda utilizar esta página para que un usuario edite sus datos personales. En esta página, actualmente, aparece información del usuario como correo electrónico, puntos obtenidos y pruebas superadas.

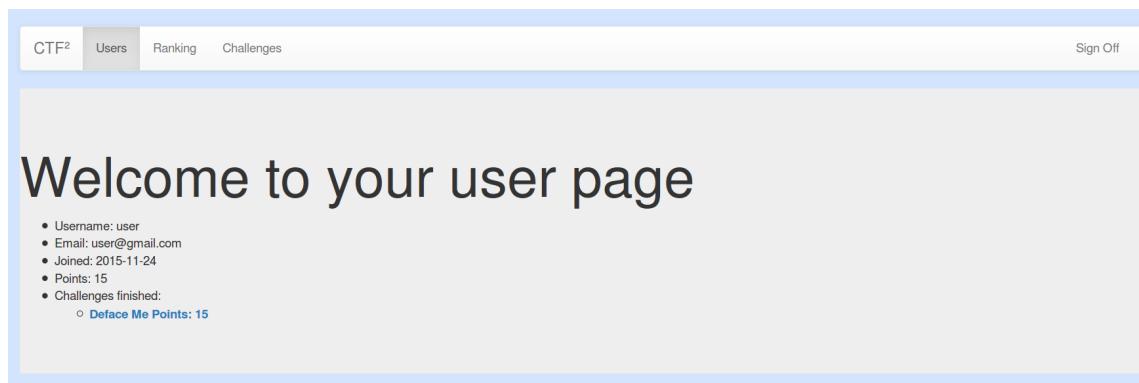


Figura 3.15: Página de usuario

Si accedemos al path `/user` o seleccionamos la opción “*Users*” en la barra superior de navegación, podemos ver un listado de todos los usuarios actualmente registrados en la plataforma. Si hacemos click en el nombre de algún usuario, se nos redirigirá a su página de usuario.

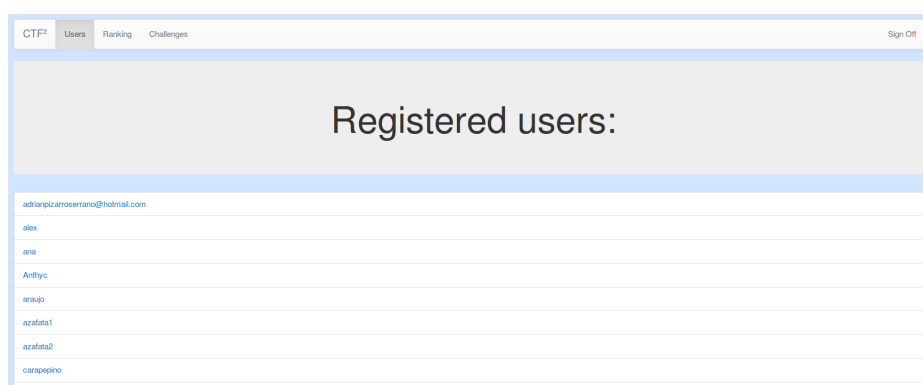
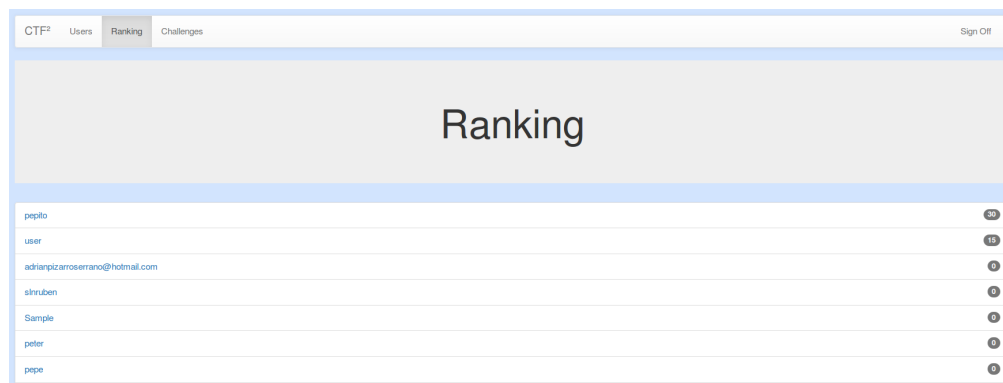


Figura 3.16: Directorio de usuarios

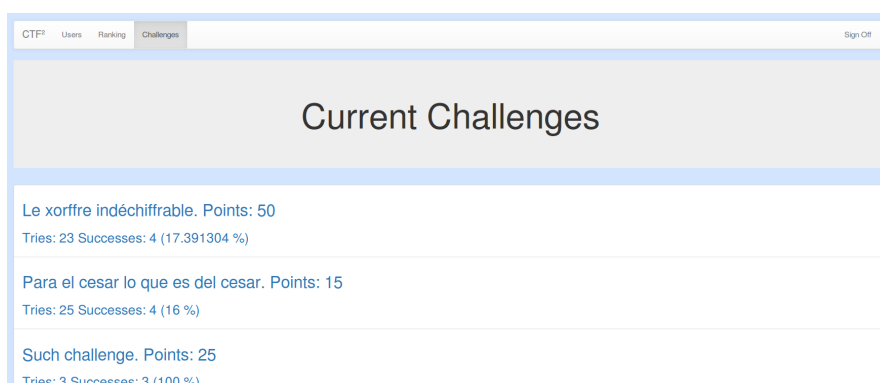
Accediendo a `/ranking` o seleccionando “*Ranking*” en la barra de navegación, llegaremos a la página que nos muestra el ranking de usuarios, según la puntuación que tengan hasta la fecha.



CTF²	Users	Ranking	Challenges	Sign Off
Ranking				
pepilo		20		
user		15		
adrianpizarroserano@hotmail.com		5		
slrhuben		5		
Sample		5		
peter		5		
pepe		5		

Figura 3.17: Página de ranking

El recurso `/challenge` (o pulsando la opción *Challenges*) nos muestra todos los desafíos que se encuentran actualmente en la plataforma, además de su puntuación máxima disponible y el porcentaje de aciertos de los usuarios por cada prueba. Seleccionando cualquier prueba accedemos a su página.



CTF²	Users	Ranking	Challenges	Sign Off
Current Challenges				
Le xorffre indéchiffrable. Points: 50 Tries: 23 Successes: 4 (17.391304 %)				
Para el cesar lo que es del cesar. Points: 15 Tries: 25 Successes: 4 (16 %)				
Such challenge. Points: 25 Tries: 3 Successes: 3 (100 %)				

Figura 3.18: Página de desafíos

Para acceder la página de una prueba podemos hacer click en ella desde el directorio de pruebas o acceder a `/challenge/[UID]`, donde UID es el identificador único de cada prueba. Inicialmente el usuario podrá ver el título de la prueba, su puntuación máxima, su descripción (la cual no debería ser muy extensa) y un botón para iniciar la prueba.

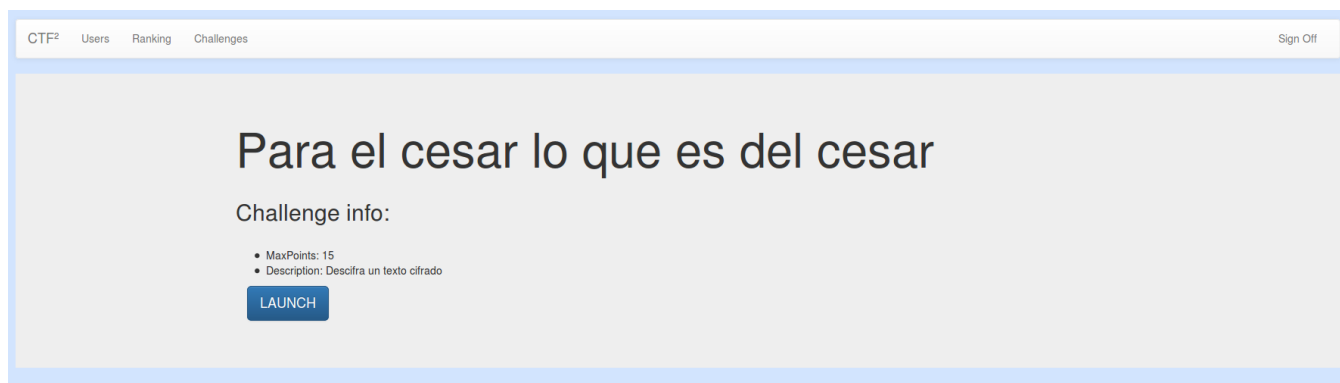


Figura 3.19: Página inicial de desafío.

Tras iniciar la prueba, se añadirá un texto más extenso en el cual estará el planteamiento de la prueba en sí. El botón de lanzar prueba se sustituirá, además, por un botón para detener la prueba y un formulario para introducir la solución.



Figura 3.20: Página de desafío una vez iniciado.

3.6.2. Uso Como Diseñador de Pruebas

Para desarrollar una prueba, lo más complicado debe ser la programación, la configuración de los entornos y el planteamiento del problema, no la incorporación a la plataforma. Hemos logrado esto dándole libertad casi absoluta a un diseñador de pruebas para poder trabajar con comodidad. Por lo tanto, un administrador debe cerciorarse de que la prueba en sí no contiene ningún código malicioso, tanto para el sistema en el cual la prueba se va a realizar, como para los usuarios que la utilicen.

Cada prueba debe existir en un directorio y, siempre que sea posible, deberá evitar acceder a recursos externos a ese directorio (salvo casos especiales, como por ejemplo el

wargame que se ha desarrollado en este proyecto, sería especialmente pesado si cada nivel del *wargame* tuviese que contener el disco duro de la máquina virtual). En el directorio raíz de la prueba debe haber obligatoriamente, un fichero llamado `info.json`. En este fichero debe estar codificados los siguientes valores:

- **Title**
- **Description**
- **Category**
- **MaxScore**
- **Creator**

En caso de ser administrador también, creando este directorio y fichero e introduciéndolos en el directorio donde se encuentran el resto de pruebas y lanzando la plataforma en modo `setup`, se nos generarán el resto de directorios y ficheros que necesitamos. No obstante, estos se pueden generar también manualmente. Uno de los ficheros generados es un fichero vacío llamado “YOUR_ID_IS_XXXXXXX”, donde XXXXXXXX es el UID de ésta prueba. Este podrá o no ser utilizado posteriormente por el desarrollador de la prueba.

También se generarán dos directorios, el primero, llamado `static`, es donde se pueden alojar ficheros estáticos como imágenes, ejecutables, códigos fuente, archivos de sonido, etc. La plataforma servirá única y exclusivamente los ficheros que se encuentren dentro de este directorio.

El segundo directorio se llama `rc` y dentro se deben encontrar tres ficheros, los cuales deben, a su vez, tener permisos de ejecución ya que la plataforma los ejecutará para iniciar y detener una prueba o comprobar una solución proporcionada por un usuario. Los tres ficheros en cuestión son los siguientes:

- **start_challenge**: Este fichero será ejecutado cuando un usuario pulse el botón de iniciar prueba. Recibirá como argumento el nombre del usuario que ha iniciado la prueba para, por ejemplo, mantener un registro de que usuarios se encuentran actualmente realizando la prueba o para generar entornos diferentes para cada usuario. Si `start_challenge` lanza otro ejecutable, por ejemplo, un *script* de `bash`

o `Python`, sería buena idea que dicho fuente se encontrase también en el directorio `rc`. Es muy importante recalcar que, la salida estándar de este fichero aparecerá en la página web, permitiendo presentar una descripción detallada de la prueba. Si la prueba en cuestión no tiene que ejecutar nada, sino que solo debe imprimir algo en la página web, con un simple `echo` de `bash` bastaría. La salida, además, será interpretada sin filtrar por el navegador web del usuario, por lo que se pueden incluir scripts o etiquetas HTML. En el caso de querer añadir enlaces a ficheros, su ruta deberá ser `"/challenge/[UID]/static/[FILE]"`, donde `UID` es el identificador de la prueba y `FILE` es el nombre del fichero, alojado en el directorio `static`. Además se asegura que, a la hora de arrancar una prueba, tendrá disponibles las variables `$CHALLENGE_ID` y `$CHALLENGE_PATH`, las cuales contendrán el `UID` de la prueba y el *path* absoluto hasta el directorio que contenga la prueba.

- **stop_challenge:** este fichero debería realizar la función opuesta a `start_challenge`. Recibe el nombre de usuario como argumento y se podría utilizar para eliminar a usuarios del registro de usuarios activos, eliminar ficheros generados por dicho usuario, eliminar una instancia de `Docker`, etc. En pruebas que no necesitan ejecutar nada, no es necesario que este fichero realice ninguna acción.
- **check_solution:** Este último fichero recibe dos argumentos, el primero es la solución enviada por el usuario y, el segundo, el nombre de usuario de quien ha enviado la solución. En pruebas que no generan soluciones diferentes para cada usuario, este argumento puede ser obviado.

3.6.3. Fichero `.ctff`

El fichero `.ctff` es un fichero `.tar.gz` o `.tgz` con todos los ficheros que conforman una prueba y el directorio que los contiene, es decir, los ficheros ejecutables del directorio `rc` (`start_challenge`, `stop_challenge` y `check_solution`), el fichero `info.json` y todos los ficheros dentro del directorio `static`. Cualquier fichero adicional no especificado, pero que sea utilizado por la prueba, debería incluirse también en el fichero `.ctff`.

No se intenta ofuscar ni ocultar información al utilizar esta extensión, pero su uso le da una mejor presentabilidad a la plataforma.

Si el desarrollador no es también administrador puede empaquetar todo el directorio de la prueba y su contenido en un fichero `.tar.gz` o `.tgz` y cambiar el nombre de dicha extensión a `.ctff` para poder compartir la prueba que ha creado.

3.6.4. Uso Como Administrador

Un administrador es cualquier persona que tenga acceso al código fuente de la plataforma, la base de datos y, al directorio que contiene las pruebas. Su principal trabajo se puede dividir en dos etapas:

- **Configuración inicial:** en esta etapa el administrador tendrá que preparar la base de datos para su uso (creando las tablas necesarias), personalizar la página de inicio de la plataforma (o cualquier otra página que el administrador vea oportuno) e instalar todas las pruebas que vea conveniente.
- **Mantenimiento continuo:** es la segunda etapa, el administrador se puede limitar a añadir o eliminar pruebas según vea conveniente.

Para la configuración de las páginas web, el administrador puede utilizar todos los *scripts* y hojas de estilo que vea necesarias, solamente tendrá que referenciarlos desde el código HTML que lo necesite y colocar dichos elementos en la carpeta `static/X/fichero` en el directorio donde se encuentre el ejecutable, donde X es un directorio que puede tener un nombre arbitrario (*scripts*, *stylesheets*, *images*, etc...). El estilo actual de la página utiliza **Bootstrap** [52]. Para la base de datos es necesario que las tablas cumplan con el formato que se explicó anteriormente y, tanto tablas como columnas, tengan los nombres especificados.

En cuanto a la instalación de pruebas iniciales, si ya tiene varias pruebas descomprimidas en el directorio de pruebas, pero no las tiene en la base de datos, puede lanzar la plataforma en el modo `setup`.

Para añadir o eliminar pruebas se pueden utilizar otros modos, los cuales ya se han enumerado anteriormente. Es recomendable utilizar primero el comando modo `list` para poder ver todas las pruebas actualmente instaladas y luego hacer `remove` de todas las pruebas que se deseen eliminar. Para la importación de pruebas (descargadas de internet, por ejemplo), lo común sería recibir un fichero `.ctff`, con el comando `install` y pasando como argumento el nombre del fichero. En este caso, se realizará la descompresión de la prueba en un directorio temporal, comprobación de que dicha prueba no existe actualmente en el entorno y, en caso de no existir, su movimiento al directorio de pruebas posterior introducción a la base de datos, según los datos hallados en el fichero `info.json`.

Capítulo 4

Conclusiones

A lo largo de los capítulos de este documento se ha descrito el proceso de construcción de diferentes retos de ciberseguridad y de una plataforma que permita su uso e incorporación de nuevos retos. En este capítulo analizaremos los objetivos que se describieron al principio de este documento y veremos en qué medida hemos sido capaces de cumplirlos. Además, se comentarán las competencias del grado que se han aplicado en este proyecto y las adquiridas en el transcurso del mismo. Por último, se hablará acerca de una serie de acciones futuras que podrían utilizarse para mejorar el proyecto actual.

4.1. Conclusiones

Como se ha dicho anteriormente, vamos a comparar los objetivos iniciales que nos propusimos al inicio del proyecto con los resultados obtenidos hasta la fecha.

- **Desarrollar una plataforma electrónica educativa para el desarrollo de las competencias necesarias para comprender, desarrollar y contrarrestar los distintos tipos de ataques a sistemas informáticos y redes a través del empleo de mecánicas de juego (gamificación).** Este objetivo se ha cumplido de manera exitosa, ya que, en efecto, hemos desarrollado dicha plataforma, junto con una serie de retos de ciberseguridad los cuales necesitan una serie de competencias específicas (conocimientos de criptografía, explotación en entornos Unix, ataques web, etc.) para ser superadas. Estas pruebas presentan siempre un enfoque similar al de un juego, intentando mantener la atención y el interés del usuario elevados. Además, la plataforma cuenta actualmente con un sistema de puntos y ranking, el cual permite que los usuarios se sientan motivados a realizar más pruebas, con el objetivo de ser los mejores del entorno.

- **Creación de un “juego de guerra” (wargame) multinivel en el que los estudiantes pueden atacar distintos sistemas aplicando diversas técnicas existentes dentro de un entorno controlado.** Este objetivo se cumplió, como se explicó en el apartado de desarrollo de pruebas, ya que se desarrolló un wargame con varios niveles. Además de esto, se desarrollaron muchas pruebas más que se enfocaban en competencias diferentes.
- **Aplicación de diversas técnicas de gamificación para influir y motivar a los estudiantes en el estudio de la seguridad de los sistemas operativos y los sistemas distribuidos.** Como se explicó anteriormente en el análisis del primer objetivo, esto se ha logrado en cierta medida, incluyendo sistemas de puntuaciones y rankings en la plataforma. No obstante, se podrían implementar aún más, como se explicará en la sección “Trabajo Futuro”
- **Crear una plataforma genérica y libre, que esté disponible a través de Internet para su uso en otras organizaciones. Al consistir en niveles, este proyecto será continuado en el tiempo de forma natural, añadiendo nuevos niveles, distintas características de gamificación y potenciando su uso mediante la interacción con distintas redes sociales.** Al igual que en el objetivo anterior, esta plataforma es totalmente genérica y será libre. De esta forma se podrán añadir funcionalidades a la plataforma y pruebas al entorno. Debido a la madurez de esta plataforma, podría ser utilizada en las titulaciones oficiales de la Escuela Técnica Superior en Ingeniería de Telecomunicación de la URJC como material didáctico.

4.2. Competencias Utilizadas y Adquiridas

Durante el desarrollo de este proyecto se han utilizado una serie de conocimientos obtenidos en el Grado, como configuración de sistemas operativos, programación orientada a objetos, uso de concurrencia, conocimientos acerca de criptografía y seguridad de la información e incluso conocimientos teóricos acerca de *exploits* y ataques criptográficos.

Aun así, algunos de los conocimientos necesarios para el desarrollo de este proyecto no habían sido obtenidos en el transcurso del grado, como pueden haber sido: explotación de sistemas UNIX, gestión de bases de datos y, el más importante, gestión del tiempo y de recursos en el desarrollo de un proyecto grande.

4.3. Trabajo Futuro

A continuación, se desarrollarán una serie de funcionalidades que se podrían añadir a la plataforma con relativa sencillez y mejorarían significativamente la calidad del producto desarrollado.

- **Implementación de más técnicas de gamificación:** aunque se implementa actualmente un sistema de puntuación y de ranking, se podrían añadir más elementos que estimulasen constantemente al usuario para seguir utilizando la plataforma y desarrollando sus conocimientos en el área de la ciberseguridad. Por ejemplo, se podría incluir un sistema de logros, en el cual se le otorgasen una serie de medallas o algo similar a un usuario por cumplir un conjunto de requisitos, como se ve en la siguiente imagen. También se podría incluir un foro o algo similar para que los usuarios pudiesen comunicarse entre sí y compartir opiniones.

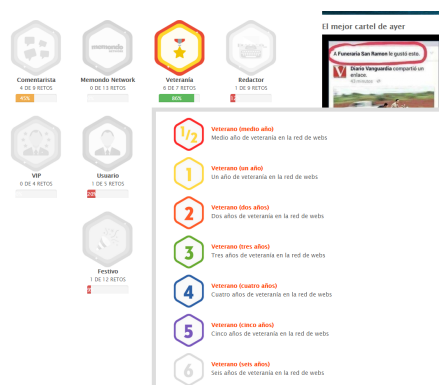


Figura 4.1: Ejemplo de página de logros

- **Añadir integración con redes sociales:** con esto un usuario podría compartir sus éxitos con sus amigos en redes sociales, dándole más visibilidad a la plataforma.
- **Mejorar el aspecto visual general de la plataforma:** Actualmente la plataforma tiene un aspecto visual bastante simple; aunque es cierto que se espera que un administrador lo modifique a su gusto, estaría bien darle más estilo a la página web.
- **Mejorar las herramientas de administración:** hay varias herramientas de administración que se podrían añadir a la plataforma para hacerle la vida más fácil a un administrador, como por ejemplo una creación automática de tablas en la base

de datos, una interfaz gráfica para la gestión de pruebas que se encuentren en un entorno y, quizás, una sección de administración en la página web, que permitiese modificar valores de la base de datos sin tener que entrar en ella con una línea de comandos e introducir las instrucciones **SQL** de forma manual.

- **Añadir más pruebas:** Lo bueno (o malo, según se vea) de este proyecto, es que siempre se pueden realizar más pruebas, que se enfoquen en nuevos ataques, o que le den una orientación diferente a ataques ya conocidos. Siempre se pueden crear nuevos *wargame*, enlazar sistemas criptográficos de diferentes maneras, etc. La imaginación y el conocimiento en el área es el límite.

Bibliografía

- [1] The jargon file, 2004. <http://www.catb.org/jargon/>.
- [2] The search engine for internet of the things. <https://www.shodan.io/>.
- [3] Maltego. <https://www.paterva.com/web7/>.
- [4] Nmap: the network mapper. <https://nmap.org/>.
- [5] Nessus vulnerabaility scanner. <http://www.tenable.com/products/nessus-vulnerability-scanner>.
- [6] Openvas. <http://www.openvas.org/>.
- [7] Metasploit. <https://www.metasploit.com/>.
- [8] Offensive security. <https://www.offensive-security.com/>.
- [9] Kali linux. <https://www.kali.org/>.
- [10] Backtrack linux. <http://www.backtrack-linux.org/>.
- [11] Wifislax. <http://www.wifislax.com/>.
- [12] Santoku linux. <https://santoku-linux.com/>.
- [13] Rapid 7. <https://www.rapid7.com/>.
- [14] Cisco certifications. <http://www.cisco.com/c/en/us/training-events/training-certifications/certifications.html>.
- [15] Ec council. <https://www.eccouncil.org/>.
- [16] Owasp. https://www.owasp.org/index.php/Main_Page.

- [17] Owasp broken web applications project. https://www.owasp.org/index.php/OWASP_Broken_Web_Applications_Project.
- [18] Cybercamp. <https://cybercamp.es/>.
- [19] Capture the flag daemon. <https://github.com/isislab/CTFd>.
- [20] Fbctf (facebook ctf. <https://github.com/facebook/fbctf>.
- [21] Bootstrap. <https://github.com/Nakiامي/mellivora>.
- [22] scorebot. <https://github.com/dichotomy/scorebot>.
- [23] Nightshade. <https://github.com/UnrealAkama/NightShade>.
- [24] shuriken-framework. <https://github.com/samuraictf/shuriken-framework>.
- [25] Modelo de desarrollo en espiral. <http://dl.acm.org/citation.cfm?doid=12944.12948>.
- [26] The go programming language. <https://golang.org/>.
- [27] Gin web framework. <https://gin-gonic.github.io/gin/>.
- [28] Revel web framework. <https://revel.github.io/>.
- [29] Beego web framework. <http://beego.me/>.
- [30] Package http. <https://golang.org/pkg/net/http/>.
- [31] Package template. <https://golang.org/pkg/html/template/>.
- [32] Vagrant. <https://www.vagrantup.com/>.
- [33] Vmware. <http://www.vmware.com/>.
- [34] Aws (amazon web services). <https://aws.amazon.com/es/>.
- [35] Chef configuration manager. <https://www.chef.io/solutions/configuration-management/>.
- [36] Puppet. <https://puppet.com/>.
- [37] Metasploitable 2. <https://community.rapid7.com/docs/DOC-1875>.

- [38] Docker. <https://www.docker.com/>.
- [39] Performance of Docker vs VMs. <http://es.slideshare.net/Flux7Labs/performance-of-docker-vs-vms>, 08 2014.
- [40] Problema del cumpleaños. <http://mathworld.wolfram.com/BirthdayProblem.html>.
- [41] Key derivation functions. <https://cryptography.io/en/latest/hazmat/primitives/key-derivation-functions/>.
- [42] Notación crow's foot. <http://www2.cs.uregina.ca/~bernatja/crowsfoot.html>.
- [43] Método kasiski. <http://crypto.interactive-maths.com/kasiski-analysis-breaking-the-code.html>.
- [44] Steghide. <http://steghide.sourceforge.net/>.
- [45] Gdb: The gnu project debugger. <https://www.gnu.org/software/gdb/>.
- [46] Página de manual: gets. <http://man7.org/linux/man-pages/man3/gets.3.html>.
- [47] Chema Alonso. *Pentesting con Foca*. 0xWORD, 2013.
- [48] Telegram bot api. <https://core.telegram.org/bots/api>.
- [49] Nodejs. <https://nodejs.org/en/>.
- [50] Criptored. <http://www.criptored.upm.es/>.
- [51] Cve-2010-2075. <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2075>.
- [52] Bootstrap. <http://getbootstrap.com/>.
- [53] Alan A. A. Donovan and Brian W. Kernighan. *The Go Programming Language*. Addison-Wesley, 10 2015.
- [54] O.S. Tezer. Sqlite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management Systems. <https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems>, 02 2014.

- [55] Rob Pike. Simplicity is Complicated. <https://talks.golang.org/2015/simplicity-is-complicated.slide>, 11 2015.
- [56] B Boehm. A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes*, 11(2):14–24, 7 1986.
- [57] astaxie. *Build Web Application with Golang*. GitBook, 2015. <https://astaxie.gitbooks.io/build-web-application-with-golang/content/en/>.
- [58] Penetration testing execution standard technical guidelines, 2012. http://www.pentest-standard.org/index.php/PTES_Technical_Guidelines.
- [59] Jon Erickson. *Hacking: The art of Exploitation*. Willam Pollock, 2010.
- [60] David Puente Castro. *Linux Exploiting. Técnicas de explotación de vulnerabilidades en Linux para la creación de exploits*. 0xWORD, 2014.
- [61] Pablo González Perez. *Metasploit para Pentesters*. 0xWORD, 3 edition, 10 2014.
- [62] Pablo González Pérez, Germán Sánchez Garcés, and Jose Miguel Soriano de la Cámara. *Pentesting con Kali*. 0xWORD, 2013.
- [63] Jordi Serra and Daniel Lerch. *Esteganografía y Estegoanálisis*. 0xWORD, 2013.
- [64] Virtualbox. <https://www.virtualbox.org/>.

Anexo I: URL de descarga

URL de descarga de código fuente y pruebas

La URL del repositorio en el cual se encuentra alojado el código fuente de la plataforma es: <https://github.com/jenarvaezg/ctff> y las pruebas desarrolladas se pueden descargar en el siguiente enlace: <https://dl.dropboxusercontent.com/u/70003427/challenges.tar.gz>