UNIVERSITAT DE BARCELONA

Trabajo de Final de Grado

GRADO DE INGENIERÍA INFORMÁTICA

Facultad de Matemáticas e Informática
Universidad de Barcelona

Ataques y vulnerabilidades web

Jaume Campderrós Vilà

Tutor: Prof. Raúl Roca Cánovas
Barcelona, mayo de 2019

AGRADECIMIENTOS

Este proyecto es el final de una etapa de varios años de trabajo y dedicación, de noches sin dormir y muchos dolores de cabeza, la cual también ha proporcionado alegrías, conocimiento y buenos amigos.

Agradezco este trabajo a mi familia, por apoyarme todos estos años y no dejar que me diese por vencido, a mis compañeros de universidad, tanto a los que siguen conmigo como a los que ya han acabado, los cuales sin ellos no seguiría aquí.

También quiero dedicárselo a todos los profesores de la facultad y del grado superior que me han acompañado y guiado durante esta etapa.

PRÓLOGO

Desde muy pequeño me empecé a aficionar por la tecnología y los ordenadores (sobre todo por los videojuegos) gracias a mi padre, el cual siempre traía a casa los últimos ordenadores o multimedia que salían al mercado.

Esta afición ha marcado mi trayectoria académica y profesional. Estudiando un grado medio y seguidamente un grado superior en administración de sistemas informáticos empecé la universidad tomándola como un objetivo personal.

Asimismo todo este tiempo he estado trabajando como administrador de sistemas, con cinco años de experiencia. Durante este tiempo empecé a interesarme por la seguridad y cada vez tenía más claro que quizás en algún momento estudiaría más a fondo todo este mundo.

Este trabajo es la realización personal del estudio de la ciberseguridad con el fin de estudiar y entender las distintas vulnerabilidades web que existen.

RESUMEN

Este proyecto trata de explicar y dar a entender la importancia de seguridad en las aplicaciones web, explicando de una forma clara y concisa las vulnerabilidades más importantes junto a sus posibles soluciones.

El trabajo se dividirá en varias secciones, las cuales serán desde explicar y analizar el cómo encontrar vulnerabilidades en una aplicación web a hacer un estudio de las distintas vulnerabilidades junto a un caso práctico de cada una de ellas mediante la realización de pruebas a través de máquinas virtuales ya diseñadas para el pentesting.

Para cada una de las vulnerabilidades, a parte del contenido descrito en esta memoria se han incluído vídeos explicativos con resúmenes muy sencillos para entenderlas fácilmente.

El término hacker tiene distintos significados. Según el propio diccionario de los hackers, es todo individuo que se dedica a programar de forma entusiasta, osea un experto entusiasta de cualquier tipo, que considera que poner la información al alcance de todos constituye un extraordinario bien. La RAE establece que una persona experta en el manejo de computadoras, que se ocupa de la seguridad de los sistemas y de desarrollar técnicas de mejora.

Se espera que con este trabajo de fin de grado se provea de una idea general sobre el mundo del hacking ético y las distintas vulnerabilidades que existen en el mundo de las web apps.

La mayor dedicación de tiempo invertido en este TFG ha sido para formarme en los distintos conceptos de ciberseguridad y en mapear estos con todos los conocimientos obtenidos en la carrera.

ÍNDICE

ÍNDICE DE ILUSTRACIONES

1. INTRODUCCIÓN

Las aplicaciones web son las herramientas más utilizadas por las organizaciones alrededor del mundo. Estas aplicaciones ofrecen muchos beneficios para las empresas los cuales pueden ser desde agilización de procesos, optimizar comunicaciones a digitalizar servicios. No obstante, estas aplicaciones o páginas web son el objetivo constante para la explotación de vulnerabilidades.

De acuerdo con el informe anual de Acunetix "Web Application Vulnerability 2019" nos muestra que más del 60% de los sitios web presenta problemas de vulnerabilidades graves, que son la amenaza más crítica en empresas. Ante este grave problema es necesario encontrar nuevos enfoques para la mitigación de vulnerabilidades las cuales van aumentando año tras año.

Uno de los principales problemas del diseño de las web apps y de sus errores de seguridad, es que muchas de las empresas están bajo presión para entregar las aplicaciones y servicios web de una manera rápida para satisfacer la demanda de los clientes. Esta prisa por entregar las aplicaciones rápido y cumplir el plazo de entrega, provoca que la aplicación no está a la par con el ciclo de desarrollo de seguridad. A día de hoy hay un mayor número de defectos potenciales que hace un año, y esto va incrementando.

Las empresas deberían tener en cuenta que una vulnerabilidad grave o crítica implicaría al atacante obtener acceso a datos e información sensible, información de clientes, cuentas bancarias, etc.

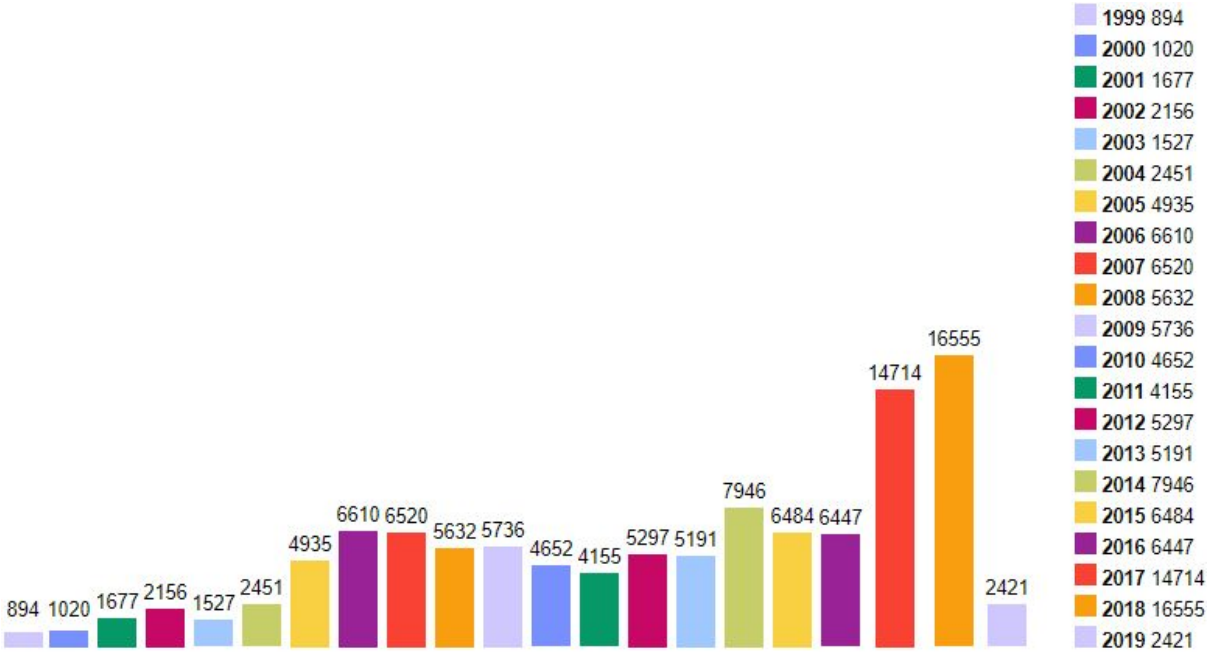


Ilustración 01 - Vulnerabilidades por año - Fuente: CVE Details

En este gráfico proporcionado por CVE Details podemos ver la evolución de la cantidad de vulnerabilidades detectadas en los sistemas a través de los años (Hasta Marzo de 2019). Podemos sacar varias conclusiones respecto el estado de seguridad de los sistemas operativos y aplicaciones más utilizadas actualmente. Una de las conclusiones más obvias es que el total de vulnerabilidades descubiertas ha ido creciendo sustancialmente por segundo año consecutivo después de una relativa estabilidad.

Este aumento es muy significativo y nos viene a decir que cada vez existen más y más vulnerabilidades. De las 16.555 vulnerabilidades descubiertas en 2018, casi un 5% han sido reportadas como críticas. Este porcentaje puede parecer pequeño, pero dependiendo del sistema afectado y el tipo de vulnerabilidad, este puede ser bastante importante.

El objetivo de este proyecto es exponer y explicar todas las vulnerabilidades web más frecuentes y más importantes, hacer un análisis de éstas y mostrar mediante casos prácticos cómo podemos detectar dichas vulnerabilidades.

2. OBJETIVOS

El principal objetivo de este trabajo es estudiar y analizar las distintas vulnerabilidades web mediante un análisis de éstas mismas y los distintos casos de pentesting para ponerlas a prueba. Cabe destacar que el objetivo no es hacer un tutorial sobre cómo hacerlo, sino un estudio.

Dichos objetivos podemos enumerarlos de la siguiente forma:

- Analizar y conocer los procedimientos a la hora de detectar vulnerabilidades en una página o aplicación web.
- Explicar la evolución o los pasos a seguir para para hacer una investigación de posibles vulnerabilidades.
- Realizar un caso práctico para demostrar y analizar cada una de las vulnerabilidades explicadas.
- Estudiar y analizar distintos métodos para evitar los fallos o vulnerabilidades en las aplicaciones web.

3. SEGURIDAD EN APLICACIONES WEB

El concepto de seguridad puede asociarse a la capacidad de protección o peligros sobre aquello que nos preocupa o consideramos importante. La seguridad se refiere principalmente a la mitigación o supresión de peligro, daño o riesgo.

Existen muchos tipos o conceptos de seguridad, tantos como peligros existan, como puede ser la seguridad laboral, urbana, pública, bancaria, etc. Todos los tipos con un mismo fin, mitigar los riesgos asociados al ámbito en cuestión.

Este trabajo se enfoca en la seguridad informática y la seguridad de la información en las aplicaciones web, que son las dos acepciones más actuales y de moda en los últimos tiempos.

Las aplicaciones web son la cara frontal y visual del software que proporcionan el acceso a la información y a los distintos procesos de los sistemas. La información contenida en ellas suele ser el activo de mayor valor en el contexto de seguridad de la información.

Esta información se suele almacenar en soportes físicos o virtuales que comprenden la infraestructura que se debe proteger. Este suele ser el objetivo de los atacantes, los cuales intentan vulnerar la seguridad para conseguir acceso a datos sensibles, destruirlos o hacerlos inaccesibles.

3.1. El valor de los datos en la red

El resultado de una actividad maliciosa llevado a cabo por un atacante cibernético sigue a un mismo principio, ¿que beneficio obtengo con esa actividad?

Este beneficio para el atacante es relativo, ya que puede ser obtenido de varias formas. Un atacante puede moverse porque alguien contrata sus servicios, podemos hacer una búsqueda rápida en google y ver los precios de hacking malicioso en el mercado negro. También es posible que dicho atacante o hacker esté realizando un "bounty program" para una empresa, o simplemente que lo haga para conseguir prestigio.

De todas formas, según los datos que tenga como objetivo o el sistema en cuestión, dictará más o menos el beneficio obtenido. Existen varios factores, entre otros, que pueden servir de motivación para atacar una aplicación web:

- **Volumen de información almacenado en la web.** La información normalmente se encuentra almacenada en bases de datos de aplicaciones web. De modo que cuando un atacante compromete la base de datos de una aplicación, suele acceder a la información de todos los usuarios que contiene.

- **Cantidad de servicios donde estamos registrados.** Podrían ser por ejemplo agua, luz, telefonía, la empresa donde trabajamos, redes sociales, foros, etc. Al estar registrados en distintas aplicaciones web se dispone de una superficie de ataque mayor.

- **Vulnerabilidades en los sistemas de información.** Un desarrollo de software sin atender a la perspectiva de seguridad de la información, una configuración pobre o inadecuada sobre los servicios donde se despliega la web o la detección de bugs de seguridad. Son aspectos que derivan en fallos de seguridad que perduran en el tiempo, haciendo más probable el éxito de los ataques.

- **Entes que compran la información.** Ya bien sean individuos, organizaciones, estados, investigaciones privadas, espionaje, redes de inteligencia, etc. Cualquier persona o colectivo puede pagar por la información alcanzada.

3.2. Bounty programs

En los últimos años, la industria tecnológica ha experimentado un giro radical en la forma de relacionarse con los ciber delincuentes. La mejor forma de protegerse de los más destructivos es incentivar a los demás para que busquen errores y vulnerabilidades en sus propios sistemas, premiandolos de una forma económica para que den la oportunidad a las compañías de solventarlos antes de que se hagan públicos. Estas prácticas tienen el nombre de “bounty programs” o “bug bounty programs”.

Hay estudios realizados (Por ejemplo éste) basados en grandes empresas como Google y Mozilla los cuales demuestran que esta práctica de recompensas por los posibles fallos del sistema propio, es más barata y eficiente en comparación que las antiguas estrategias basadas en la contratación de consultores externos o por ejemplo a la automatización de las revisiones de código.

	accounts.google.com	Other highly sensitive services [1]	Normal Google applications	Non-integrated acquisitions and other lower priority sites [2]
Remote code execution	$20,000	$20,000	$20,000	$5,000
SQL injection or equivalent	$10,000	$10,000	$10,000	$5,000
Significant authentication bypass or information leak	$10,000	$5,000	$1,337	$500
Typical XSS	$3,133.7	$1,337	$500	$100
XSRF, XSSI, and other common web flaws	$500 - $3,133.7 (depending on impact)	$500 - $1,337 (depending on impact)	$500	$100

Ilustración 02 - Tabla de recompensas de Google - Fuente: helpnetsecurity

Hay una lista enorme de compañías que han apostado o actualmente apuestan por los “bug bounty programs”, entre ellas Google, Facebook, Dropbox, Microsoft o Yahoo. El uso de dichos programas se está convirtiendo en el pan de cada día y una tendencia en la industria tecnológica. Estas empresas están empezando a optar por externalizar la gestión de sus “bounty programs” y gracias a ello están surgiendo muchas startups dedicadas a hacerse cargo de estas tareas.

Podemos ver una lista de una gran cantidad de recompensas que están activas en este mismo momento en el famoso portal de hackerone.

3.3. Informes de vulnerabilidades

Los hackers o investigadores que descubren vulnerabilidades en una aplicación web, ya sea mediante un bounty program o por cuenta propia, suelen informar a los fabricantes o desarrolladores correspondientes. En muchos casos, esta tarea suele ser difícil ya que la intencionalidad juega un rol fundamental en esta etapa y define cómo se llevarán a cabo las actividades para mitigar o suprimir las vulnerabilidades descubiertas.

Estas vulnerabilidades encontradas deben ser informadas a la entidad responsable para que la misma pueda ser tratada y mitigada en el menor tiempo posible. La idea es que el hacker reporte la vulnerabilidad directamente a la compañía mediante un canal privado. Esto es debido a que si la vulnerabilidad se publica, esta podría ser aprovechada por otros atacantes y explotarse con fines no legítimos.

Uno de los posibles riesgos que asumen los hackers es que la entidad responsable no contemple el report como un acto de buena fe y pueda tomar acciones legales contra los mismos. No obstante, la seguridad se hace con los aportes de todos, y es necesario reportar las vulnerabilidades antes de que alguien con malas intenciones pueda tomar provecho de ellas.

A continuación vamos a enumerar una serie de puntos que se utilizan para realizar estos informes.

1. Leer las pautas proporcionadas por la empresa.

Cada compañía participante en los bounty programs tiene listas o guías de cómo entregar los informes o el código necesario para informar una vulnerabilidad. Es de vital importancia leer estas guías o pautas proporcionadas para no perder el tiempo al entregar un informe el cual ni siquiera se van a leer al no tener el formato concreto.

2. Incluir muchos detalles.

Si se quiere que un informe sea tomado en serio ha de proporcionarse un informe detallado el cual debería incluir como mínimo:

- La URL y los parámetros afectados utilizados para encontrar la vulnerabilidad.
- Una descripción del navegador, sistema operativo (si corresponde) y/o versión de la aplicación.
- Una descripción del impacto percibido. ¿Cómo podría ser explotado el error?
- Pasos para reproducir el error.

Todos estos criterios son comunes para todas las grandes compañías, en caso de querer ir un paso más lejos sería recomendable incluir capturas de pantalla o algún vídeo de prueba. Ambos son de gran ayuda para las compañías y ayudarán a entender la vulnerabilidad.

3. Confirmar la vulnerabilidad.

Una vez leídas las pautas proporcionadas, creado y redactado el informe e incluso se han incluído capturas de pantalla, es necesario tomarse un momento para asegurarse de lo que se está informando es una vulnerabilidad actualmente.

Hay que asegurarse de se ha confirmado la vulnerabilidad antes de enviar el informe. Puede ser una decepción muy grande pensar que se ha encontrado una vulnerabilidad significativa cuando en realidad se ha malinterpretado algo durante las pruebas.

4. Mostrar respeto hacia la compañía.

Cuando una empresa lanza un nuevo bounty program pueden recibir muchísimos informes. Una vez enviado el propio, hay que permitir a la empresa la oportunidad de revisarlo y ponerse en contacto con uno. Algunas compañías publican sus timelines en sus guías o pautas, pero puede ser que otras empresas no lo hagan.

En caso de enviar el informe y no tener noticias de la compañía en dos semanas por lo menos se puede postear un mensaje en el informe para saber si hay alguna actualización o notícia. Por otro lado, en caso de que la compañía confirme la vulnerabilidad, se puede trabajar con ellos para confirmar cuando lo hayan arreglado.

Durante un bounty program hay un listado de problemas los cuales incluyen:

- **Ruido:** Los bug bounty programs reciben muchos informes inválidos.
- **Priorización:** Los bounty programs tienen que encontrar una manera de priorizar la solventación de vulnerabilidades. Esto es complicado cuando tienes múltiples vulnerabilidades con un impacto similar combinado con muchos informes entrando continuamente.
- **Confirmaciones:** Cuando se revisa un informe, los bugs tienen que ser validados y esto necesita tiempo. Es por eso que es necesario dar instrucciones claras y una explicación sobre que se ha encontrado, como reproducirlo y por qué es importante.
- **Recursos:** No todas las compañías pueden permitirse dedicar un equipo de personal para mantener un bounty program. Es posible que algunas compañías tengan incluso una única persona dedicada a este tema.
- **Solucionar el problema:** Programar necesita tiempo, especialmente si hay un ciclo de desarrollo que puede incluir debugging, tests, deployment y producción. Es por ello que las líneas de comunicación son esenciales y la necesidad de que se respeten ambas partes.
- **Manejar relaciones:** Los bug bounty programs quieren que los hackers vuelvan. Es por ello que es necesario crear una especie de buena relación con ellos.
- **Relaciones con la prensa:** Siempre está la presión de que un bug pueda ser pasado por alto, tarde mucho tiempo en resolverse o una recompensa de pago sea demasiado baja. Los hackers enfadados (con un buen motivo) pueden acudir a Twitter o a los medios.

5. Recompensas.

En caso de enviar un informe a una compañía que paga una recompensa, hay que respetar su decisión de la cantidad pagada. Normalmente los bounty programs tienen una lista o tabla con el dinero pagado por cada vulnerabilidad encontrada.

Si se está en desacuerdo con una cantidad recibida, hay que mantener y argumentar una discusión de por qué se cree que merece una recompensa más elevada. Hay que evitar situaciones donde se reclama otra recompensa más elevada sin elaborar el por qué lo crees. La compañía debería mostrar respeto por el tiempo y el valor invertido.

6. “No vendas la piel del oso antes de cazarlo.”

Este dicho viene a significar que no deberías celebrar la victoria hasta estar totalmente seguro de los resultados obtenidos. Ha ocurrido muchas veces que algún hacker de renombre ha reportado un bug o vulnerabilidad grave a la empresa del bounty program, y antes de recibir el ok y la recompensa, éste ya estaba alardeando a través de las redes sociales y haciendo un report. Seguidamente la empresa marcó como inválido el informe al no considerar un bug el problema.

Escribir o hacer buenos informes es esencial tanto para las compañías interesadas en solucionarlos como para el hacker en cuestión. Igualmente se puede aplicar al hecho de encontrar bugs más importantes o menos. En algunos portales como HackerOne, los informes colgados se puntúan según unos parámetros que tienen. Los buenos hackers que consiguen muchos puntos, consiguen invitaciones a bounty programs cerrados con más dinero en recompensas y más prestigio entre la comunidad.

Con esto quiero decir que el escribir buenos informes siempre aportará beneficios de una forma u otra.

4. PENETRATION TESTING

Esta sección recoge la parte práctica del proyecto, la cual consiste en hacer dos casos prácticos de penetration testing (pentesting).

Estos dos casos prácticos se realizan mediante el uso de varias máquinas virtuales las cuales contienen servicios web preparados. El objetivo de estos casos es detectar todas las vulnerabilidades posibles y demostrar cómo se realiza el proceso.

El primer caso práctico será con una máquina virtual preparada para pentesting llamada BadStore. Esta máquina virtual proporciona una página web con distintas vulnerabilidades preparadas a propósito.

El segundo caso práctico será sobre la página web realizada en la asignatura optativa “COWEB” (Computación orientada al web) e impartida por el profesor Simone Balocco. Dicha página web contiene varias vulnerabilidades las cuales se identificarán y se aplicarán las distintas correcciones, documentando el proceso. Cabe puntuar que el objetivo de la asignatura no era el de la seguridad y por ello existen las vulnerabilidades que se comentarán.

4.1. Caso práctico - BadStore

Este caso práctico de pentesting consiste en utilizar la aplicación web BadStore la cual montamos en una máquina virtual para poder atacar desde otra máquina virtual con Linux y enumerar las distintas vulnerabilidades que encontramos.

4.1.1. Escenario

El escenario en este caso práctico consiste en tener dos máquinas virtuales sobre Linux las cuales se ejecutan sobre VirtualBox. La primera de ellas es BadStore la cual descargamos la imagen ISO desde el siguiente enlace: *https://www.vulnhub.com/entry/badstore-123,41/*

Esta máquina virtual hará de servidor el cual aloja la aplicación web de BadStore y sobre la cual haremos las distintas pruebas o ataques de vulnerabilidades. La otra máquina virtual es un Kali Linux la cual utilizaremos para realizar las distintas pruebas. Creamos una máquina virtual basada en Debian la cual instalamos la ISO de Kali que podemos encontrar en el siguiente enlace: *https://www.kali.org/downloads/*

Una vez tenemos instaladas y preparadas las dos máquinas virtuales ya podemos arrancar BadStore primero y dejarla encendida en segundo plano mientras trabajamos con Kali.

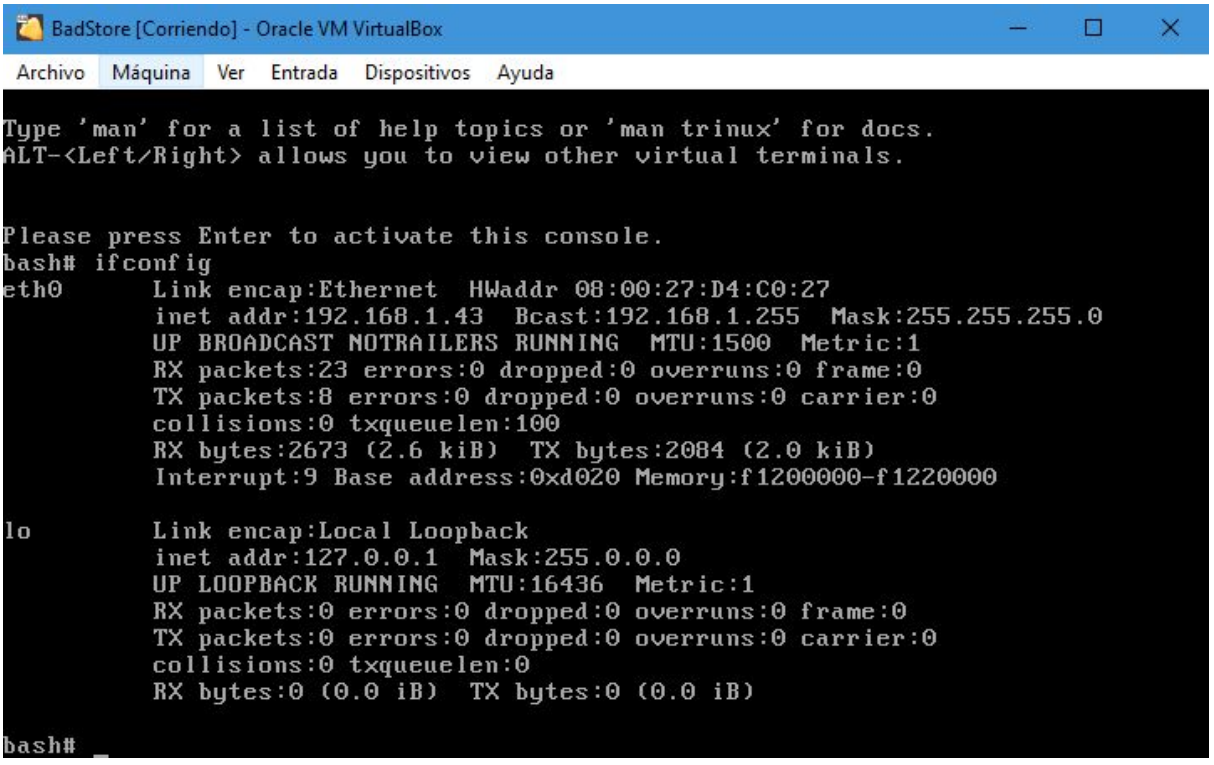


Ilustración 03 - Ejecución de la máquina virtual de BadStore

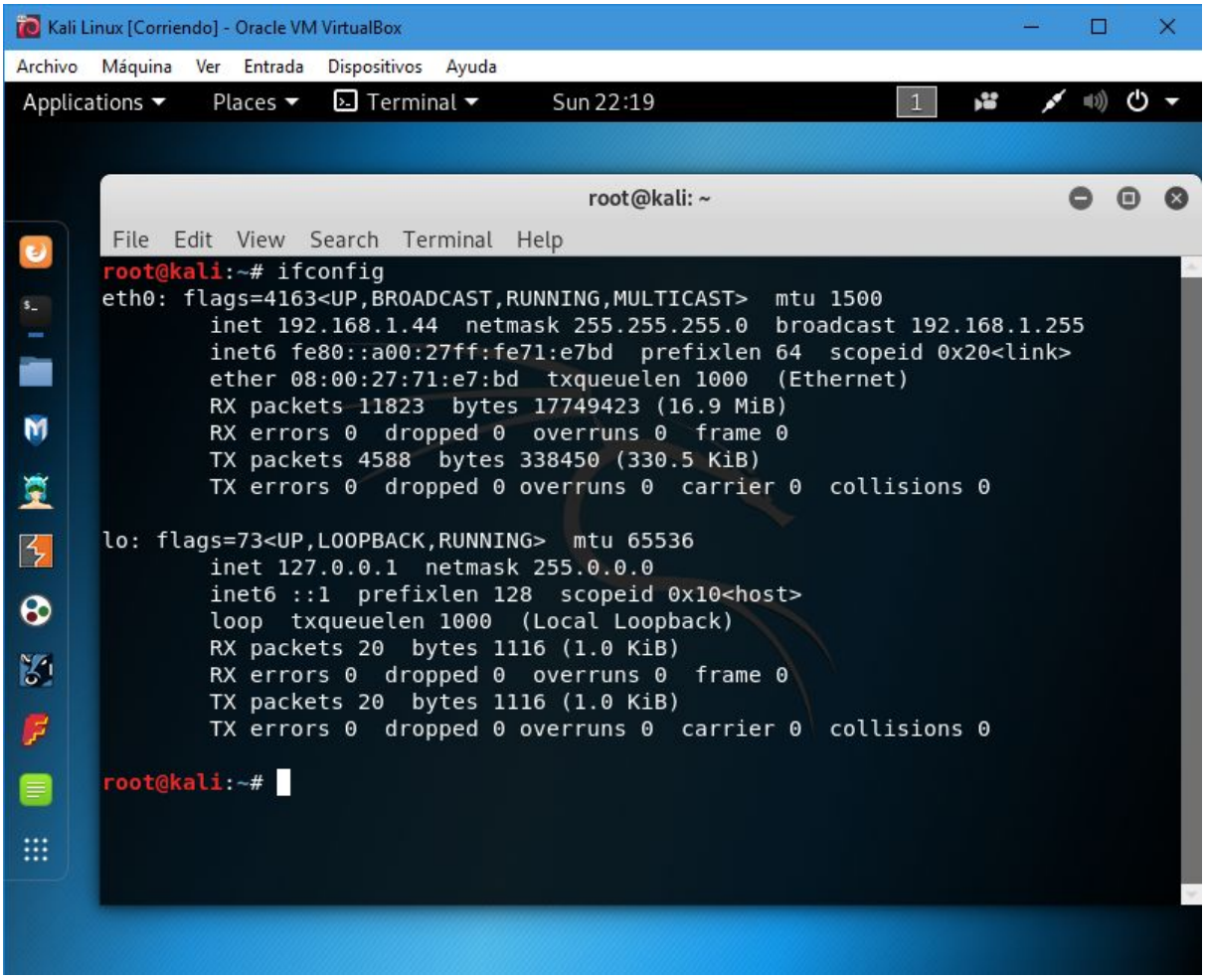


Ilustración 04 - Ejecución de la máquina virtual de Kali Linux

4.1.2. Pentesting

Para empezar la parte práctica y empezar a buscar vulnerabilidades dentro de esta web app, vamos a dar el primer paso con la parte de reconocimiento. En la máquina virtual de Kali vamos a utilizar un comando en la consola para escanear la red interna y ver qué es lo que hay conectado. Esta utilidad en concreto es "nmap", con ella podemos encontrar dispositivos conectados a la red, puertos abiertos y un largo etcétera.

Dentro de todos los muchos parámetros que podemos utilizar con esta utilidad vamos a utilizar "-sS" el cual nos permitirá ver distinta información como los puertos abiertos y demás. Cabe decir que esto lo estamos haciendo con una red privada y personal, en caso de hacer el reconocimiento en otra red es de buena práctica utilizar "-sP" primero, ya que este parámetro hace un sondeo ping y sólamente determina si los dispositivos están vivos o no. Seguidamente se haría un nmap en concreto para el o los objetivos en concreto. Otra opción interesante es usar "-p" para sondear los puertos indicados.

Comando: nmap -sP 192.168.1.0/24

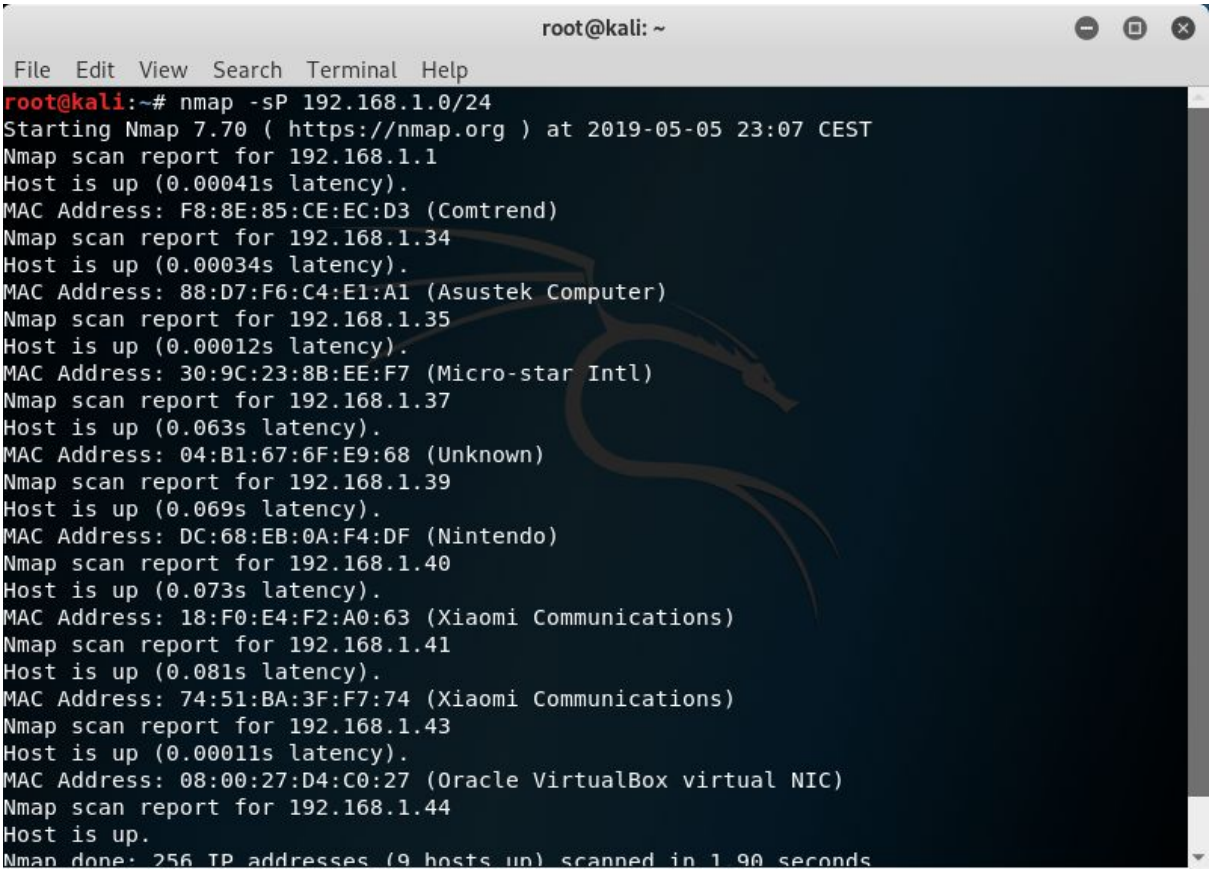


Ilustración 05 - Escaneo de la red con nmap

Se ha ejecutado el comando de nmap para toda la red 192.168.1.0 y hemos encontrado que entre todos los dispositivos conectados nuestro objetivo es la dirección 192.168.1.43. Seguidamente hacemos otro escaneo pero esta vez solamente de la dirección objetivo y cambiamos el parámetro por "-sS" ya que nos dirá qué puertos están abiertos. *Comando: nmap -sS 192.168.1.43*

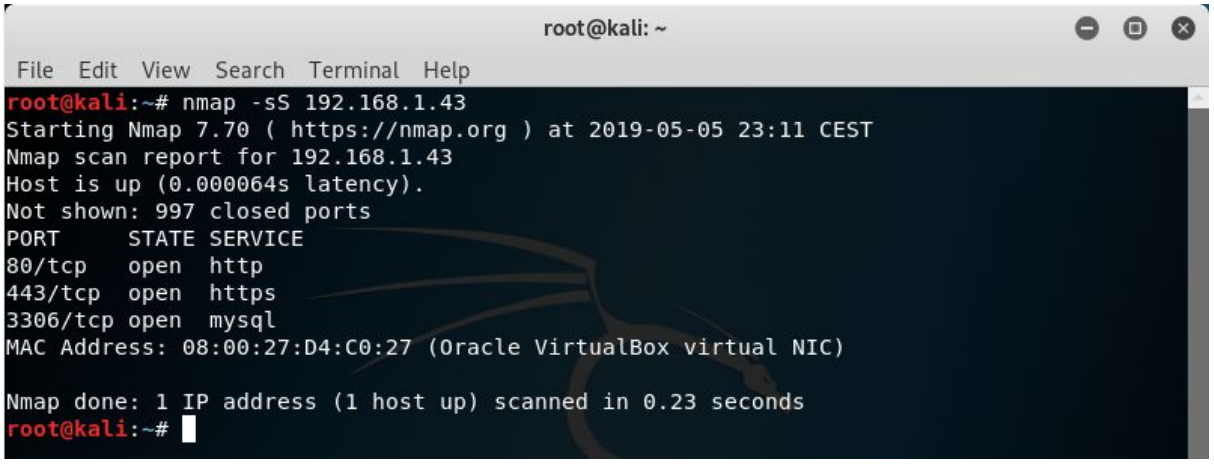


Ilustración 06 - Escaneo de puertos con nmap al objetivo BadStore

Podemos ver que la dirección objetivo tiene abiertos los puertos de http, https y mysql. A continuación vamos a hacer otro escaneo pero esta vez sólamente para los puertos en

concreto que queremos (menos ruido) y cambiando el parámetro a “-sV” el cual nos sirve para la detección de versiones. También agregamos una opción “-O” la cual nos devolverá el sistema operativo de la máquina. *Comando: nmap -sV -O -p 80,443,3306 192.168.1.43*

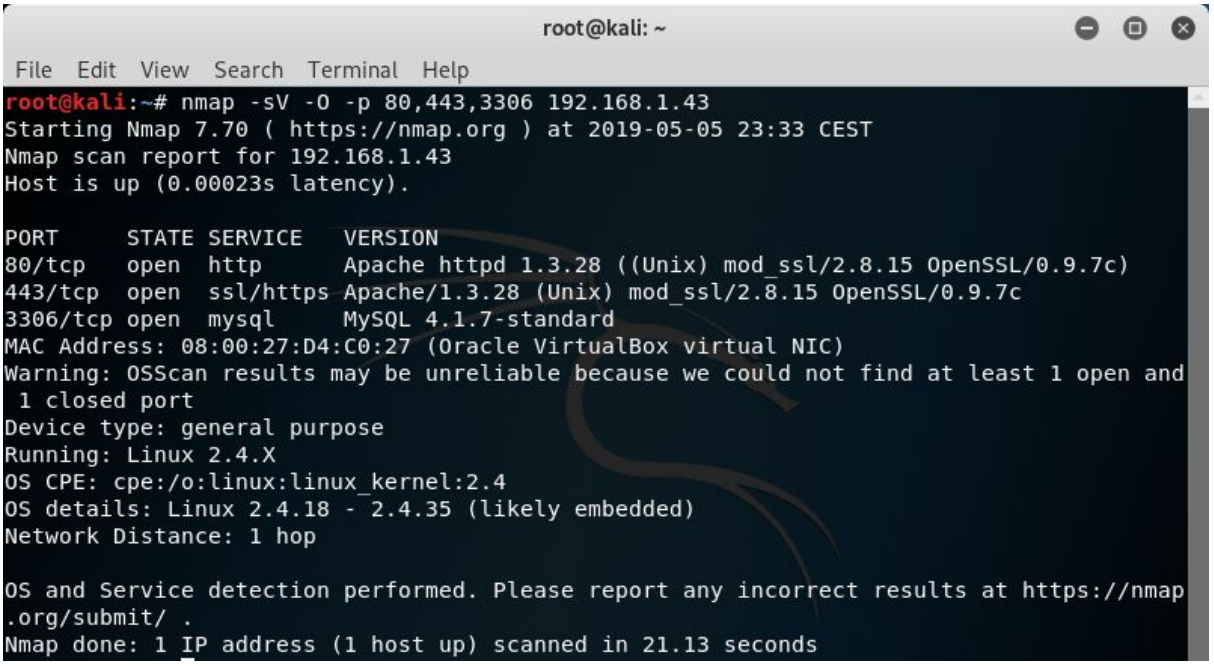


Ilustración 07 - Escaneo de puertos del objetivo con versión y SO

Por ahora podemos ver que el objetivo tiene un sistema operativo Linux, un Apache 1.3 con los puertos 80 y 443 de TCP abiertos y un servidor MySql en el puerto TCP 3306. Una vez tenemos esta información es hora de entrar a la web y testear.

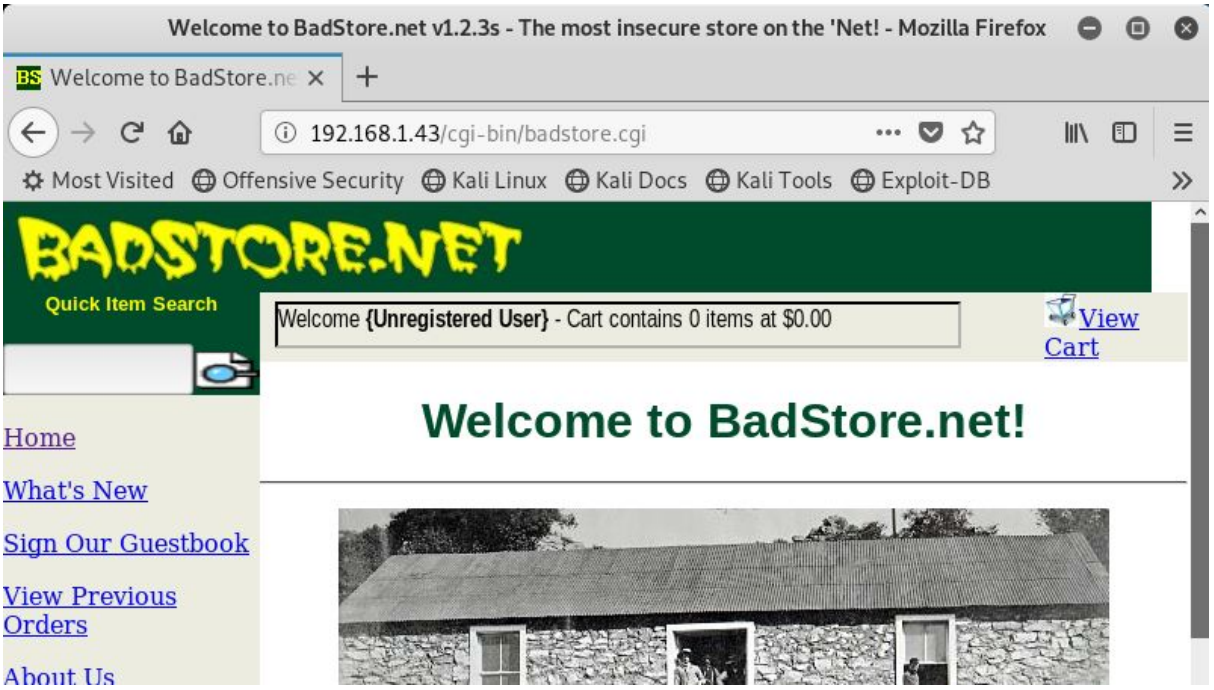


Ilustración 08 - Vista general de la web objetivo

Una de las cosas que vemos al entrar en “Home” es que la página está contenida en un directorio llamado “cgi-bin”. Viendo esto podemos hacer una búsqueda más exhaustiva de los directorios que se compone la aplicación web en el servidor. Para ello utilizamos una herramienta llamada “DirBuster”, en la cual podemos hacer un escaneo de diferentes directorios y archivos contenidos en una dirección destino mediante fuerza bruta.

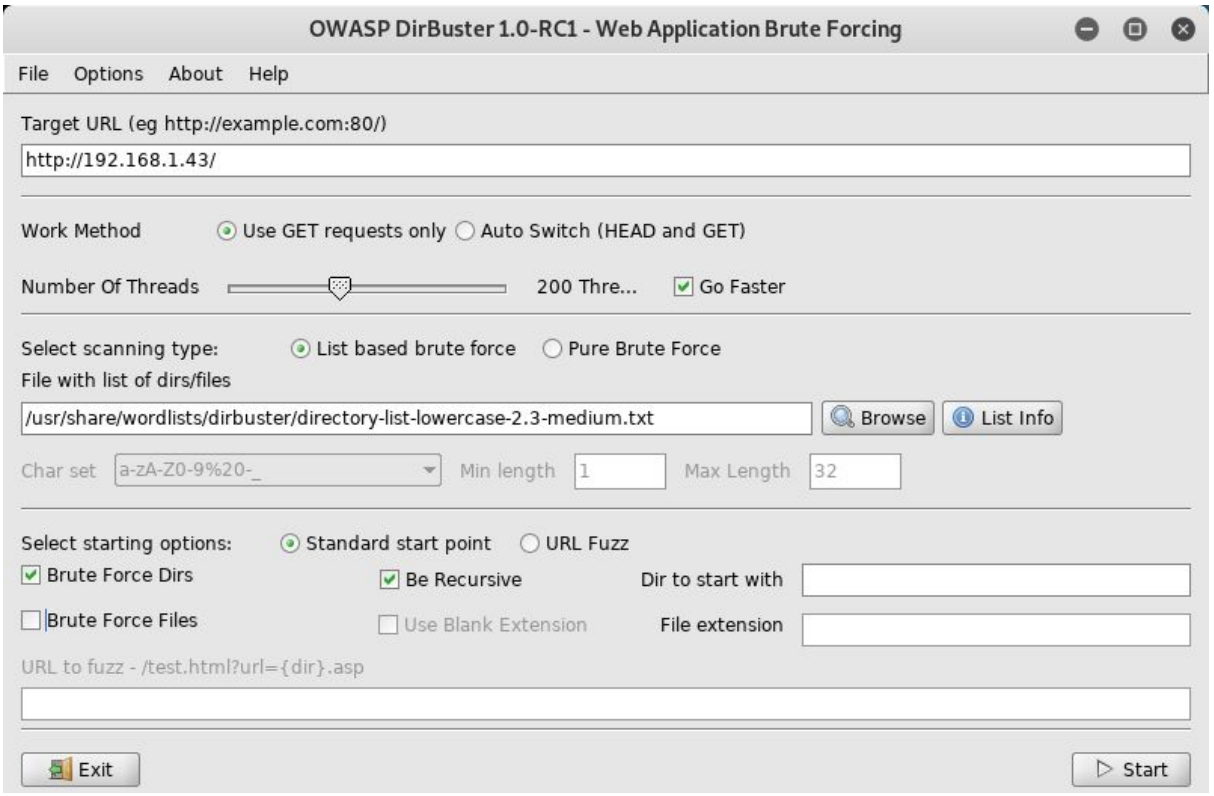
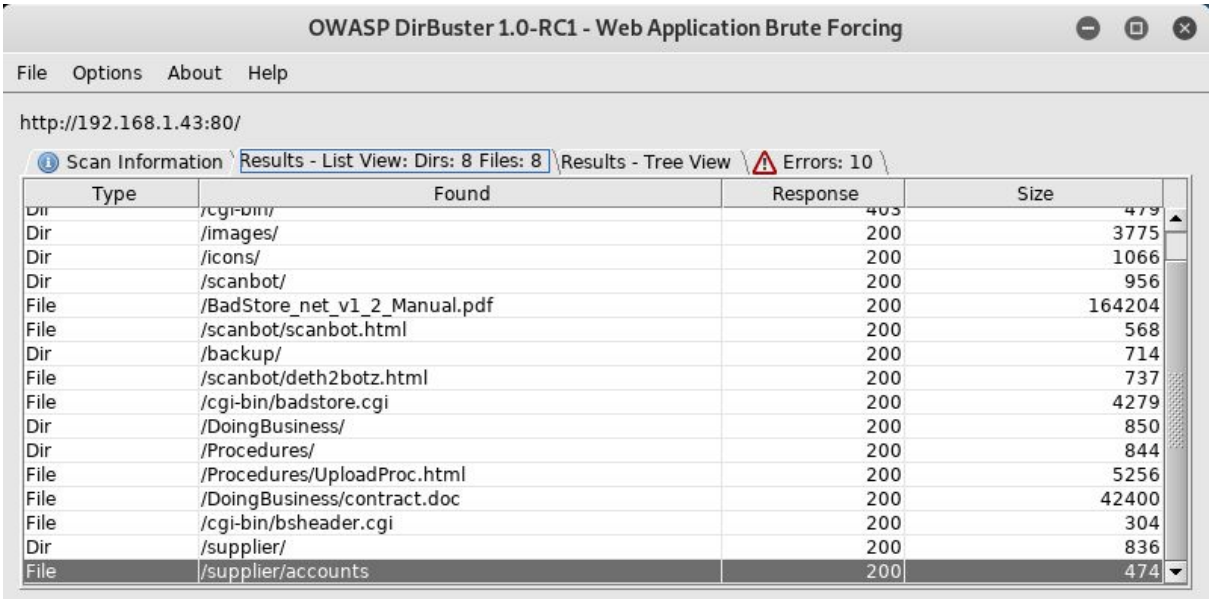


Ilustración 09 - Configuración de DirBuster para buscar directorios con una wordlist

Vemos en la primera ilustración (la cual es la configuración usada) que estamos buscando directorios en la dirección objetivo del servidor. Para ello estamos utilizando fuerza bruta con la ayuda de una wordlist general de nombres de directorios. En Kali Linux podemos encontrar un gran número de wordlists diseñadas para ayudarnos en estos casos. Podemos observar que todas ellas están localizadas en el directorio “/usr/share/wordlists” y hay de todo tipo. En este caso se utiliza una especialmente diseñada para esta aplicación la cual son todos los nombres en minúsculas.

Cabe destacar que utilizar fuerza bruta sin la ayuda de una wordlist de directorios (o archivos) podría llevar muchísimo tiempo. Haciendo la prueba sin introducir una wordlist con un tamaño de hasta 32 caracteres de string nos llevaría un tiempo infinito. Utilizando la wordlist mencionada anteriormente, nos ha llevado una totalidad de una hora y media aproximadamente.

Una vez completado todo el escaneo pasamos a ver los resultados.



Type	Found	Response	Size
Dir	/cgi-bin/	403	479
Dir	/images/	200	3775
Dir	/icons/	200	1066
Dir	/scanbot/	200	956
File	/BadStore_net_v1_2_Manual.pdf	200	164204
File	/scanbot/scanbot.html	200	568
Dir	/backup/	200	714
File	/scanbot/deth2botz.html	200	737
File	/cgi-bin/badstore.cgi	200	4279
Dir	/DoingBusiness/	200	850
Dir	/Procedures/	200	844
File	/Procedures/UploadProc.html	200	5256
File	/DoingBusiness/contract.doc	200	42400
File	/cgi-bin/bsheader.cgi	200	304
Dir	/supplier/	200	836
File	/supplier/accounts	200	474

Ilustración 10 - Resultados de la búsqueda de DirBuster

Podemos observar cómo hemos encontrado varios directorios y archivos. Entrando a cada uno de ellos para testear que pueden ser y si tenemos acceso como tal a ellos podemos acceder sin problema a prácticamente todos los directorios y archivos en estos. Por ejemplo, en el directorio /DoingBusiness/ encontramos un archivo “contract.doc” el cual no tiene prácticamente interés, pero eso significa una vulnerabilidad grave, pudiendo entrar a todo el contenido.

Si entramos en el último directorio encontrado “/supplier/” vemos que tiene otro subdirectorio el cual tiene un nombre muy interesante, “/accounts”. Esto es signo de un gran interés y vale la pena explorarlo en su totalidad. Entrando al contenido de este directorio podemos ver lo siguiente:

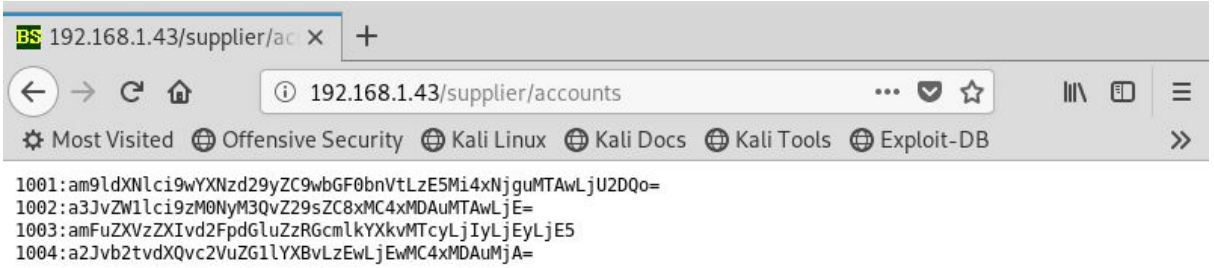


Ilustración 11 - Contenido del archivo accounts

Deducimos por el estilo de este texto que se trata de distintos usuarios de la aplicación web con un texto encriptado en algún formato. Vamos a intentar descodificar los datos codificados y para ello utilizamos un programa llamado “Decodify” el cual nos servirá para detectar de qué tipo es. Una vez descargado el archivo de Git (Enlace disponible en el Glosario) podemos ejecutar el programa para un string en concreto, probamos el primero del usuario con el ID 1001:

Ilustración 12 - Ejecución de Decodify para un string en Base64

Podemos observar que el string estaba codificado en Base64 y efectivamente nos da los datos, los cuales son las credenciales de usuario. Ejecutando para el resto de strings tenemos el archivo encontrado de cuentas de los suppliers con los usuarios y contraseñas.

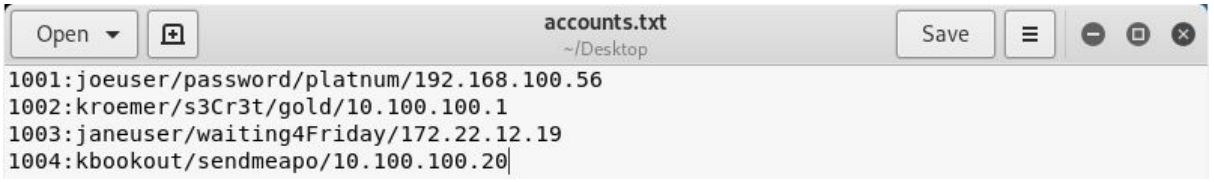


Ilustración XX - Archivo con credenciales de los suppliers

Ahora que hemos visto la estructura de los directorios y buscado por posibles archivos ocultos dentro de estos vamos a pasar a testear el funcionamiento de la web. Viendo que hay un apartado de registro y login podemos empezar a testear por ahí. Vamos a crear una nueva cuenta llamada “jaume” y ver que parámetros pasa por la URL.

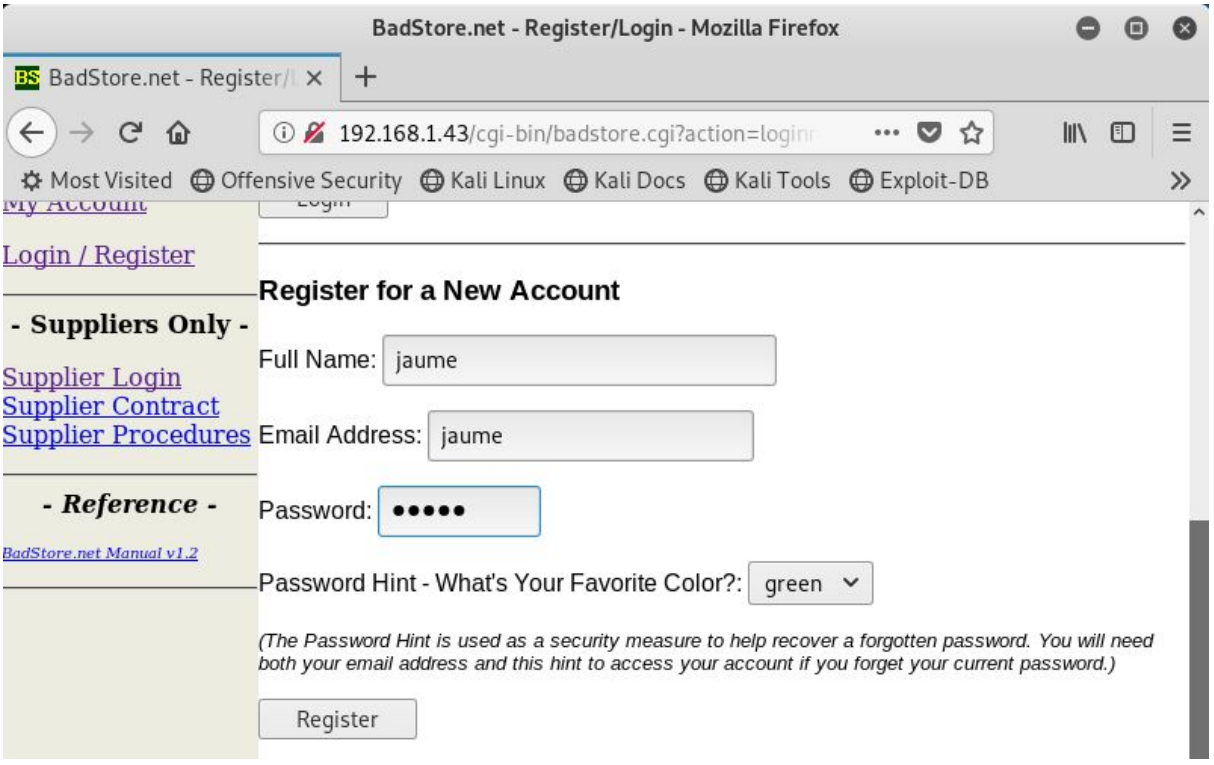


Ilustración 13 - Estructura de la página de registro

Una de las cosas que podemos apreciar es que en la URL de la página de login/registro después de “badstore.cgi” aparece un “action” el cual nos redirige a la web en concreto. Vamos a abrir el monitor de red de Firefox pulsando F12, clicando en “Network” y seguidamente damos a “Analyze” para que analice las peticiones que entran a partir de ahora.

Ahora procedemos a registrar el usuario anterior clicando al botón de “Register”. Podemos ver en el monitor de red los datos que hemos capturado. Vemos que se ha hecho una petición POST la cual si clicamos en ella podemos ver toda la información, como headers, cookies, etc. Vamos a ir a “Params” para ver que se está enviando.

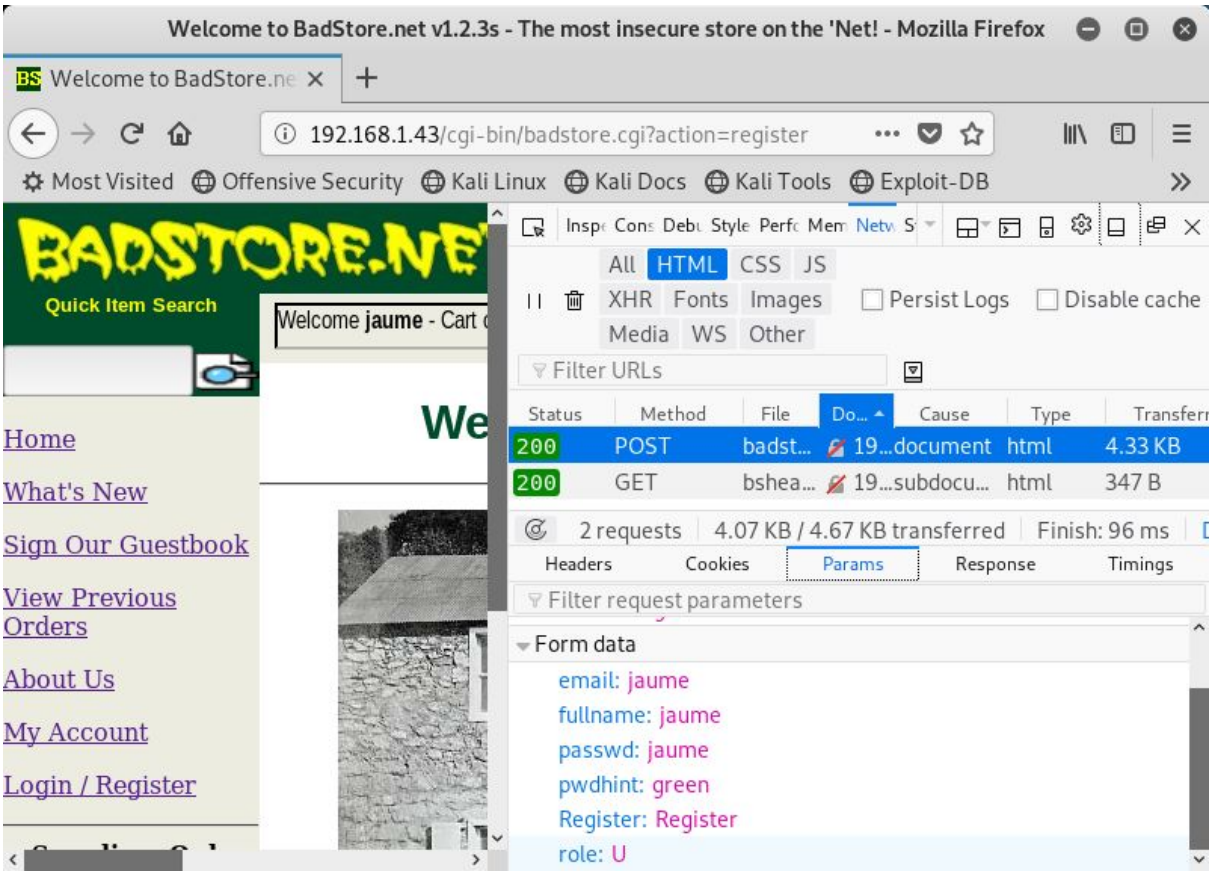


Ilustración 14 - Parámetros de la petición de registro

Podemos ver los campos que utiliza la URL para enviar los datos de creación de usuario. Prestamos atención al último de ellos, que es “role: U”. Esto es interesante, podemos deducir que al crear un usuario le asigna por parámetro el rol concreto en la web. Probamos en crear un nuevo usuario mediante la URL utilizando estos parámetros que hemos encontrado.

URL:
http://192.168.1.43/cgi-bin/badstore.cgi?action=register&email=juan&fullname=juan&passwd=juan&pwdhint=green&Register=Register&role=U

Introduciendo los parámetros encontrados y utilizando el carácter “&” efectivamente nos damos cuenta que podemos crear usuarios correctamente sin tener que usar el formulario. Incluso podemos crear usuarios con permiso de administrador simplemente cambiando el parámetro “role=U” por “role=A”.

Vamos a ver otro apartado de la web. Si entramos como usuarios sin registrar y vamos a “My Account” podemos ver una página la cual nos permite resetear la contraseña de un usuario. El problema que hay es que solo nos pide la cuenta de usuario (email) y un “password hint” el cual solamente consta de seleccionar una de 6 opciones.

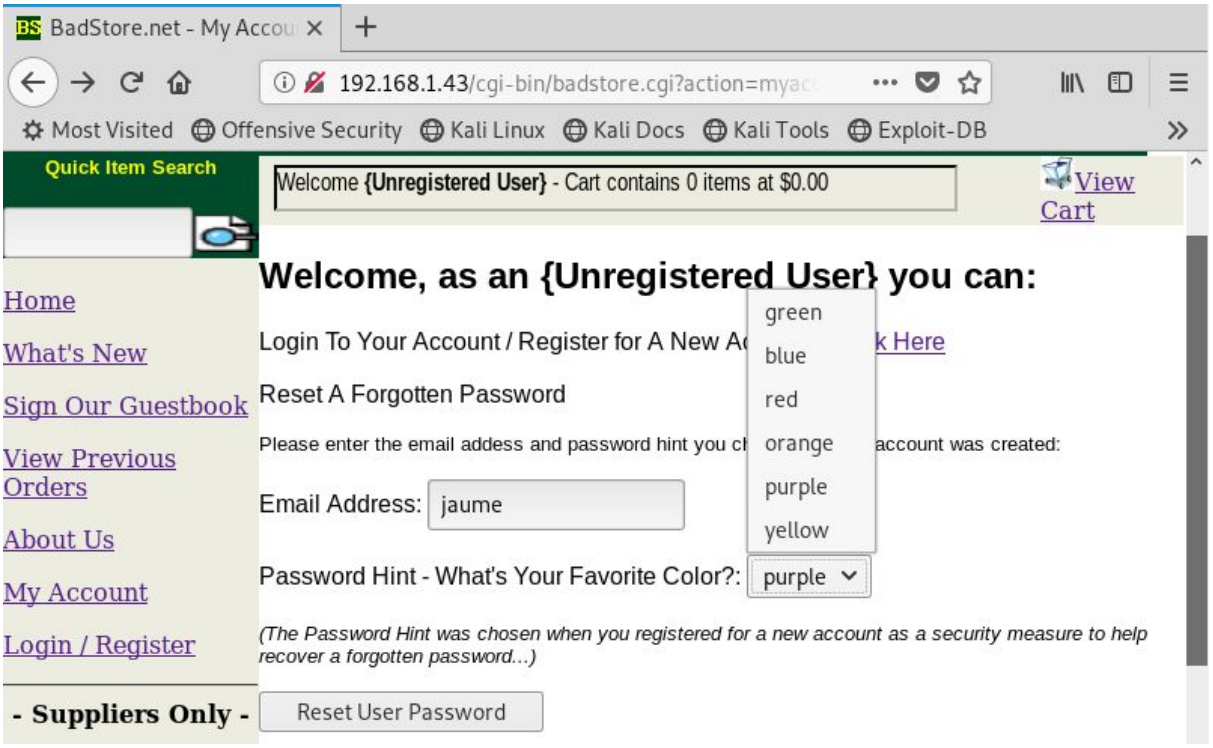


Ilustración 15 - Página de My Account para usuarios no registrados

Al hacer la prueba con la cuenta creada anteriormente e introduciendo un password hint que no es el correspondiente (cuando creamos la cuenta pusimos “green”) nos resetea la contraseña igualmente, por lo tanto aquí tenemos un grave problema. La opción de “doble seguridad” del password hint no funciona absolutamente para nada.

Una vez reseteamos la contraseña vemos un problema aún más grave. La contraseña reseteada que introduce por defecto es la misma siempre: “Welcome”. Esto se resume que podemos cambiar la contraseña de quien nosotros queramos, siempre que sepamos el usuario.

Ilustración 16 - Resultado de resetear la contraseña

Vamos a hacer una prueba a ver si podemos cambiar la contraseña de una cuenta con privilegios. Después de hacer varias pruebas he visto que el usuario principal es “admin” y efectivamente hemos podido resetear la contraseña de este por la que pone por defecto “Welcome”.

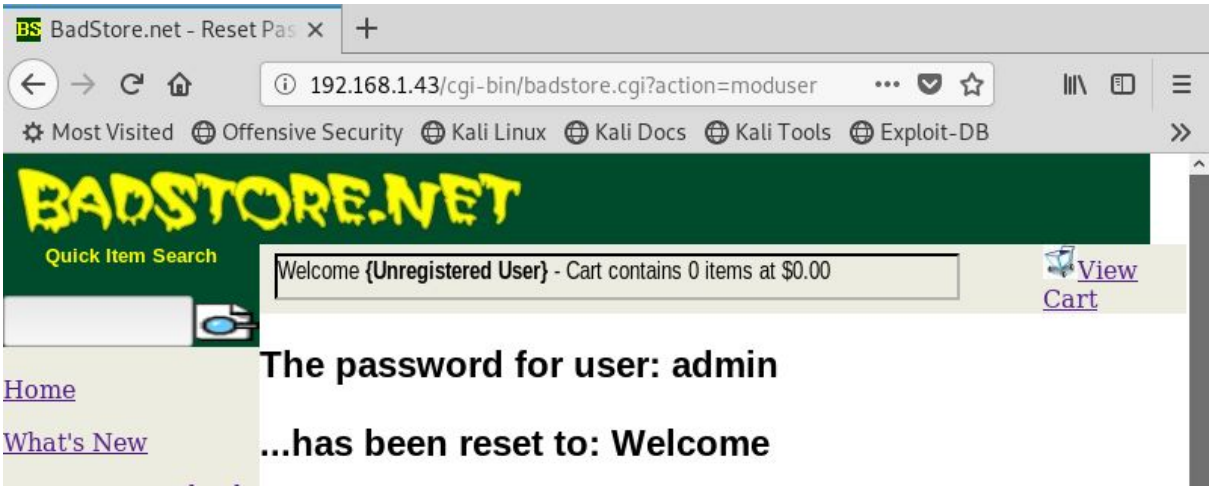


Ilustración 17 - Reset de contraseña para el usuario admin

Ahora podemos entrar al portal como superuser y entrar al panel de administrador el cual podemos visitar cambiando la URL por “action=admin”.

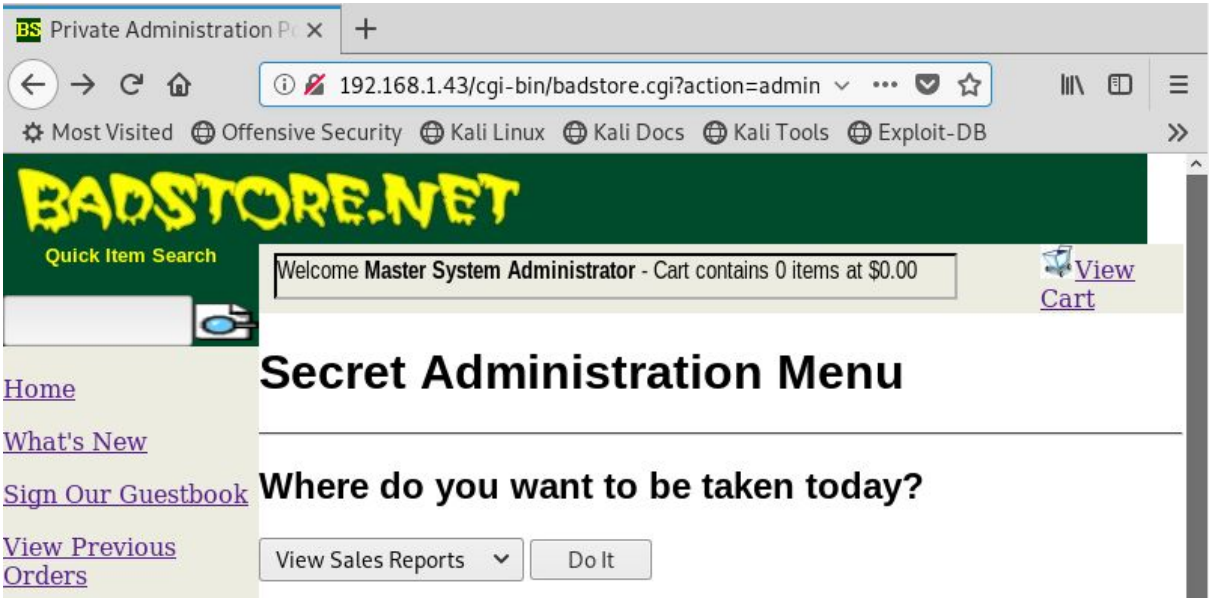
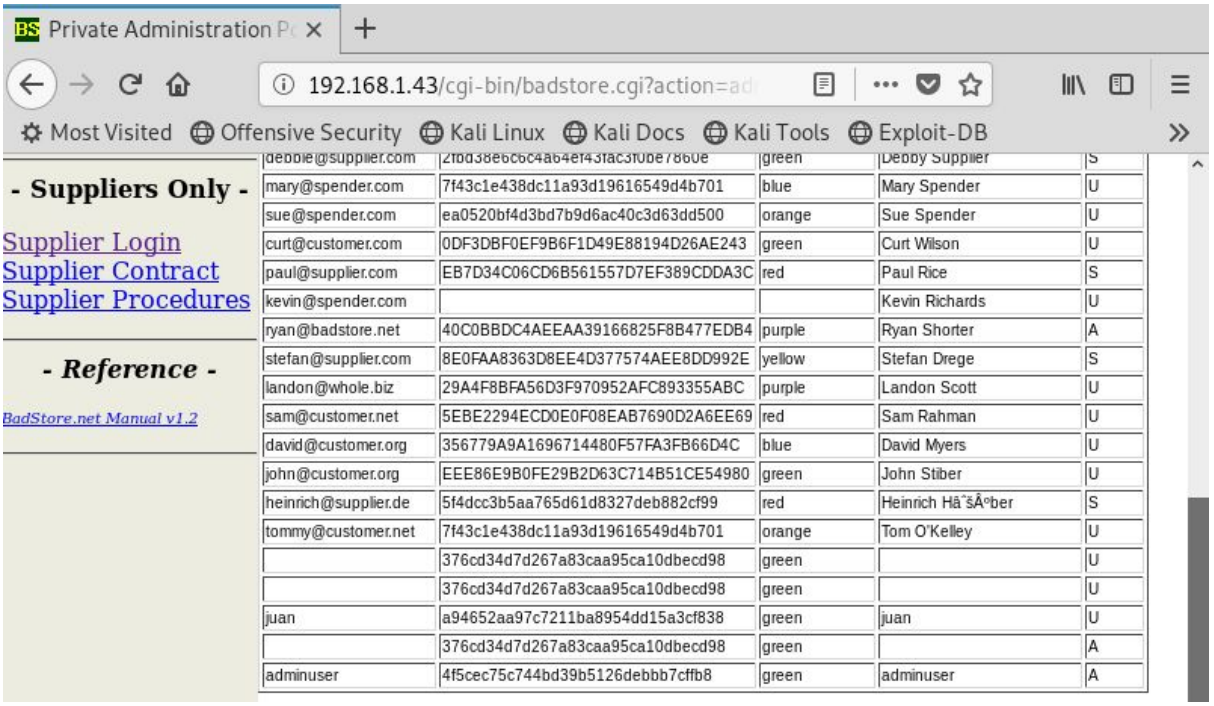


Ilustración 18 - Panel de control como admin

Ahora podemos entrar a la página donde se muestran todos los usuarios con sus contraseñas codificadas (las cuales podríamos decodificar sin ningún problema) y ver toda su información, así como las compras, reports, etc.



debbie@supplier.com	2fdd38e6c6c4ab64ef43fac3f0de7860e	green	Debby Supplier	S
mary@spender.com	7f43c1e438dc11a93d19616549d4b701	blue	Mary Spender	U
sue@spender.com	ea0520bf4d3bd7b9d6ac40c3d63dd500	orange	Sue Spender	U
curt@customer.com	0DF3DBF0EF9B6F1D49E88194D26AE243	green	Curt Wilson	U
paul@supplier.com	EB7D34C06CD6B561557D7EF389CDDA3C	red	Paul Rice	S
kevin@spender.com			Kevin Richards	U
ryan@badstore.net	40C0BBDC4AEEAA39166825F8B477EDB4	purple	Ryan Shorter	A
stefan@supplier.com	8E0FAA8363D8EE4D377574AEE8DD992E	yellow	Stefan Drege	S
landon@whole.biz	29A4F8BFA56D3F970952AFC893355ABC	purple	Landon Scott	U
sam@customer.net	5EBE2294ECD0E0F08EAB7690D2A6EE69	red	Sam Rahman	U
david@customer.org	356779A9A1696714480F57FA3FB66D4C	blue	David Myers	U
john@customer.org	EEE86E9B0FE29B2D63C714B51CE54980	green	John Stiber	U
heinrich@supplier.de	5f4dcc3b5aa765d61d8327deb882cf99	red	Heinrich HâˆšÂºber	S
tommy@customer.net	7f43c1e438dc11a93d19616549d4b701	orange	Tom O'Kelley	U
	376cd34d7d267a83caa95ca10dbecd98	green		U
	376cd34d7d267a83caa95ca10dbecd98	green		U
juan	a94652aa97c7211ba8954dd15a3cf838	green	juan	U
	376cd34d7d267a83caa95ca10dbecd98	green		A
adminuser	4f5cec75c744bd39b5126debbb7cffb8	green	adminuser	A

Ilustración 19 - Página de current users del panel de admin

Podemos observar el último usuario de la lista “adminuser”, éste lo hemos creado mediante la modificación de la URL en vez de utilizar el formulario de registro. Podemos comprobar que cambiando el parámetro de rol si que funciona el darle permisos a los usuarios creados y con este usuario también podríamos entrar a este menú de administradores.

Haciendo varias pruebas más también me he dado cuenta que cualquier usuario (aunque no sea administrador) puede entrar a la página principal del menú de admin.

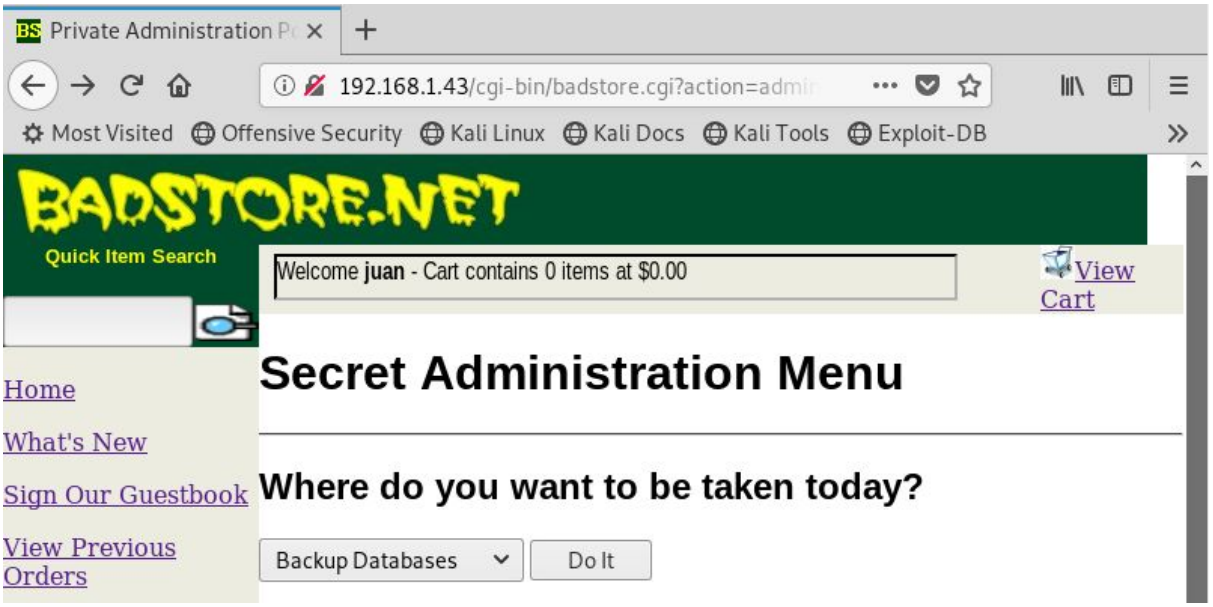


Ilustración 20 - Acceso al menú principal de admin como usuario

Vemos como estamos registrados como el usuario “juan” el cual no tiene permisos de administrador, pero puede acceder a este apartado igualmente. De aquí no deja pasar, ya que este usuario no tiene permisos de administrador, pero da una información crítica sobre cómo es el menú y qué operaciones es capaz de hacer el administrador.

Ilustración 21 - Error al intentar entrar como usuario no admin

Otro punto importante a testear es el tráfico que produce la página web y cómo trata los paquetes. Para observar esto utilizaremos Wireshark (más información sobre el programa en el glosario).

Ejecutamos el programa y seleccionamos sobre qué interfaz de red queremos trabajar, en mi caso selecciono “eth0” que es la que tiene conexión con el resto de la red. Una vez dentro de la interfaz como tal, vamos a poner un filtro para desechar el resto de paquetes de la red que no nos interesan. El filtro como tal es: “ip.src == 192.168.1.44” ya que queremos capturar el tráfico que enviamos. Podemos empezar la búsqueda y seguidamente hacemos un login en la página web con un usuario.

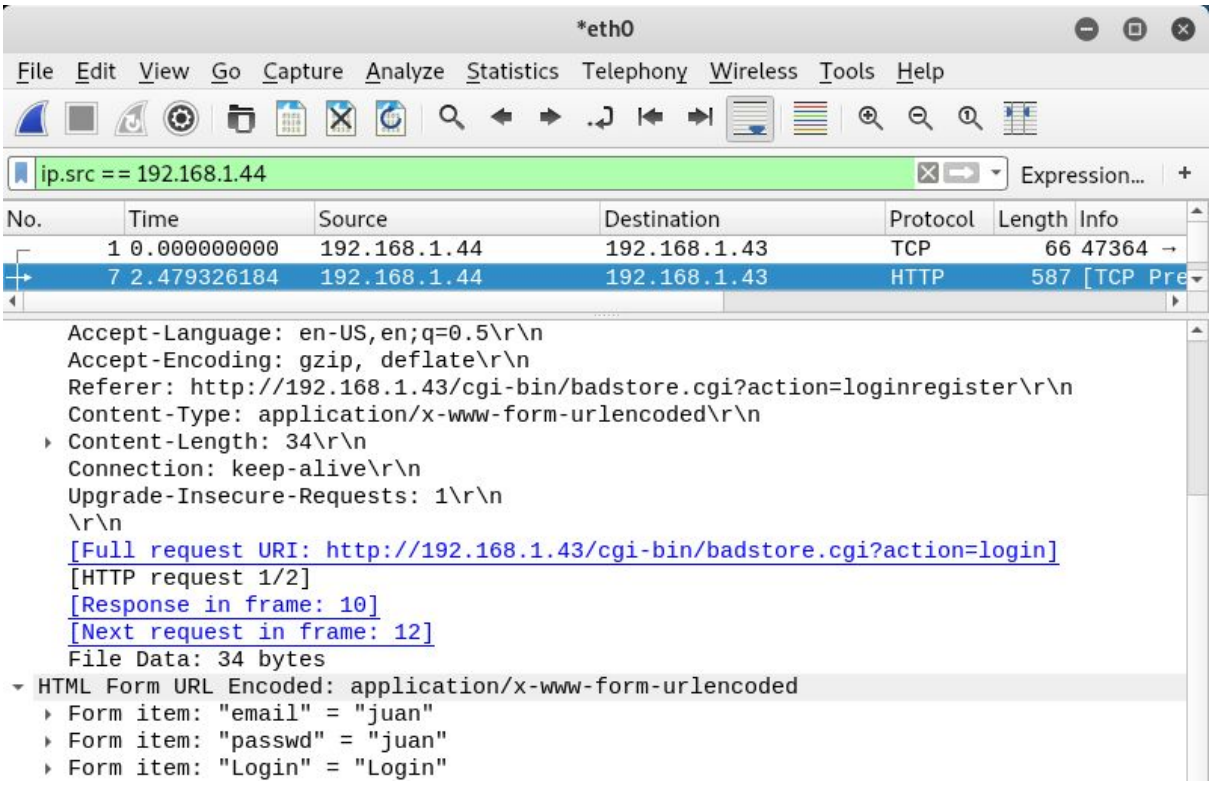


Ilustración 22 - Captura de paquetes con Wireshark

Vemos un paquete HTTP el cual tiene destino la IP de BadStore, lo abrimos y vemos algo interesante en la parte de abajo del todo del paquete. Podemos ver la información del HTML Form la cual no está codificada, está como texto plano. Esto es signo de que el tráfico no está encriptado del todo y cualquiera que esté esnifando el tráfico de la red puede obtener los credenciales de la gente que quiera registrarse.

Vamos a seguir con el testeo de la página de login, ya que tiene campos para introducir texto y eso siempre es un foco de atención cuando se están buscando vulnerabilidades. Anteriormente en el apartado de reconocimiento vimos que la aplicación web tiene una base de datos MySQL y el puerto abierto. Además, en los forms para hacer login o registro siempre hacen una conexión a una base de datos para comprobar y guardar la información de los usuarios. Al saber que dispone de un MySQL podemos intentar explotar esta parte.

Vamos a pasar un texto entre comillas simples simulando un campo de la base de datos y ver qué ocurre. Texto introducido: ‘asdasdasdasd’.

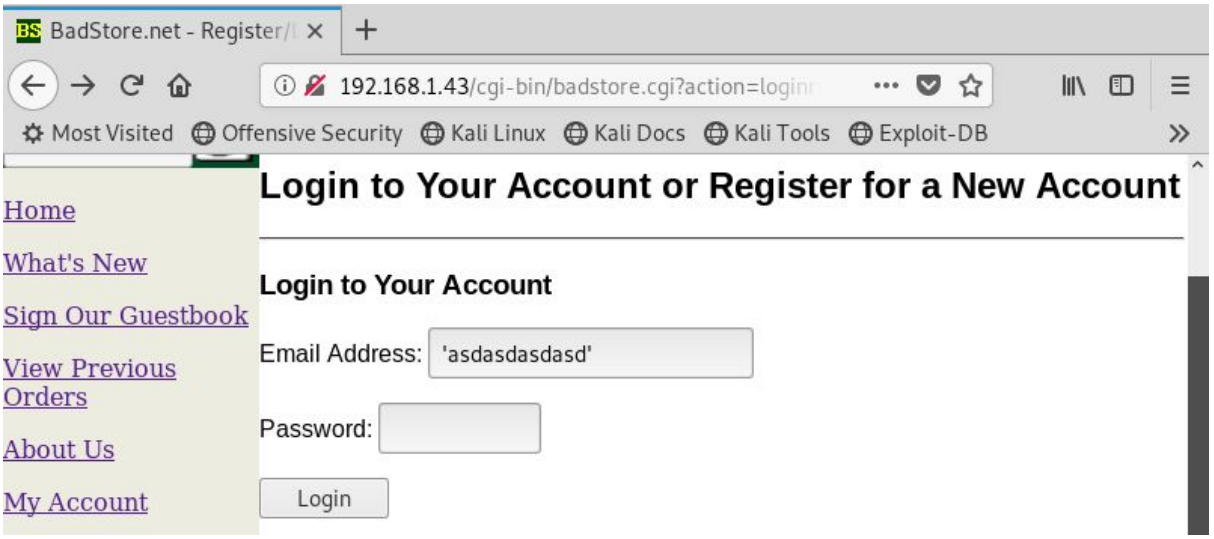


Ilustración 23 - Introducción de una consulta SQL

Ilustración 24 - Error de búsqueda SQL

Con el error que recibimos podemos ver que hay una vulnerabilidad clara de SQLi, ya que el formulario nos permite introducir consultas sin validación alguna.

Cuando testeamos inyección SQL se pueden utilizar herramientas automáticas las cuales introducen muchas consultas seguidas para ver si conseguimos la información deseada, esto automatiza el proceso. También hay distintas listas de consultas creadas por ciertas personas con varias formas de saltarse el acceso. Vamos a probar una de las más típicas.

Código: admin’ or ‘1’=’1

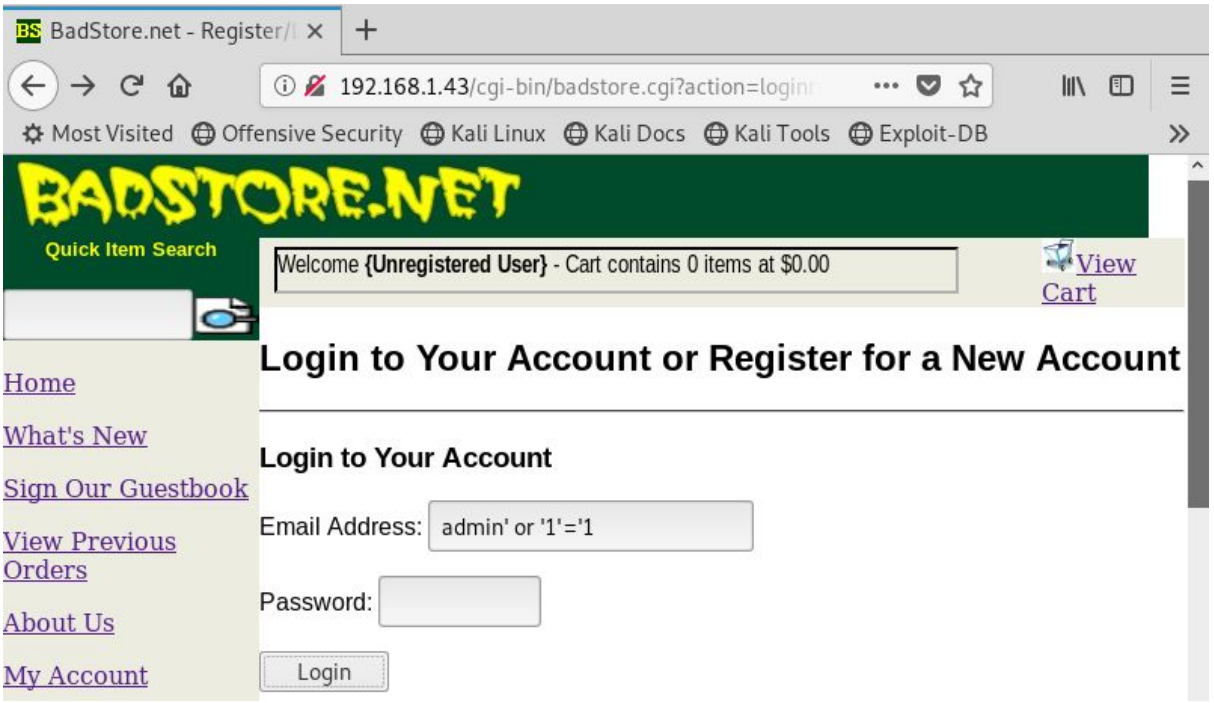


Ilustración 25 - Introducción de SQLi en el login

Y efectivamente nos hemos saltado la comprobación de credenciales y hemos entrado como administrador del sistema sin ni siquiera introducir la contraseña.

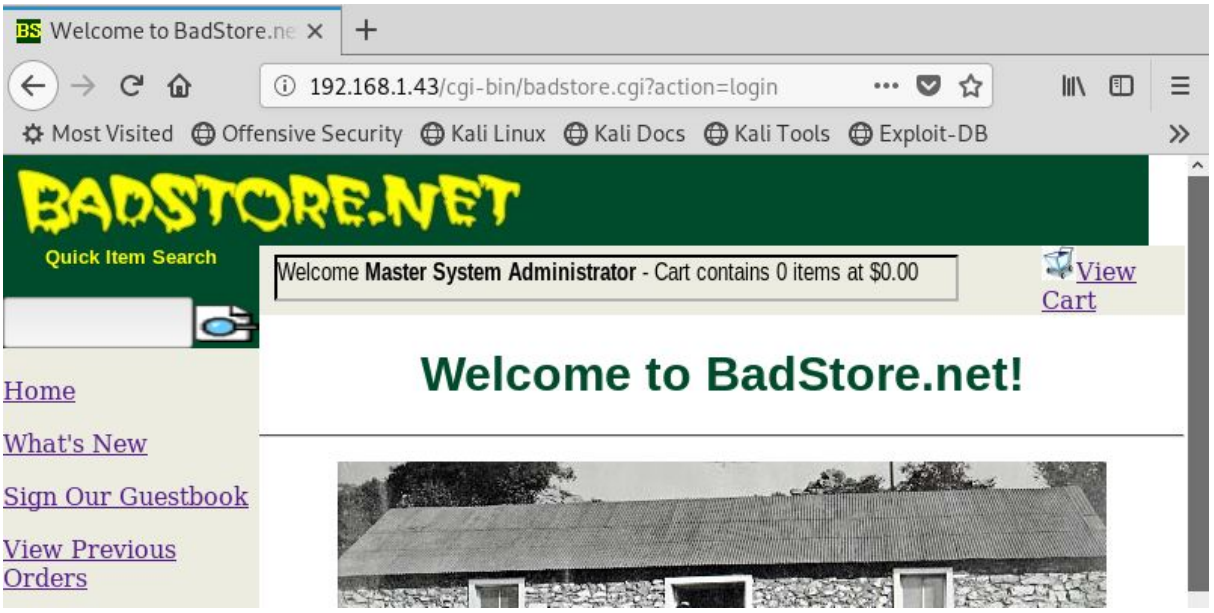


Ilustración 26 - Entrada por SQLi como admin

Igualmente vamos a hacer una sencilla prueba con la base de datos MySQL del servidor. Consiste simplemente en intentar entrar con usuario y contraseña. Podríamos utilizar un programa de fuerza bruta para intentar averiguar el usuario y contraseña de la base de datos en cuestión como por ejemplo Metasploit mysql_login. Viendo la seguridad general del sitio podemos hacer la prueba con el usuario root típico.

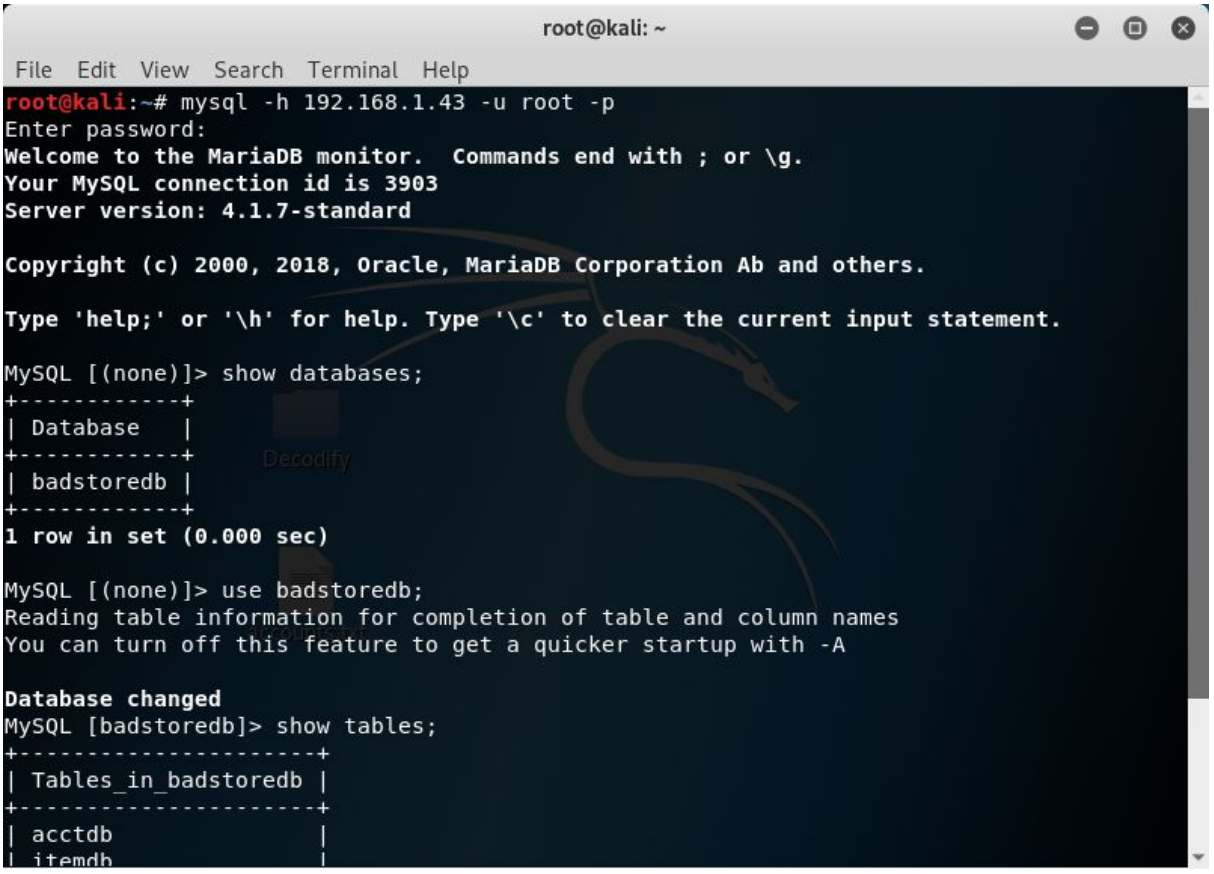


Ilustración 27 - Conexión de la base de datos de BadStore

Bingo, hemos conectado perfectamente con el usuario “root” y contraseña “root”. Para ello hemos utilizado los siguientes comandos:

Comando: mysql -h 192.168.1.43 -u root -p

Asimismo podemos ver toda la información de la base de datos de BadStore, una brecha de seguridad enorme y crítica.

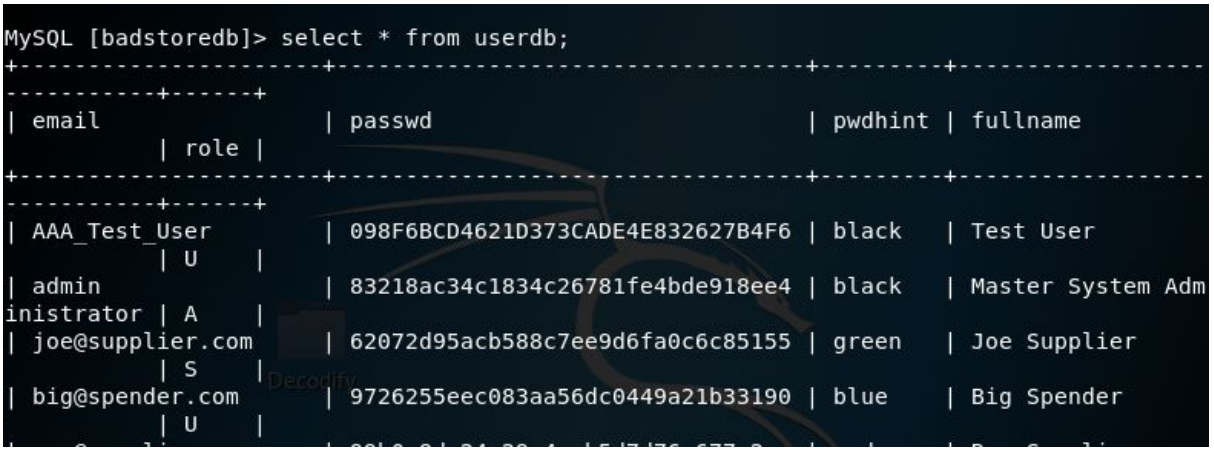


Ilustración 28 - Ejemplo de visualización de la tabla de usuarios

Vamos a seguir analizando la página web en búsqueda de otras vulnerabilidades.

Esta vez entramos en la página del guestbook, donde los clientes firman o ponen algún comentario relacionado con la página. Una vez dentro vemos que consiste en un formulario con varios campos de nombre, mail y el comentario como tal.

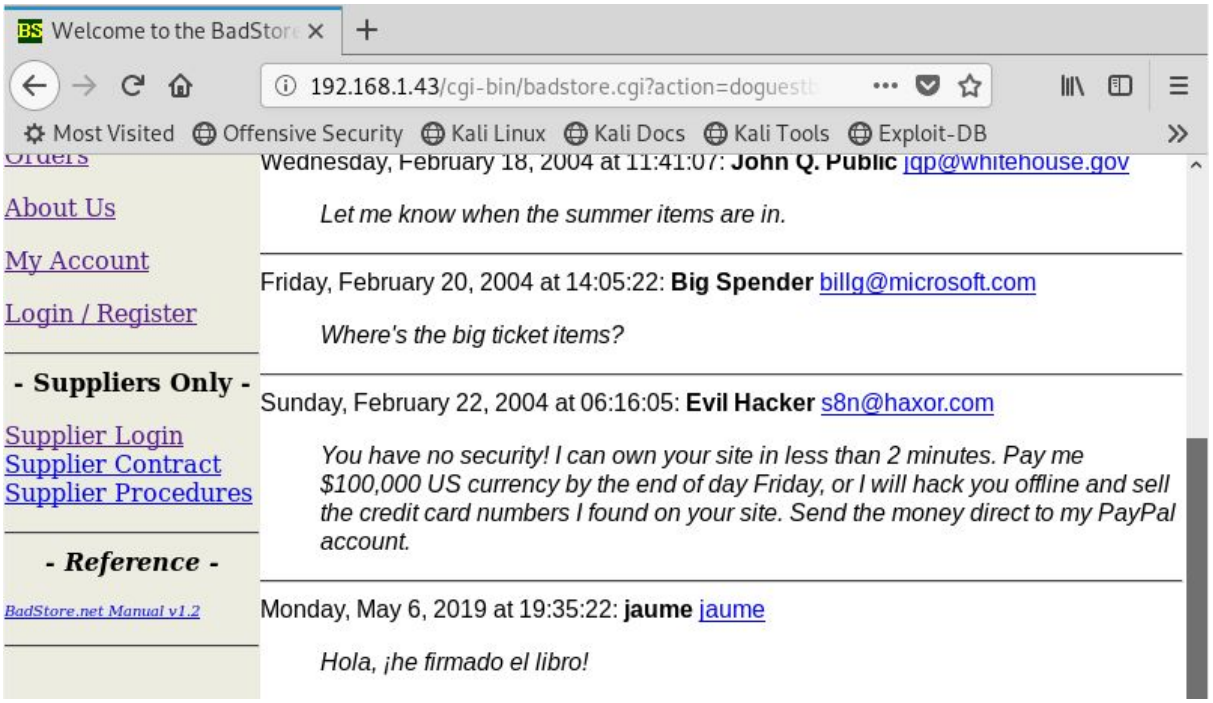


Ilustración 29 - Prueba de firma del guestbook

Hacemos la prueba e introducimos un comentario como cualquier otro de los que existen. Cabe destacar que a la hora de introducir el correo electrónico que nos piden, no hace ninguna verificación. Ahora vamos a jugar con los campos y a probar de introducir código.

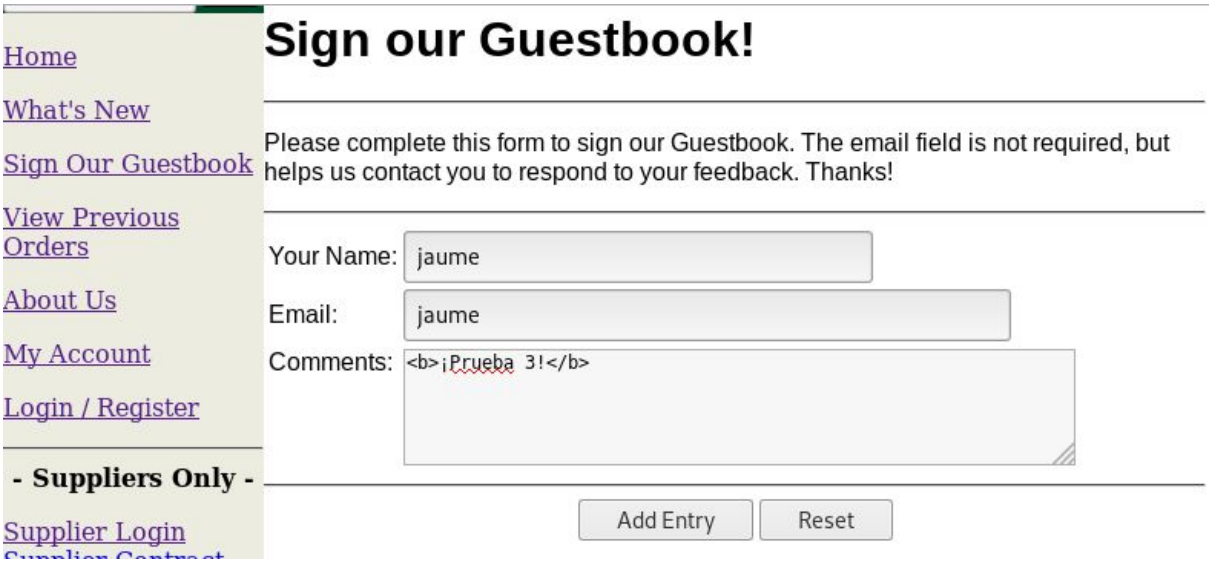


Ilustración 30 - Inserción de código HTML en el guestbook

Ilustración 31 - Resultado de la inserción de código HTML en el guestbook

Vemos que nos ejecuta el código HTML sin problemas ya que nuestro texto ha salido en negrita. Estamos ante una vulnerabilidad XSS y bastante importante. Vamos a probar ahora a ver si podemos ejecutar código JavaScript.

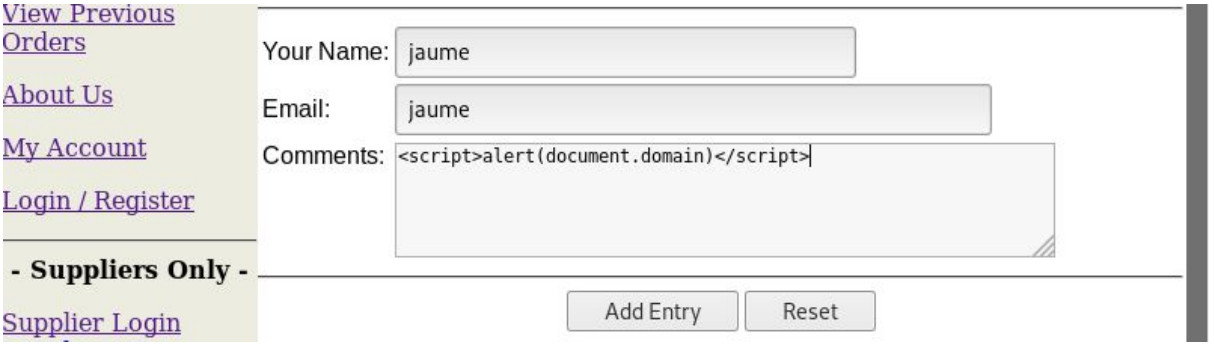


Ilustración 32 - Inserción de código JavaScript en el guestbook

Al aceptar la introducción de código HTML es muy probable que podamos ejecutar código JavaScript en este formulario. Hacemos la prueba y añadimos un script que simplemente muestre una alerta con la dirección IP del dominio.

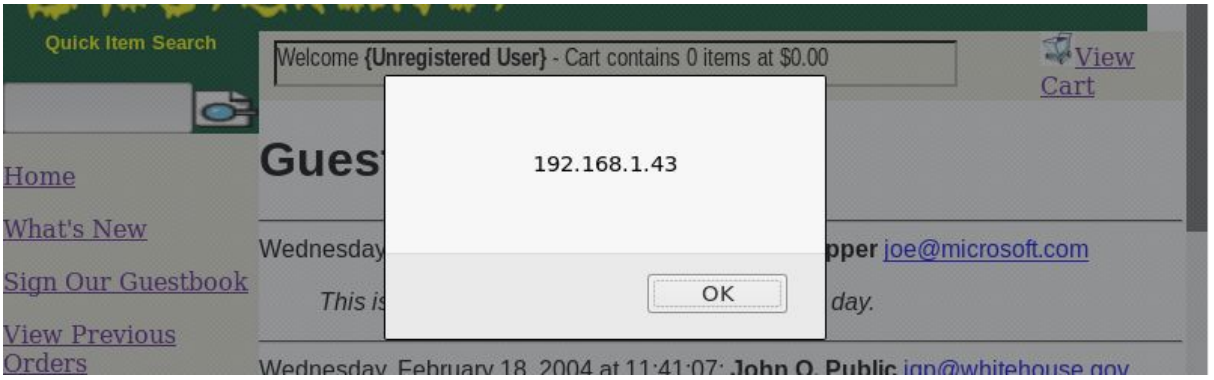


Ilustración 33 - Muestra del alert introducido por script

Pasamos a otro apartado de la web que se ve a primera vista, el buscador. Vamos a investigar un poco el funcionamiento de ésta, a ver si detectamos algo extraño. Ejecutamos una búsqueda vacía, sin introducir nada.

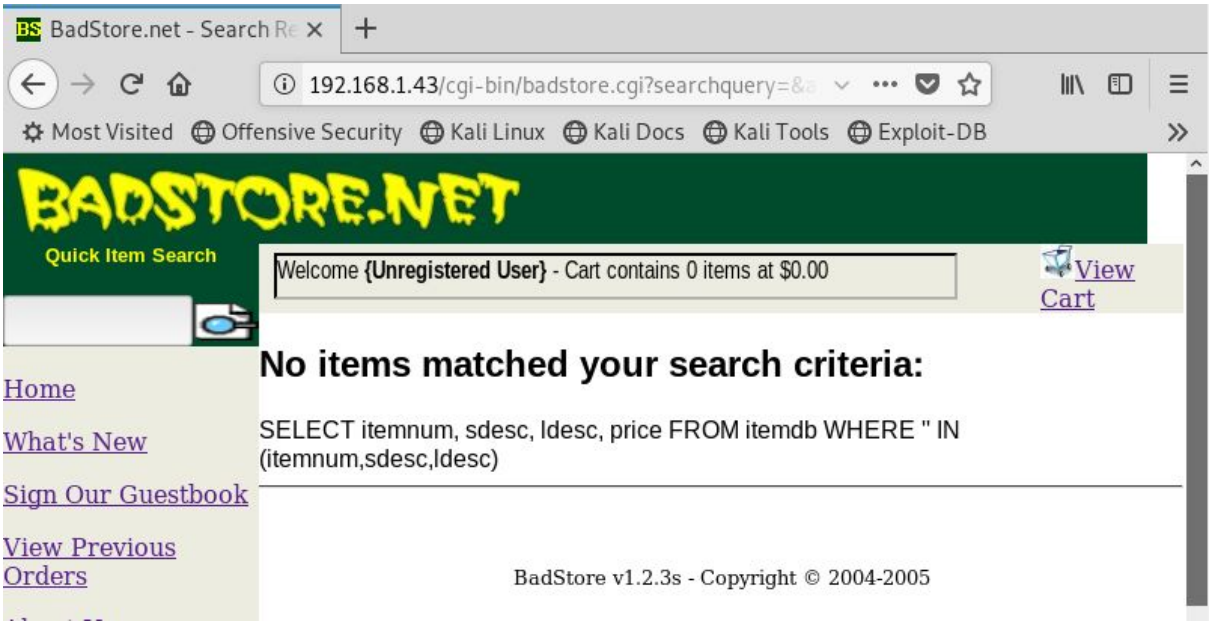


Ilustración 34 - Error de búsqueda vacía

Nos aparece un error de SQL el cual nos está mostrando la búsqueda que está haciendo dónde "WHERE ‘’" está vacío y por eso no encuentra nada. Esto nos da una pista de la vulnerabilidad que hay aquí. Por no hablar de que aparece la tabla "itemdb" y los campos. Nos encontramos ante otro caso de SQLi.

Vamos a intentar hacer una inyección de código para conseguir un archivo del directorio del servidor. Introducimos en el cuadro de búsqueda el siguiente código:

Código: ‘ UNION select null,null,null,load_file(‘/etc/passwd’)##

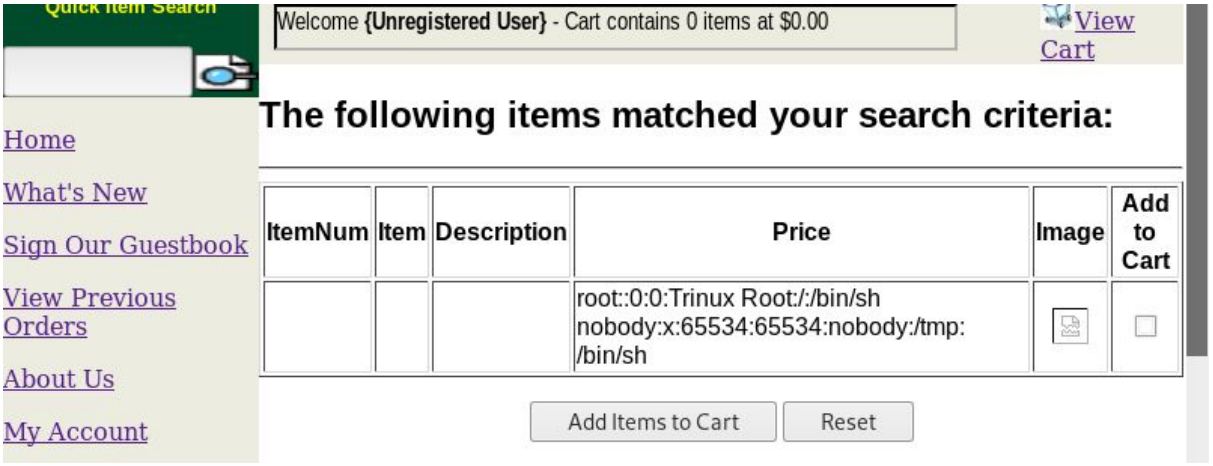


Ilustración 35 - Retorno del archivo passwd del servidor mediante SQLi

Con este comando estamos haciendo una búsqueda en SQL agregando otra búsqueda, dejando en blanco los tres primeros campos y cargando el contenido del archivo passwd en el campo siguiente. Con esto tenemos las credenciales de la base de datos de root y con ellos podríamos acceder y tener toda la información de todas las tablas y bases de datos. También podemos conseguir la versión de la base de datos con el siguiente comando:

Comando: ‘ *UNION select null,null,null,version()##*

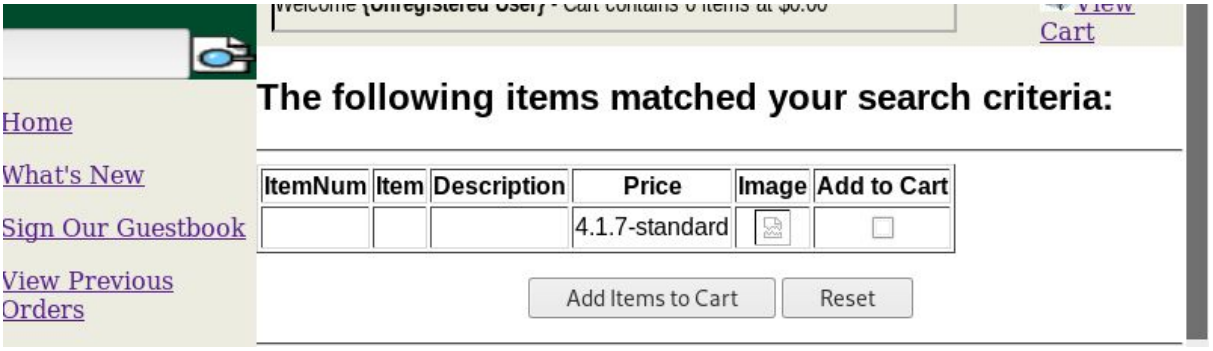


Ilustración 36 - Versión de MySQL devuelta con SQLi

Ahora vamos a testear la URL de búsqueda con una aplicación llamada sqlmap. Para ello copiamos la dirección de búsqueda de la página de BadStore:

URL:
http://192.168.1.43/cgi-bin/badstore.cgi?searchquery=&action=search&x=16&y=12

Seguidamente abrimos una terminal y ejecutamos sqlmap con el parámetro que está a continuación. La opción “--dbs” la usamos para que muestre las bases de datos disponibles.

Comando:
sqlmap -u “http://192.168.1.43/cgi-bin/badstore.cgi?searchquery=&action=search&x=16&y=12” --dbs

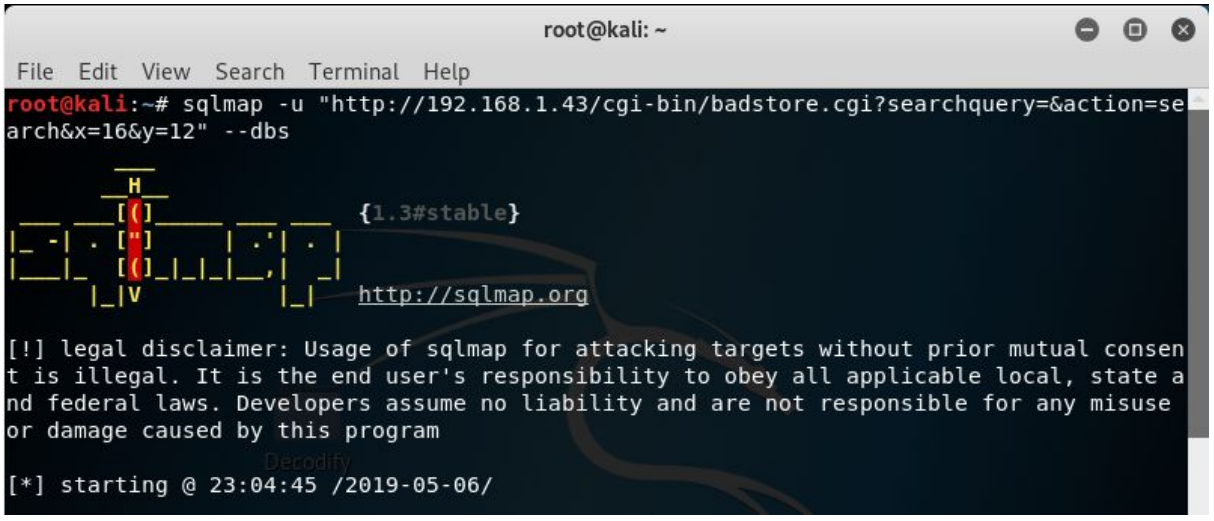


Ilustración 37 - Ejecución de sqlmap con la URL de búsqueda

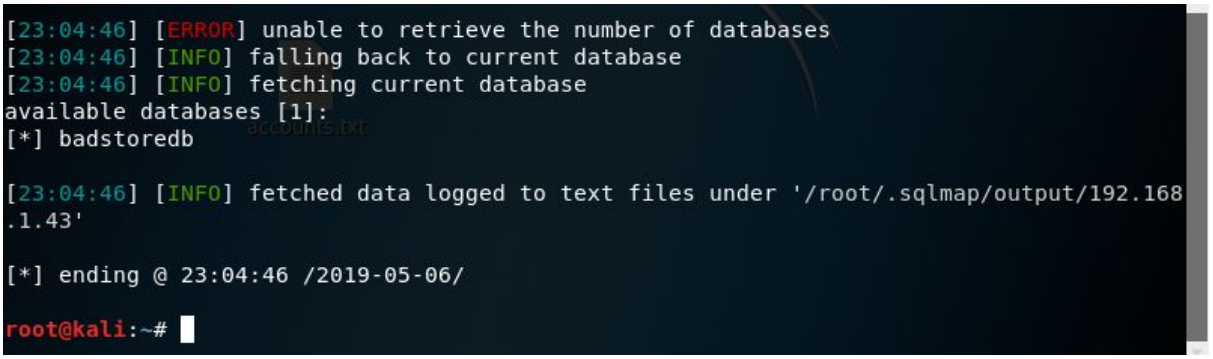


Ilustración 38 - Resultado de sqlmap con --dbs

Podemos ver en el resultado que nos ha encontrado la base de datos “badstoredb”. Ahora que sabemos la base de datos pasamos a buscar las tablas que tiene, para ello utilizamos la opción -D para conectarnos a la base de datos y --tables para listar las tablas.

Comando:
sqlmap -u “http://192.168.1.43/cgi-bin/badstore.cgi?searchquery=&action=search&x=16&y=12” -D badstoredb --tables

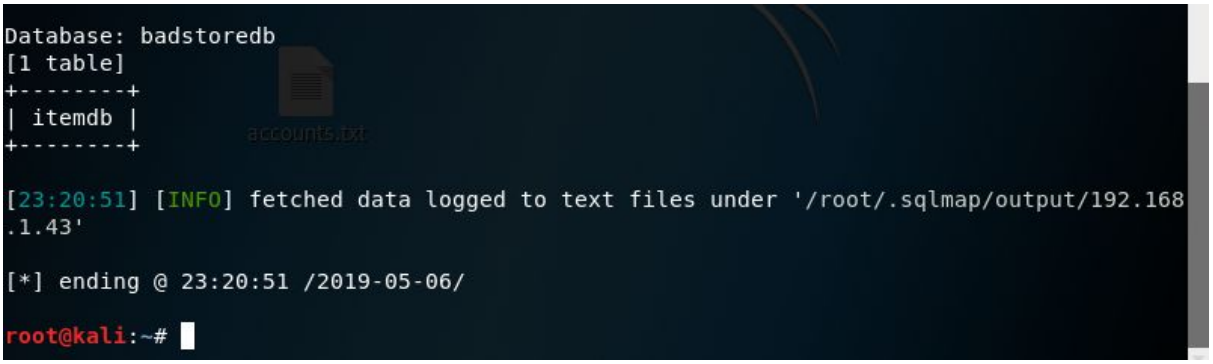


Ilustración 39 - Resultado con las tablas de la DB

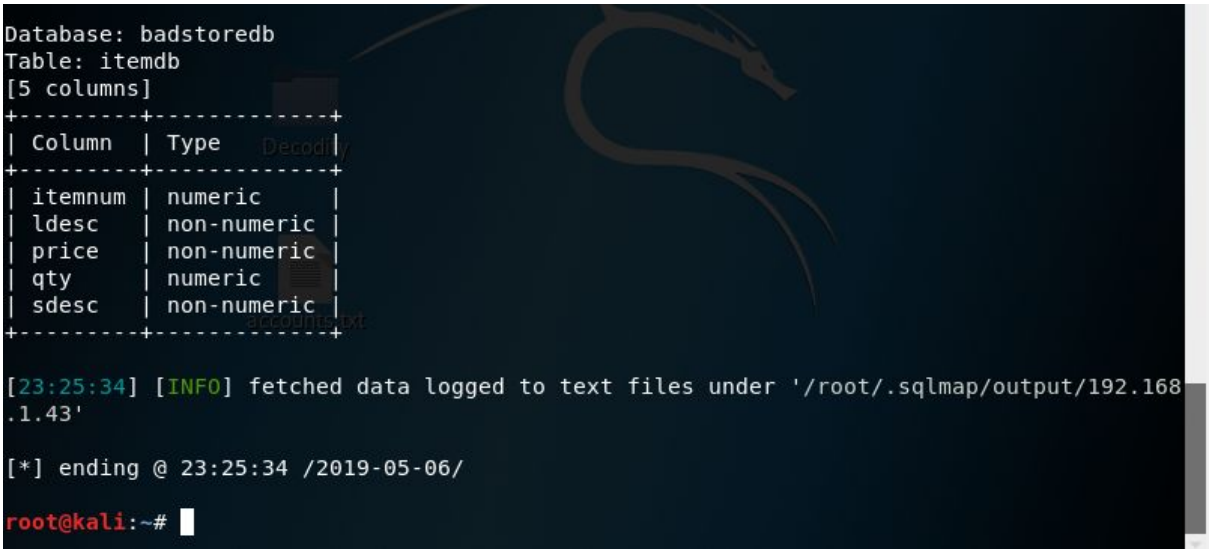


Ilustración 40 - Resultado con las columnas de la tabla

En la última imágen podemos ver que hemos conseguido todas las columnas de la tabla itemdb. Se ha utilizado el siguiente código:

Comando:
sqlmap -u “http://192.168.1.43/cgi-bin/badstore.cgi?searchquery=&action=search&x=16&y=12” -D badstoredb -T itemdb --columns

Con esta práctica hemos conseguido prácticamente todos los datos de la base de datos que está haciendo servir el servidor web.

Otro punto testeado ha sido el cuadro de búsqueda en sí. Al ser un input del usuario es posible que sea susceptible a XSS, tal y como hemos visto anteriormente con las firmas del guestbook. Vamos a probar con el siguiente payload: <h1>Vulnerable a XSS</h1>

Ilustración 41 - Vulnerabilidad XSS en el buscador

Introduciendo un simple código HTML podemos ver que se ha ejecutado correctamente y ha cambiado la página resultante. Incluso podemos introducir un script para ejecutar JavaScript. Introducimos: <script>alert(“XSS”)</script>

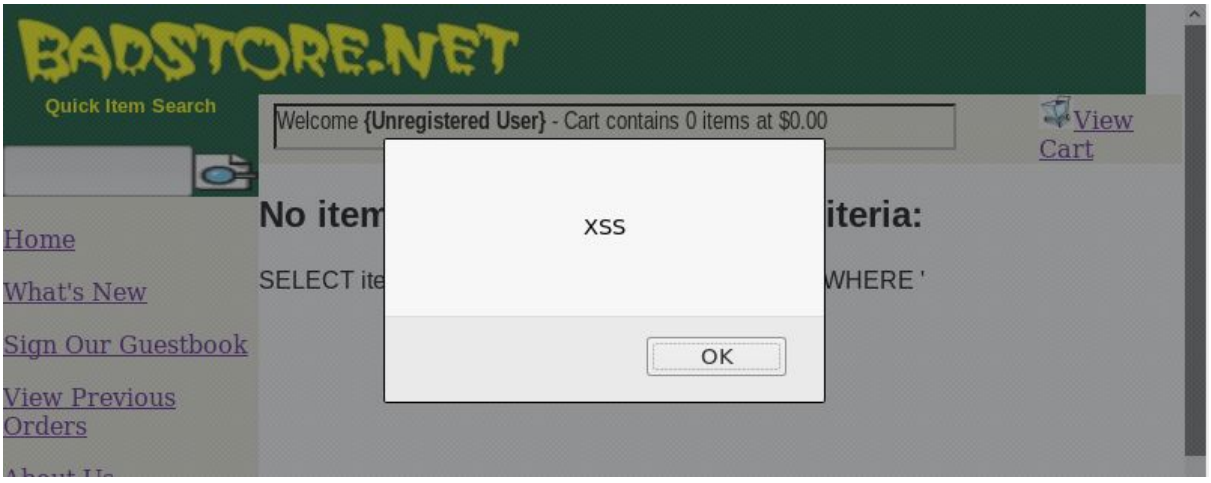


Ilustración 42 - Introducción de JavaScript a través del buscador

4.2. Caso práctico - COWEB

En este caso de pentesting vamos a utilizar la página web que hemos creado en la asignatura optativa de “Computación orientada al web” o también llamada COWEB. Como el otro caso práctico, vamos a analizar tanto la página web como su estructura y se va a detectar las distintas vulnerabilidades que existan.

La asignatura de COWEB consta de crear una página web con distintas funcionalidades que se van añadiendo a lo largo del semestre. Dicha página contiene scripts en JavaScript, conexiones AJAX, conexiones a bases de datos, etc.

Cabe destacar que el objetivo de la asignatura no se centra en la seguridad de aplicaciones y por lo tanto es posible que hayan varias vulnerabilidades tanto básicas como importantes.

Es un buen objetivo poder utilizar recursos creados en la carrera para su estudio y poder ver los errores de programación que se cometen y las vulnerabilidades que generan. Asimismo vamos a solucionar dichos problemas y vulnerabilidades encontradas y a mostrar el código para poder evitarlas.

4.2.1. Escenario

El escenario en este caso práctico es muy parecido al anterior. Vamos a utilizar máquinas virtuales para los testeos, concretamente dos.

La primera de ellas es un “Windows 10 x64” el cual contiene un servidor XAMPP. De este servidor utilizamos Apache para alojar la página web y MySQL para alojar las bases de datos.

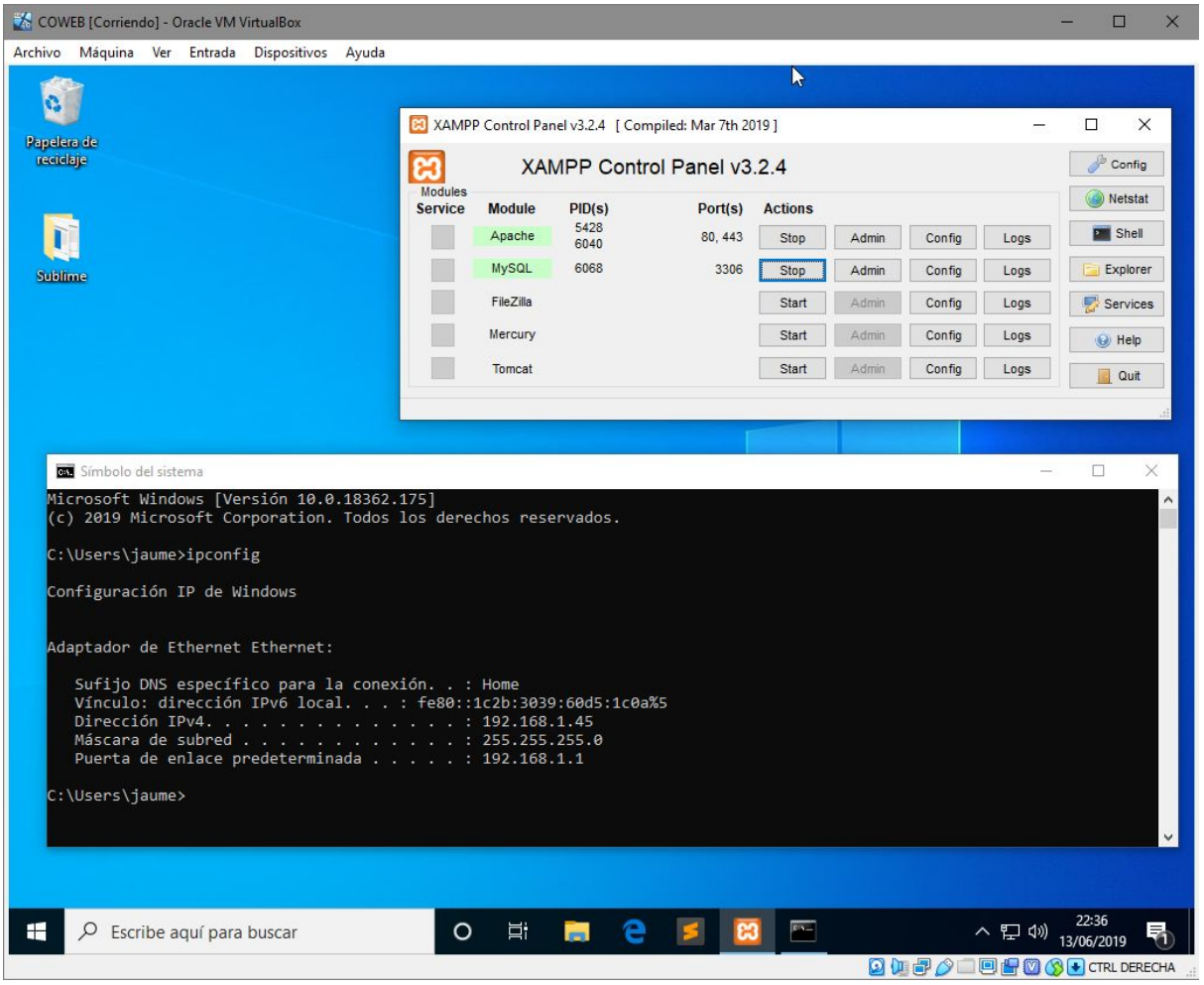


Ilustración 43 - Máquina virtual de COWEB

Podemos observar en la captura de pantalla que esta vez ejecutamos el servidor con la IP “192.168.1.45” y que tenemos ambos servicios ejecutándose.

La segunda máquina virtual contiene un Kali Linux la cual usaremos para realizar todo el análisis. Podemos decir que las especificaciones y la máquina como tal es igual que la del caso práctico anterior.

4.2.2. Pentesting

Vamos a empezar la parte de pentesting para buscar vulnerabilidades dentro de la página web, para ello empezamos por el reconocimiento. En la máquina virtual de Kali utilizamos un comando para escanear la red interna y ver qué dispositivos hay conectados.

Comando: nmap -sP 192.168.1.0/24

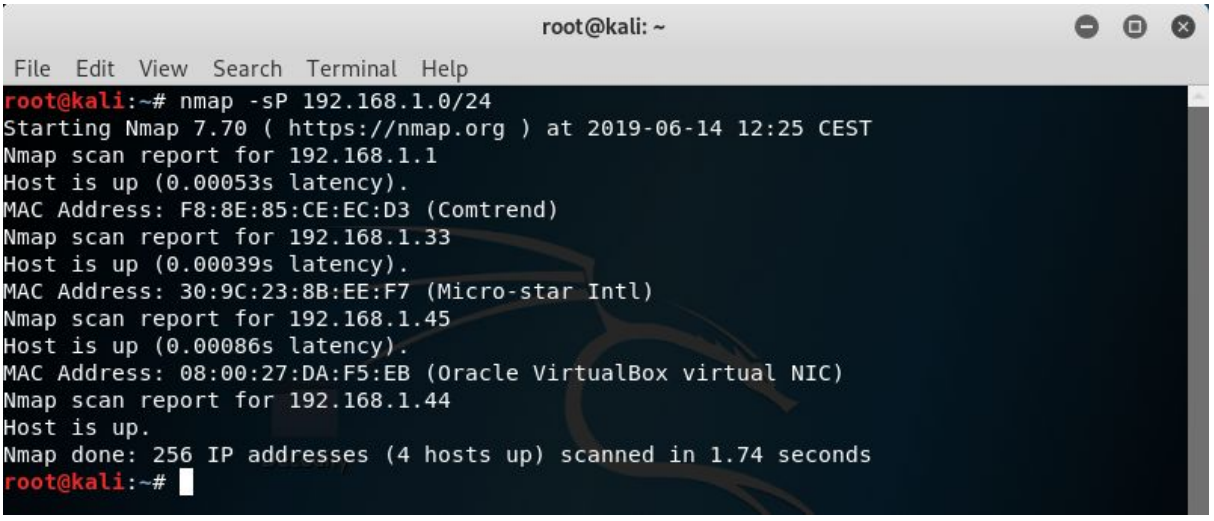


Ilustración 44 - nmap en Kali para explorar COWEB

Utilizando el comando anterior con la opción “-sP” (hace un sondeo ping y sólamente determina si los dispositivos están vivos o no) podemos observar que el objetivo está en la IP “192.168.1.45”. Al saber la IP objetivo, vamos a utilizar el comando sobre dicha IP con la opción “-sS” ya que nos dirá que puertos hay abiertos.

Comando: nmap -sS 192.168.1.45

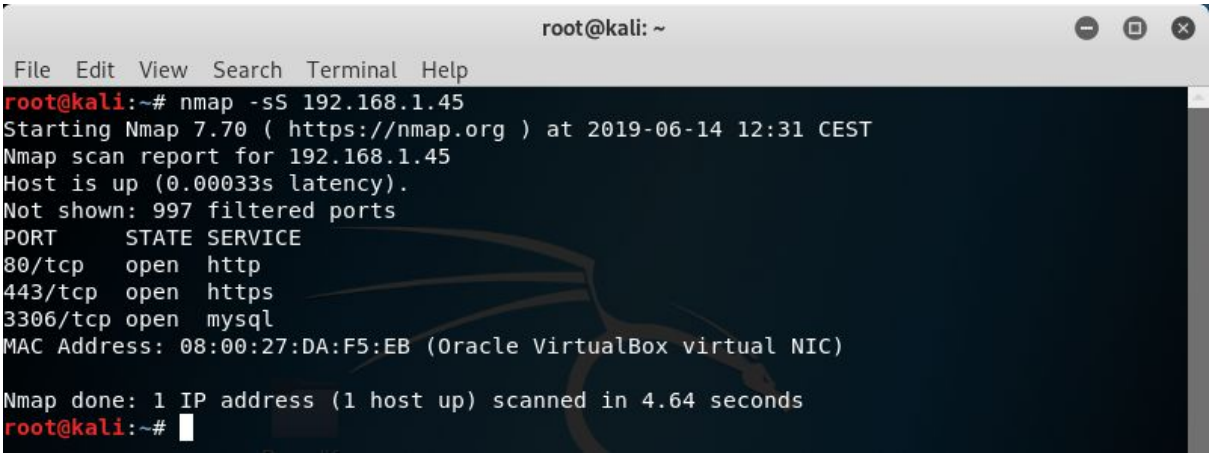


Ilustración 45 - nmap para ver los puertos de COWEB

Mediante el comando vemos que tiene los puertos 80, 443 y 3306 abiertos. Ya tenemos más información sobre el objetivo y como punto a destacar vemos el puerto de mysql.

Seguidamente realizamos otro escaneo pero esta vez sólamente de los puertos que hemos encontrado abiertos, de esta forma hacemos menos ruido en la red. Utilizamos el parámetro “-sV” que sirve para detectar versiones, también usamos “-O” para ver el sistema operativo.

Comando: nmap -sV -O -p 80,443,3306 192.168.1.45

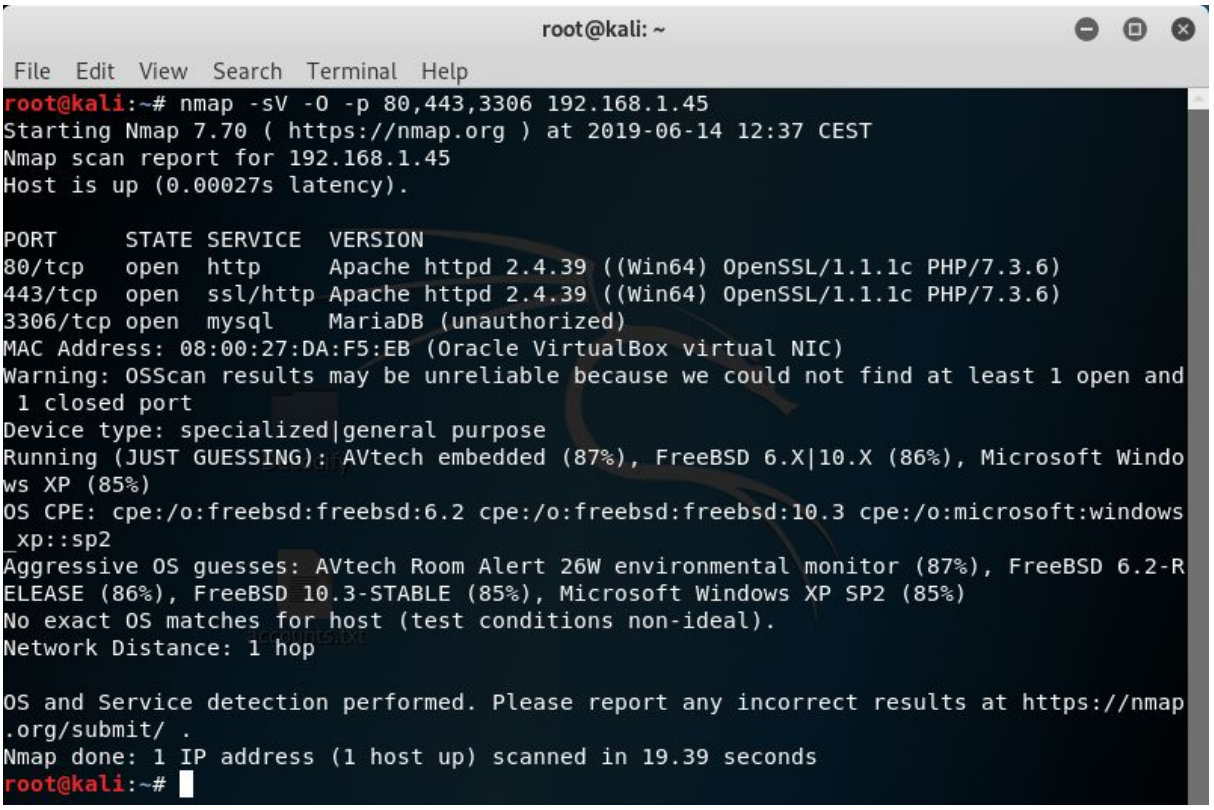


Ilustración 46 - Escaneo de puertos y versión a COWEB

En la ilustración podemos ver que se está usando un servidor Apache y una base de datos mysql MariaDB sobre un Windows. Con toda esta información disponible, podemos entrar en la página web y empezar a testear.

Vamos a dar una visión general de la página web para ver como se ve.

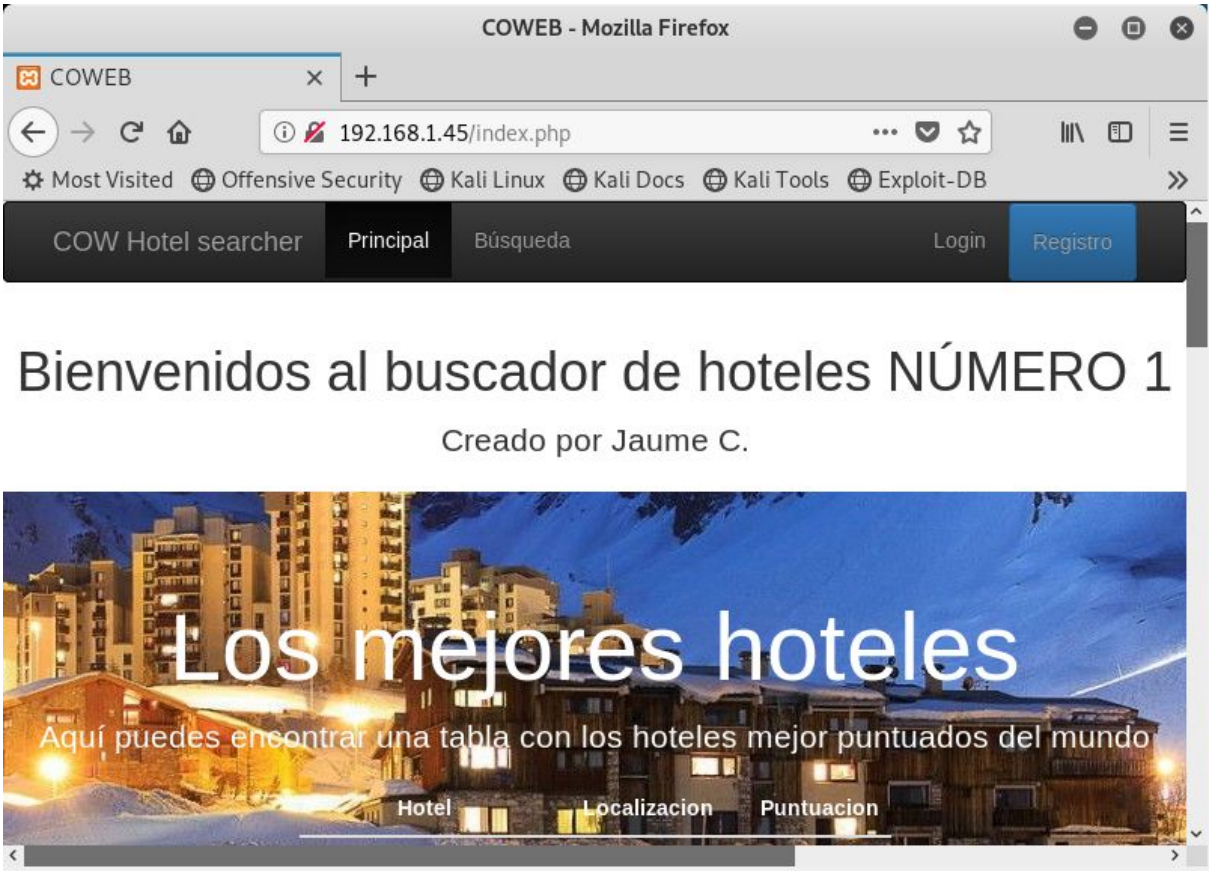


Ilustración 47 - Vista general de la página de COWEB

Podemos observar una página web sencilla, que pretende dar información al usuario. Lo primero que llama la atención es la barra de navegación del apartado superior. Vemos que hay dos enlaces, uno a la página principal y otro a otra página de búsqueda. Por otro lado tenemos dos botones, uno de login y otro de registro. Es una buena idea empezar por este apartado, ya que suele ser, junto a los formularios para introducir texto, una brecha de vulnerabilidades y un buen vector de ataque.

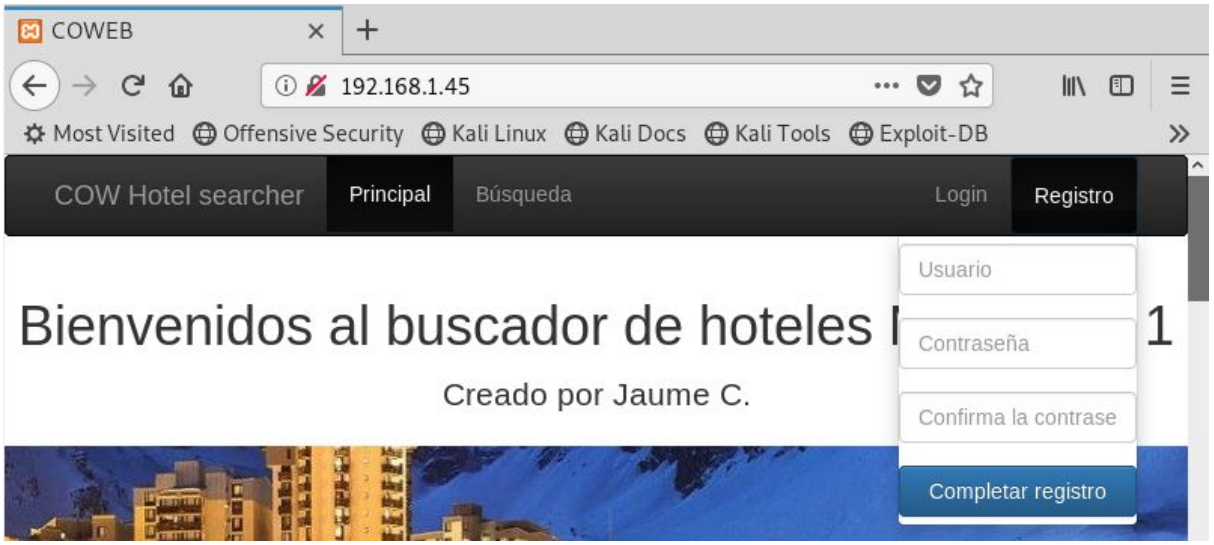


Ilustración 48 - Formulario de registro de COWEB

Vamos a empezar por crear un usuario. Cuando clicamos sobre el botón de registro podemos ver el formulario el cual nos pide un usuario y la contraseña con su confirmación. Vamos a crear el usuario “jaume” con contraseña “jaume”. Antes de clicar en el botón de “Completar registro” vamos a pulsar F12 en Firefox, vamos a la pestaña “Network” y filtramos por el tráfico HTML.

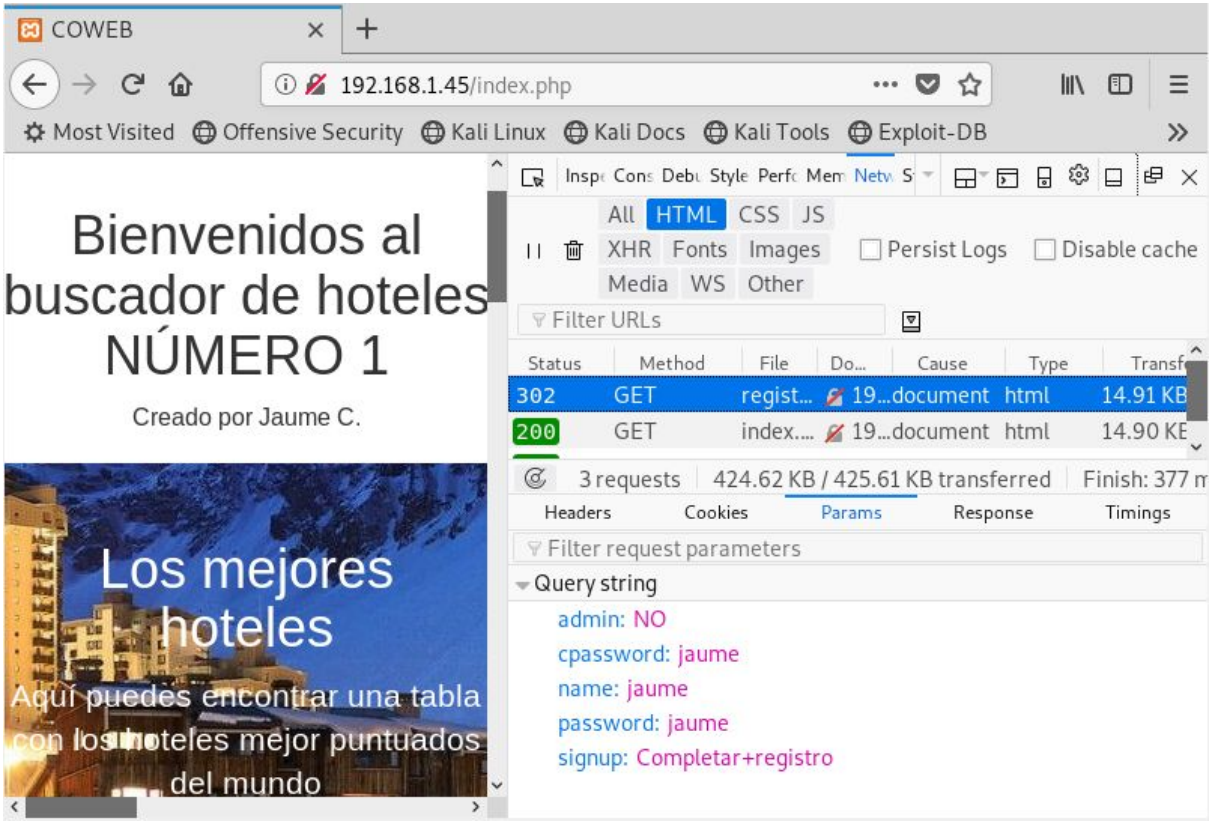


Ilustración 49 - Resultado del análisis de tráfico HTML para el registro de COWEB

Una vez creado el usuario y con el análisis del tráfico HTML podemos ver que hemos capturado una petición GET de la página “register.php” y podemos ver todos los parámetros pasados en la URL. Observamos que hay un parámetro oculto, ya que la página no nos lo ha pedido en ningún momento del formulario de registro, el de admin, el cual pone por defecto “NO”.

Una vez analizado el registro de un nuevo usuario y teniendo toda la información recopilada, vamos a pasar a intentar crear otro usuario pero esta vez directamente por la URL, ahora que sabemos qué parámetros son los que pasa por la URL.

URL:
http://192.168.1.45/register.php?name=fakeadmin&password=fakeadmin&cpassword=fakeadmin&admin=YES&signup=Completar+registro

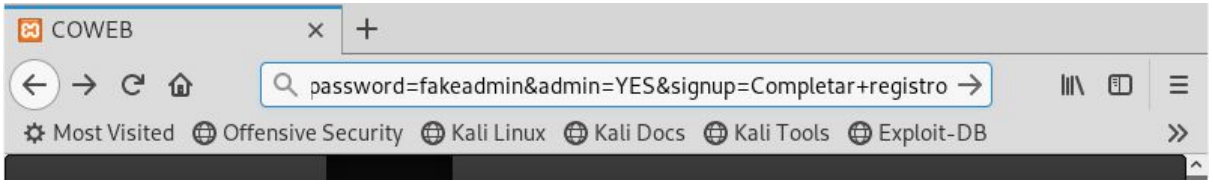


Ilustración XX - URL con el registro de usuario en COWEB

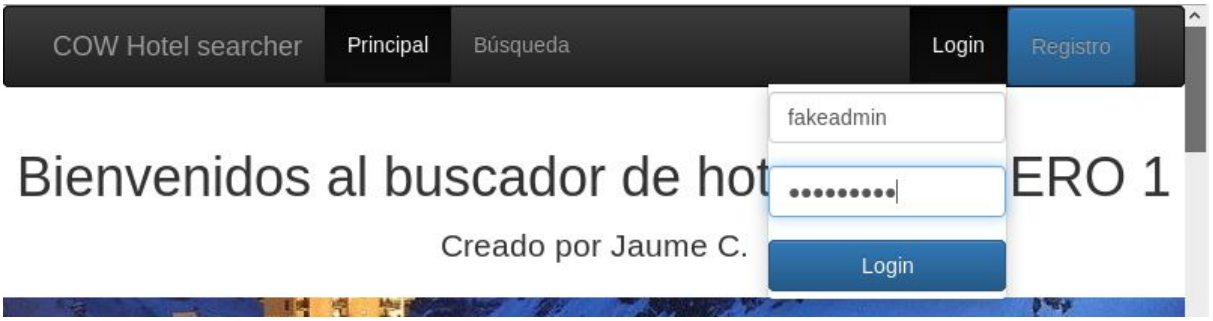


Ilustración 50 - Login con la cuenta fakeadmin en COWEB

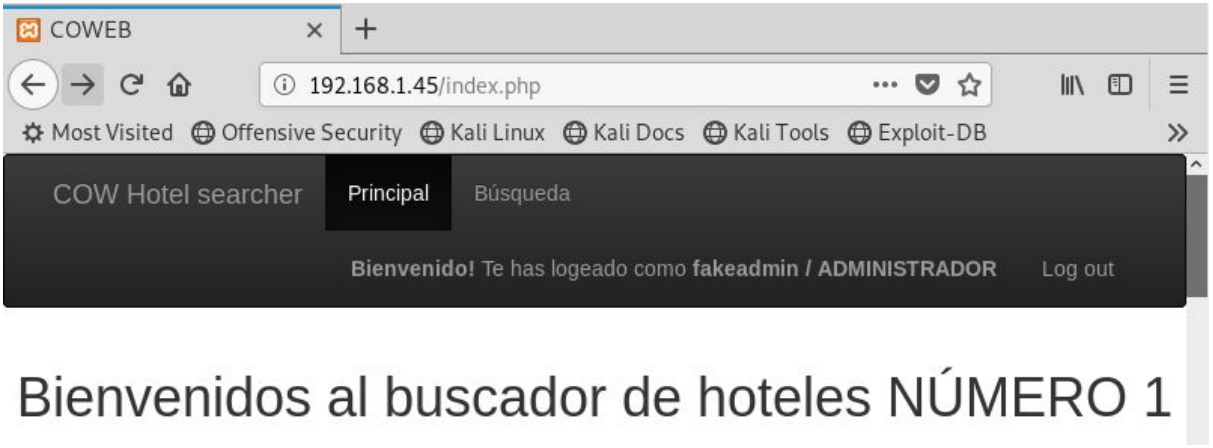


Ilustración 51 - Página principal como usuario fakeadmin en COWEB

Haciendo un login con los credenciales que hemos introducido por URL podemos ver como se ha creado la cuenta satisfactoriamente. Además, hemos introducido el parámetro “admin=yes” y hemos podido crear un usuario con permisos de administrador.

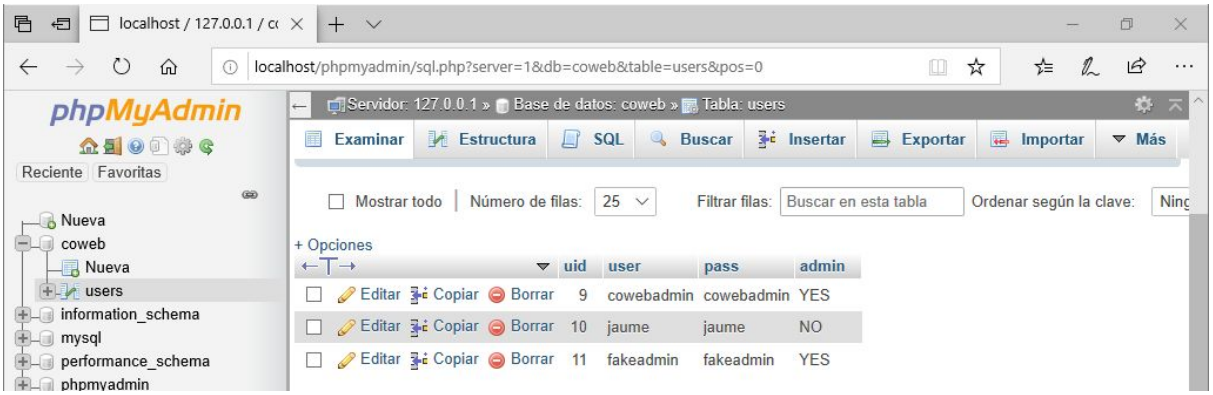


Ilustración 52 - Usuarios de la base de datos del servidor Windows de COWEB

En este punto que podemos crear usuarios con permisos de administrador sin problema alguno procedemos a testear el login. Los formularios que conectan con bases de datos suelen ser un objetivo clave a la hora de atacar o probar vulnerabilidades. Ahora vamos a introducir la siguiente consulta: “'or '1'='1”. Si la consulta a la base de datos está mal construida o concatenada se puede usar la consulta anterior en el usuario o contraseña para colarnos como el primer usuario de la tabla.

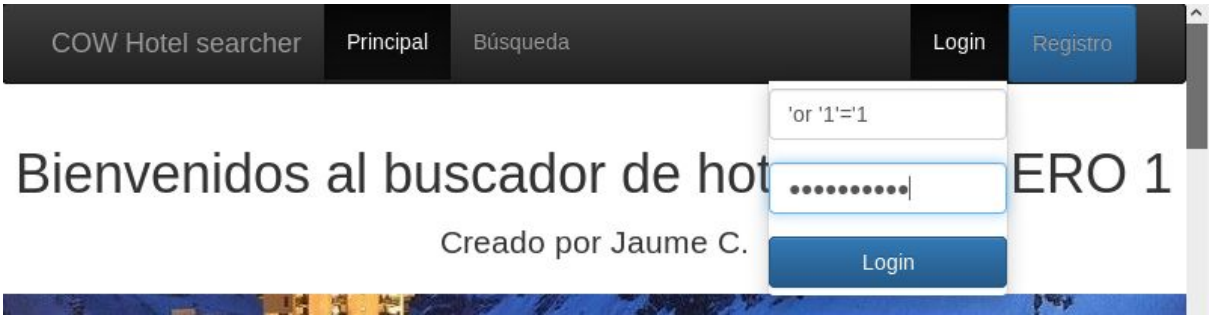


Ilustración 53 - Introducción de la consulta como usuario y contraseña

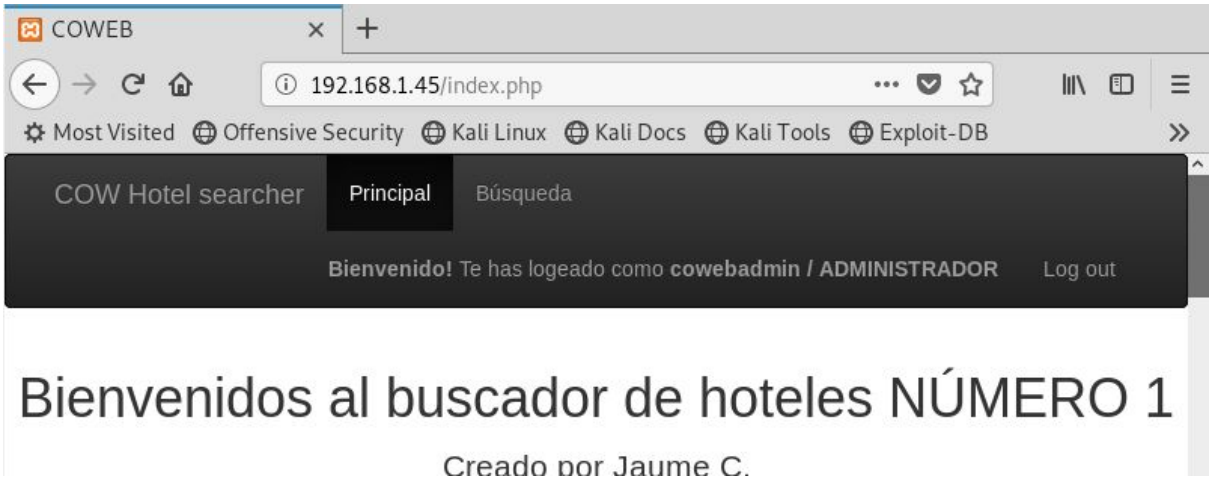


Ilustración 54 - Resultado de la introducción de SQLi

Como resultado podemos ver que al introducir esa consulta nos ha entrado con el usuario 1 de la base de datos, que es el administrador del sitio y ahora tenemos el control absoluto de la página web. Mediante esta prueba podemos decir que la web es vulnerable a SQL injection y es un problema muy grave, ya que cualquiera puede entrar como administrador.

Vamos a ver el código de la página de “login.php” y a corregir el error de vulnerabilidad que tenemos.

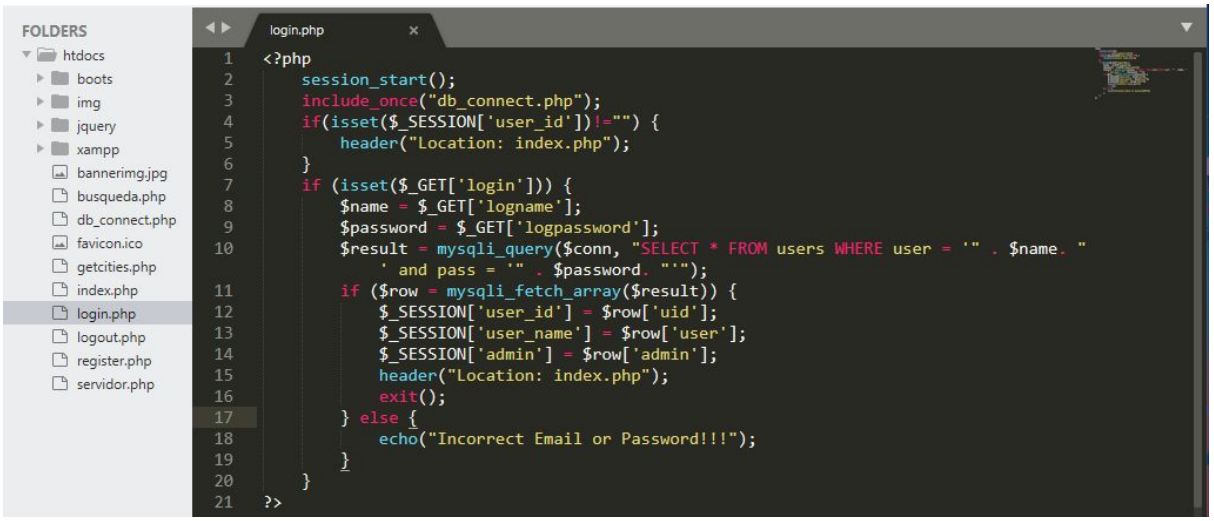


Ilustración 55 - Código de login.php

Lo primero que vemos en el código es que se está usando el método GET en vez de POST, esto va a ser lo primero que vamos a cambiar para más seguridad. Seguidamente podemos apreciar que para hacer la consulta con la base de datos se utiliza “mysqli_query” y como se ha comentado anteriormente, se concatena la búsqueda, es decir, selecciona todo de la tabla usuarios donde el usuario sea el parámetro que pasamos por la URL y la contraseña sea la misma. Para solucionar el problema con SQL injection simplemente tenemos que agregar a las variables pasadas por parámetro la siguiente función: “mysqli_real_escape_string”. Otra cosa que podemos hacer para aumentar la seguridad sería encriptar la contraseña, ya que se guarda en la base de datos sin protección alguna. El código quedaría así:

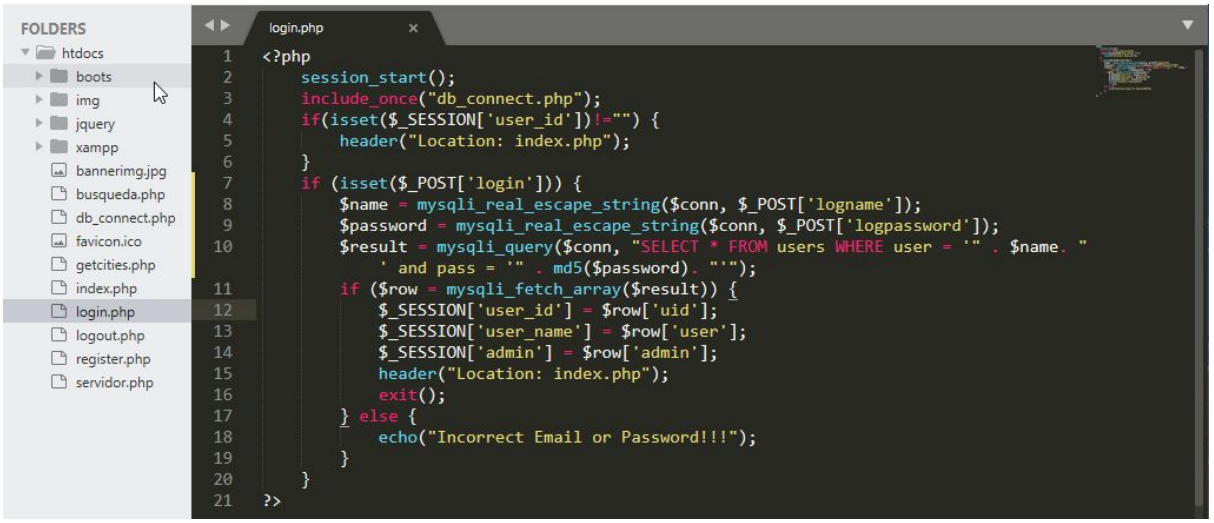


Ilustración 56 - Código modificado de login.php

Se ha modificado la parte pertinente en el formulario de login en “index.php” para que el método sea “post” en vez de “get”. Como resultado podemos comprobar que en el formulario ya no es posible hacer SQL injection y la página está protegida.

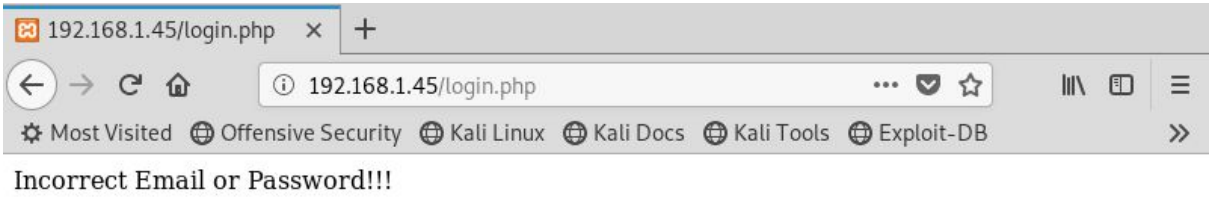


Ilustración 57 - Resultado fallido de introducción de SQL injection

Ahora vamos a modificar la parte de registro para arreglar los problemas de creación de usuarios.

```
<li class="nav-item">
    <a href="#" data-toggle="dropdown" class="btn
    btn-primary dropdown-toggle get-started-btn mt-1 mb-1
    ">Registro</a>
    <ul class="dropdown-menu form-wrapper">

        <li>
            <form action="register.php" method="get">
                <div class="form-group">
                    <input type="text" class="
                    form-control" name="name" placeholder
                    ="Usuario" required="required">
                </div>
                <div class="form-group">
                    <input type="password" class="
                    form-control" name="password"
                    placeholder="Contraseña" required="
                    required">
                </div>
                <div class="form-group">
                    <input type="password" class="
                    form-control" name="cpassword"
                    placeholder="Confirma la contraseña"
                    required="required">
                </div>
                <input type="hidden" name="admin" value="
                NO">
                <input type="submit" name="signup" class=
                "btn btn-primary btn-block" value="
                Completar registro">
            </form>
```

Ilustración 58 - Código de la parte de registro de index.php

Lo primero de todo que vamos a eliminar va a ser el campo oculto que pasamos como parámetro del formulario en el cual ponemos como valor de administrador “NO”. Este apartado lo borramos y lo pasaremos directamente en la consulta de SQL. Otra modificación básica que hacemos es cambiar el método del formulario por “POST”.

Vamos a ver el código de “register.php”.

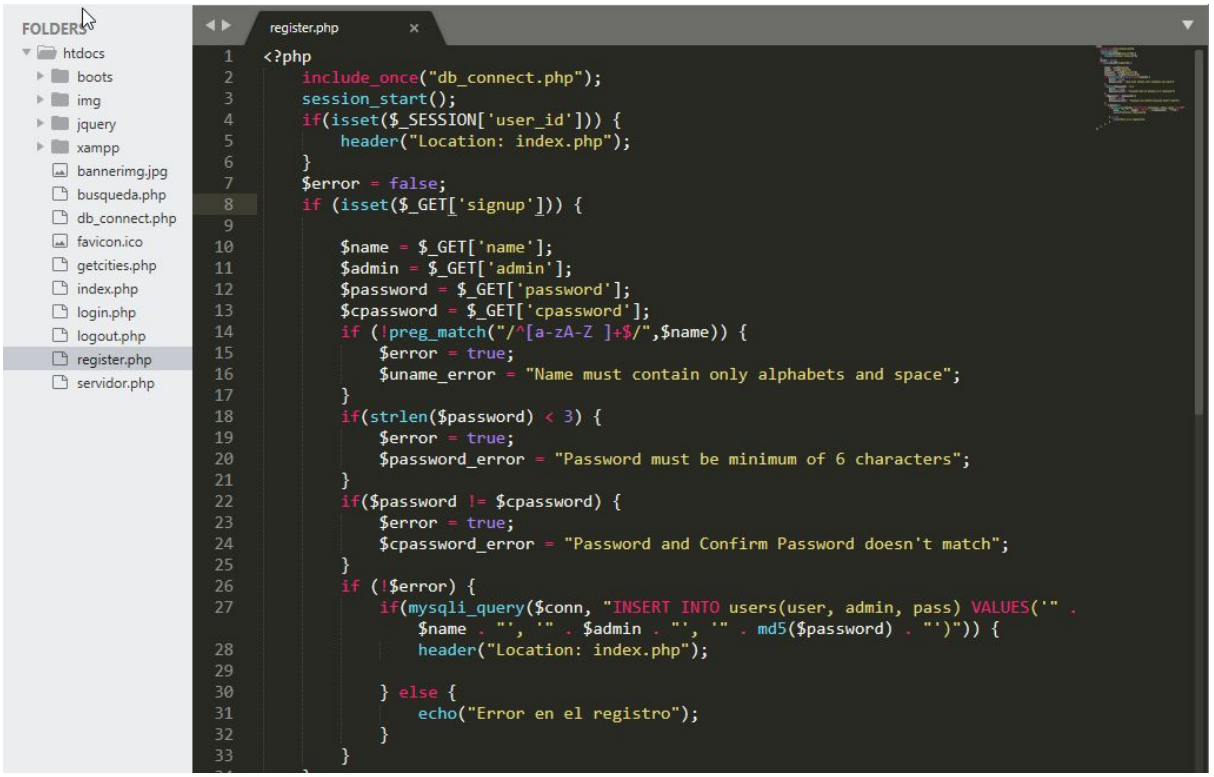


Ilustración 59 - Código de register.php

Lo que vamos a modificar en este código van a ser los métodos con los que coje las variables, vamos a cambiarlos por “POST” y como hicimos en “login.php” vamos a agregar la función “mysqli_real_escape_string” a las variables. A parte, vamos a borrar la parte de obtener el parámetro de ADMIN y lo pasamos por el insert de la base de datos.

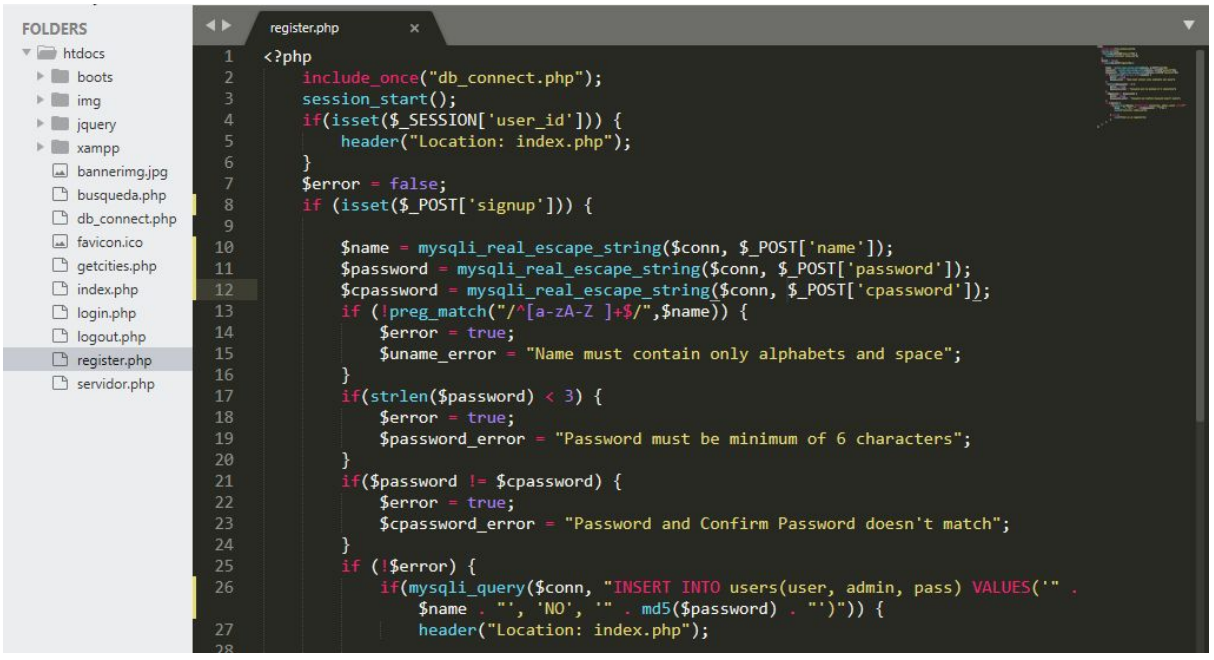


Ilustración 60 - Código modificado de register.php

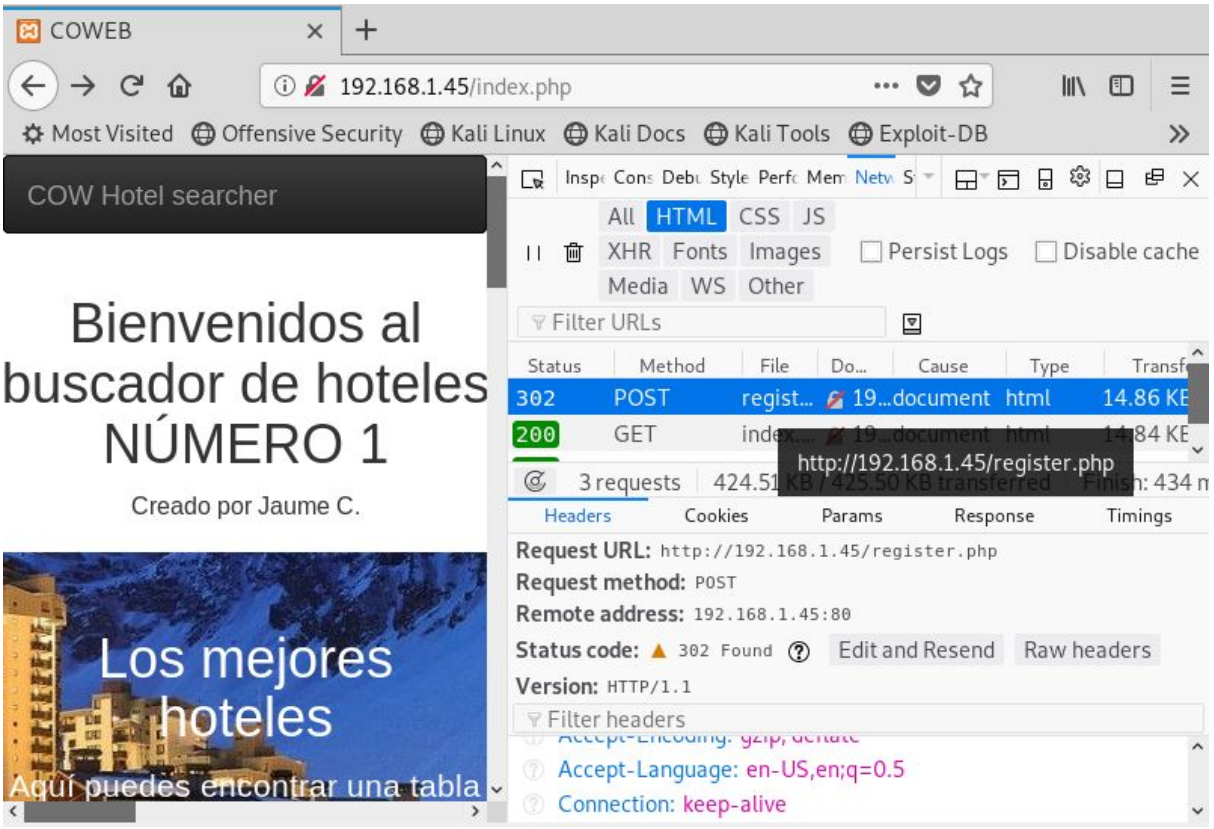


Ilustración 61 - Resultado de creación de un nuevo usuario en COWEB

De esta forma hemos ocultado los parámetros de la URL, impidiendo que los usuarios sepan qué estructura se utiliza para la creación de usuarios. Asimismo pasamos el parámetro de admin en la consulta de SQL y no como parámetro, impidiendo a los usuarios poder crear usuarios con permisos de administrador.

Vamos a ver el apartado de búsqueda de la página web, que aún no hemos testeado.

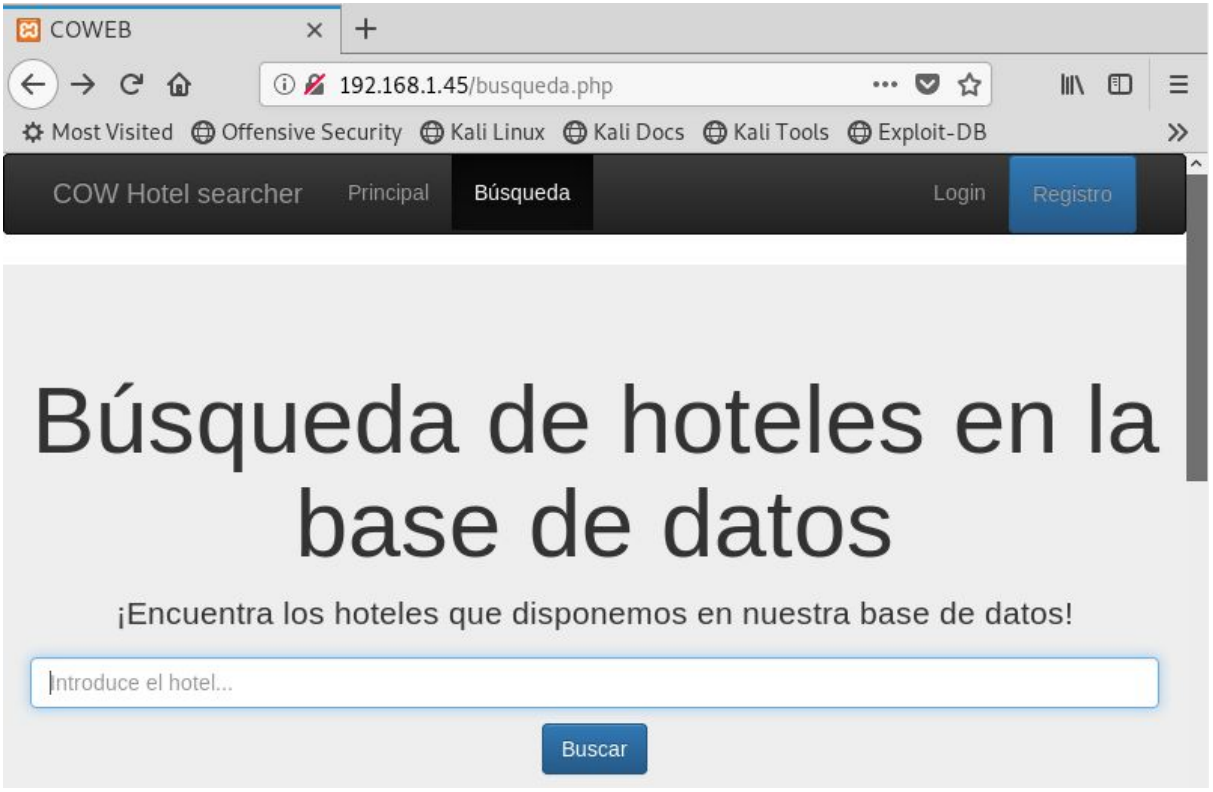


Ilustración 62 - Página de búsqueda de COWEB

La función de este formulario es muy sencilla, simplemente nos muestra por pantalla lo que escribimos para buscar.

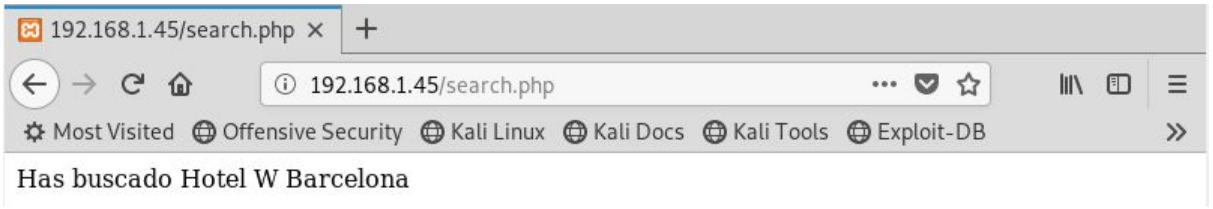


Ilustración 63 - Resultado de búsqueda de COWEB

Vamos a probar la introducción de código JavaScript.

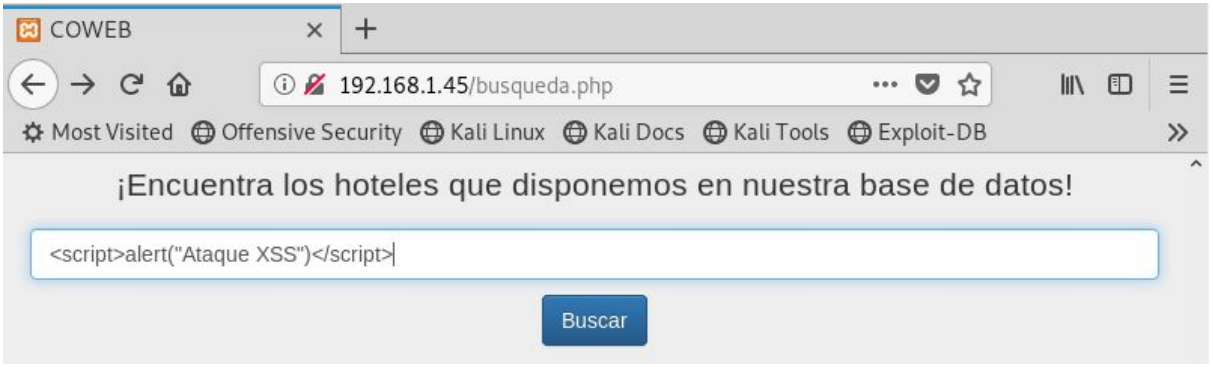


Ilustración 64 - Introducción de código JS en el buscador de COWEB

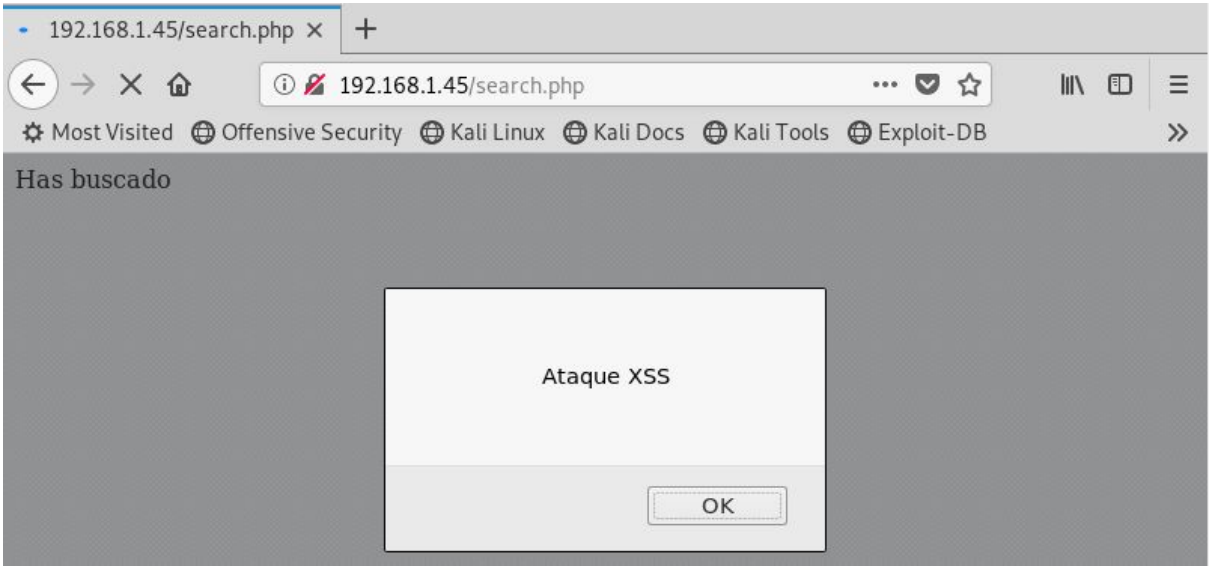


Ilustración 65 - Resultado de la introducción del script en el buscador

El archivo “search.php” no hace ningún control sobre lo que introduce el usuario y podemos ver en la ilustración que se ejecuta perfectamente el código JS introducido. Tenemos una vulnerabilidad de Cross-site scripting o XSS.

Para solucionar este fallo simplemente tenemos que modificar el código php de “search.php” y agregar una función a la hora de tomar los parámetros pasados con POST.

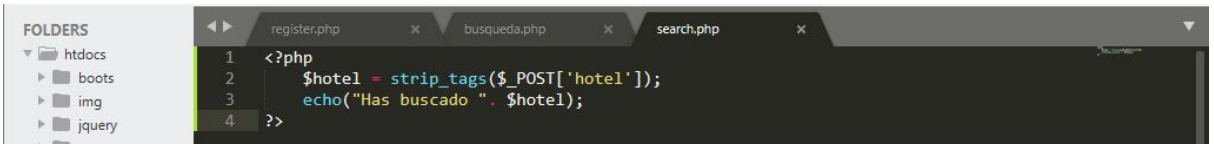


Ilustración 66 - Código de búsqueda arreglado

Al utilizar la función de “strip_tags” lo que hacemos es hacer un control sobre el input introducido por el usuario, borrando los tags que pueda poner y así evitar que pueda introducir código malicioso. Hay muchas formas de poder evitar ataques XSS y esta es una de ellas. Si ahora introducimos el mismo código que anteriormente tendremos:

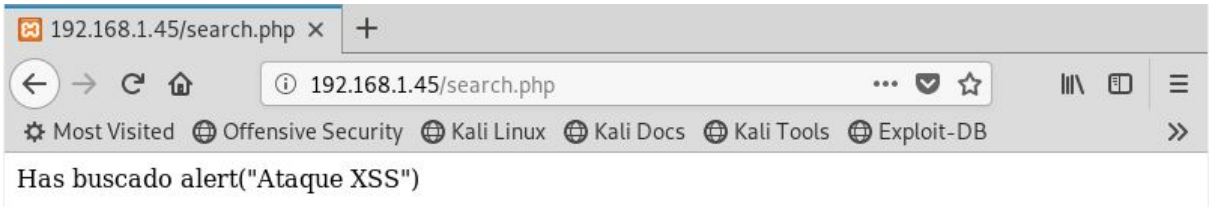


Ilustración 67 - Resultado de la corrección del código de búsqueda

Podemos observar que simplemente muestra el string introducido, obviando los tags de “<script>”. De esta forma, ahora tenemos la web segura de ataques de SQL injection y de XSS.

5. GLOSARIO

Acunetix - Empresa dedicada a la seguridad informática, pionera en el desarrollo de software de automatización de seguridad de aplicaciones web.

BadStore - Aplicación web dedicada para practicar los ataques a las principales vulnerabilidades web que podemos encontrar a día de hoy.

Black hat - Un hacker de sombrero negro o black hat es un pirata informático que viola la seguridad informática para obtener beneficios personales o maliciosos.

Blacklist - Es un mecanismo de control de acceso que deniega el acceso al contenido que sea mediante un listado de distintos elementos, como por ejemplo, direcciones IP.

Bounty Program - También llamado “Bug Bounty Program” es una oferta o trato ofrecido por páginas web, organizaciones y software developers por la que personas pueden recibir reconocimiento y una compensación (normalmente monetaria) por encontrar bugs.

CVE Details - Fuente de información de vulnerabilidades de seguridad CVE gratuita. Se puede consultar los detalles de vulnerabilidad de CVE, exploits, referencias, metasploit…

Decodify - Aplicación para detectar y decodificar strings encriptados.

DirBuster - Aplicación diseñada para obtener por fuerza bruta los nombres de directorios y archivos en servidores web.

Grey hat - Referencia al hacker que actúa ilegalmente, pero con buenas intenciones. Típicamente un híbrido entre white hat y black hat.

Hacking ético - El hacking ético se define a través de lo que hacen los profesionales que se dedican a ello, los hackers éticos. Estas personas son contratadas para hackear un sistema e identificar y reparar posibles vulnerabilidades.

Kali Linux - Distribución de Linux basada en Debian diseñada principalmente para la auditoría y seguridad informática en general. Fundada y mantenida por Offensive Security.

Metasploit mysql_login - Módulo de Metasploit para encontrar credenciales de servidores MySQL por fuerza bruta.

nmap - Utilidad open source para rastreo de redes y auditorías de seguridad.

Pentesting - También llamado test de penetración, consiste en atacar un sistema informático para identificar fallos, vulnerabilidades y demás errores de seguridad existentes.

setoolkit - Herramienta open source enfocada al testeo de penetraciones alrededor de la ingeniería social.

SQLi - Inyección SQL. Método de infiltración de código intruso que se vale de una vulnerabilidad presente en una aplicación en el nivel de validación de las entradas para realizar operaciones sobre una base de datos.

sqlmap - Software de código abierto utilizado para detectar y explotar vulnerabilidades de bases de datos. Proporciona opciones para inyectar códigos maliciosos en ellas.

VirtualBox - Software de virtualización desarrollado por Oracle Corporation.

White hat - Referencia a un hacker ético especializado en pruebas de penetración y en otras metodologías para detectar vulnerabilidades y mejorar la seguridad de los sistemas. También hace referencia a los hackers que no son agresivos o no realizan actividades ilícitas.

Wireshark - Es un analizador de protocolos utilizado para realizar análisis y solucionar problemas en redes de comunicaciones, para desarrollo de software y protocolos y como una herramienta didáctica. Antes conocido como Ethereal.

XSS - Cross-site scripting. Tipo de vulnerabilidad informática que puede permitir a una tercera persona inyectar código JavaScript o lenguajes similares.

6. ANEXOS

En esta sección se anexan las distintas partes de teoría que no han entrado en el trabajo principal. Asimismo estas secciones son igualmente importantes y necesarias para el entendimiento de las distintas vulnerabilidades y metodologías que hay disponibles.

Cabe destacar que previamente estas partes se encontraban dentro del grueso del trabajo principal pero finalmente se han trasladado a esta sección.

6.1. METODOLOGÍAS

El hacking ético es una especialización dentro del ámbito de la seguridad informática la cual está más buscada o reconocida por las grandes empresas a nivel mundial, cada día crece la necesidad de tener personas con los suficientes conocimientos en estas áreas para poder contrarrestar los ataques de la creciente comunidad de hackers.

Se usan distintos términos o clasificaciones para diferenciar a los distintos tipos de hacker los cuales suelen ir definidos por el estatuto jurídico de las actividades que realizan. Vamos a dividirlos en tres grupos principales.

- **Hacker de sombrero blanco (White hat):** Este tipo de hackers penetra la seguridad por razones no maliciosas, ya bien sea para poner a prueba la seguridad de su sistema o bien mientras trabaja para una compañía de software de seguridad. El término de white hat hace referencia a un hacker ético. También podríamos incluir en este grupo las personas que hacen pruebas de penetración (pentesting) y evaluaciones de vulnerabilidad dentro de un acuerdo contractual. Actualmente hay certificaciones, cursos y clases en línea cubriendo toda la amplia esfera de hacking ético.

- **Hacker de sombrero negro (Black hat):** Los hackers de sombrero negro son personas que violan la seguridad por razones más allá de la malicia o para el beneficio personal. Estas personas son la personificación de todo lo que el público teme de un hacker. Las actividades que realizan pueden ir desde entrar a redes seguras para destruir o encriptar los datos a publicar dichos datos para beneficio personal.

- **Hacker de sombrero gris (Grey hat):** Estos hackers son una combinación de los white hat y los black hat. Este tipo de hackers puede violar un sistema informático con el propósito de notificar al administrador que su sistema ha sido vulnerado, luego ofreciéndose para reparar dicho sistema por un precio.

La línea que separa a un hacker de sombrero blanco a uno de sombrero negro es muy delgada. En cuanto a conocimientos ambos tienen la capacidad de reconocer vulnerabilidades y fallos en sistemas, para sacar provecho de la situación , el hacker ético o white hat tiene como objetivo explotar las vulnerabilidades y reportarlas, el fin no es (o no debería ser) el sacar provecho económico de la situación, por lo contrario el objetivo es hacer recomendaciones o diseñar soluciones para mejorar el sistema.

El resultado del ataque de los sistemas y redes alrededor de todo el mundo ha provocado la pérdida o modificación de los datos sensibles a las organizaciones a lo largo del tiempo, representando miles o millones de dólares. Esta situación es debido a la ineficiencia de los esquemas de seguridad con los que cuentan la mayoría de empresas. Para solucionar este problema existen metodologías para el hacking ético las cuales nos ayudan a alcanzar una seguridad informática apropiada para cualquier red o sistema.

6.1.1. Listado de metodologías

A continuación se presenta un listado con las metodologías más importantes y más seguidas a día de hoy en cuanto a seguridad se refiere.

OWASP (Open Web Application Security Project).

OWASP es un proyecto de código abierto dedicado a determinar y combatir las causas que hacen que el software sea inseguro. Es una metodología de pruebas la cual está enfocada en la seguridad de aplicaciones. Pretende ayudar a las personas a evaluar y tomar medidas de seguridad a través de todo el proceso de desarrollo de software. Asimismo se pueden relacionar los costes de de un software inseguro al impacto sobre el negocio y de este modo gestionar las decisiones o recursos apropiados para la gestión del riesgo. Las características más representativas de OWASP son:

- Pruebas de firma digital de aplicaciones Web.
- Comprobaciones del sistema de autenticación.
- Pruebas de Cross Site Scripting.
- Inyección XML
- Inyección SOAP
- HTTP Smuggling
- Sql Injection
- LDAP Injection
- Polución de Parámetros
- Cookie Hijacking
- Cross Site Request Forgery

OSSTMM (Open-Source Security Testing Methodology Manual).

Esta metodología propone un proceso de evaluación de una serie de áreas las cuales reflejan los niveles de seguridad presentes en una infraestructura. Dicha metodología está documentada perfectamente en un manual.

Este manual es uno de los estándares profesionales más completos y utilizados en las auditorías de seguridad para revisar la seguridad de los sistemas desde internet.Incluye un marco de trabajo que describe las fases que hay que realizar para la ejecución de la auditoría. Este trabajo se ha realizado con la colaboración de más de 150 expertos colaborando entre sí a través de internet. Esta metodología consta de seis partes las cuales comprenden todo el sistema:

- Seguridad de la información.
- Seguridad de los procesos.
- Seguridad de las tecnologías de internet.
- Seguridad de las comunicaciones.
- Seguridad inalámbrica.
- Seguridad física.

El resultado obtenido es un entendimiento profundo sobre la conectividad de todos los elementos. La gente, procesos, sistemas y todo el software están relacionados de alguna forma entre ellos.

ISSAF (Information Systems Security Assessment Framework).

Marco metodológico de trabajo desarrollado por la OISSG que permite clasificar la información de la evaluación de seguridad en diversos dominios usando diferentes criterios de prueba. Algunas de las características más representativas de ISSAF son:

- Brinda medidas que permiten reflejar las condiciones de escenarios reales para las evaluaciones de seguridad.
- Esta metodología se encuentra principalmente enfocada en cubrir los procesos de seguridad y la evaluación de los mismos para así obtener un panorama completo de las vulnerabilidades existentes.
- Permite el desarrollo de matriz de riesgo para verificar la efectividad en la implementación de controles.

CEH (Certified Ethical Hacker).

Metodología de pruebas de seguridad desarrollada por el International Council of Electronic Commerce Consultants (EC-Council) algunas de las fases enunciadas en esta metodología son:

- Obtención de Información.
- Obtención de acceso.
- Enumeración.
- Escala de privilegios.
- Informe.

OFFENSIVE SECURITY.

Metodología líder a nivel mundial para el desarrollo de pruebas de penetración y estudios de seguridad, la metodología contempla principalmente los métodos para el desarrollo de estudios de seguridad enfocados en seguridad ofensiva y teniendo como marco la posibilidad real de explotación independientemente de los indicadores de riesgos y vulnerabilidades, las principales ventajas de adoptar este marco metodológico son:

- Enfoque sobre la explotación real de las plataformas.
- Enfoque altamente intrusivo.
- Enfoque orientado a resultados tangibles y no a estadísticas generadas por herramientas.

6.1.2. Patrones de hacking

En este apartado vamos a describir una aproximación básica sobre el hacking de una aplicación. Primero de todo hay que redefinir qué es lo que consideramos éxito. Se puede considerar como objetivo el encontrar bugs en un programa, encontrar todos los bugs que se pueda o simplemente hacer dinero. Si por ejemplo el objetivo es encontrar vulnerabilidades o bugs en aplicaciones importantes, el objetivo de ganar dinero puede no estar asegurado. Los hackers asiduos hacen pruebas con estas aplicaciones y por ello es bastante probable que no se pueda encontrar ningún bug nuevo y por lo tanto no recibir una compensación económica. Es importante definir el éxito como la experiencia y el conocimiento ganado, en vez de los bugs encontrados o el dinero ganado. El objetivo debería ser el aprender algo nuevo, reconocer patrones y testear nuevas tecnologías.

Redefinir el concepto de éxito de esta forma ayuda a estar positivo durante el hacking, las vulnerabilidades vendrán con el tiempo, ya que mientras las personas estén escribiendo código, van a cometer errores y por lo tanto habrá bugs o vulnerabilidades por descubrir. Una vez definido el concepto de éxito hay que aplicar una metodología.

6.1.2.1. Reconocimiento

Antes de empezar a testear una aplicación siempre hay que hacer un reconocimiento previo para aprender más sobre esta. Hay muchas partes a considerar cuando se testea una aplicación y se puede empezar haciendo una serie de preguntas básicas:

- ¿Cual es el alcance del programa?
- ¿Cuantos subdominios tiene la compañía?
- ¿Cuántas direcciones IP tiene la compañía?
- ¿Qué clase de sitio es?
- ¿Qué tecnologías usa?
- ¿En qué lenguaje está programado?
- ¿Que framework usa?
- ¿Que base de datos tiene?

Estas son solo algunas de las cosas que deberíamos considerar al empezar. También hay que tener en cuenta que para responder todas estas preguntas y hacer un testeo correcto de la aplicación normalmente se utilizan distintas herramientas las cuales se pueden ejecutar en segundo plano. Estas herramientas que se utilizan para hacer el reconocimiento y adquirir la información que precisamos, se puedan utilizar desde el ordenador propio, pero se corre el riesgo de que empresas de seguridad o firewalls en la red agreguen la propia IP en una blacklist impidiendo la navegación a cualquier página web.

Para evitar estos casos, lo que se suele hacer es utilizar un servidor privado virtual (VPS) desde un hosting remoto el cual permita realizar pruebas seguras desde sus sistemas.

Vamos a ver a continuación un listado de pasos a seguir dentro del apartado de reconocimiento.

1.1. Enumeración de subdominios.

Se puede empezar el reconocimiento de la aplicación web por encontrar subdominios utilizando las VPS. Cuantos más subdominios se encuentren, más superficie estará disponible para el ataque. Hay una aplicación destinada a esta tarea llamada “Subfinder”, la cual obtiene registros de subdominio para un sitio en concreto, incluyendo registros de certificados, resultados de motores de búsqueda y otros.

Los subdominios asociados a un certificado SSL específico son fáciles de encontrar debido a los logs de registros transparentes que los certificados SSL registran. Por poner un ejemplo, si un sitio registra un certificado para “test.example.com”, es muy probable de que el subdominio exista. Por el contrario, es posible registrar un certificado para un subdominio “*.example.com” en el cual solo podríamos encontrar subdominios por fuerza bruta.

Para encontrar subdominios a través de fuerza bruta existen distintas “wordlists” las cuales contienen un listado de los subdominios más comunes. Por ejemplo la aplicación comentada anteriormente nos da esa opción, otros usuarios han colgado listas las cuales son públicas para que todo el mundo las utilice, como por ejemplo la lista de Jason Hendrix. Para buscar certificados SSL, crt.sh es una gran referencia para comprobar si los certificados comodín (wildcard) han sido registrados.

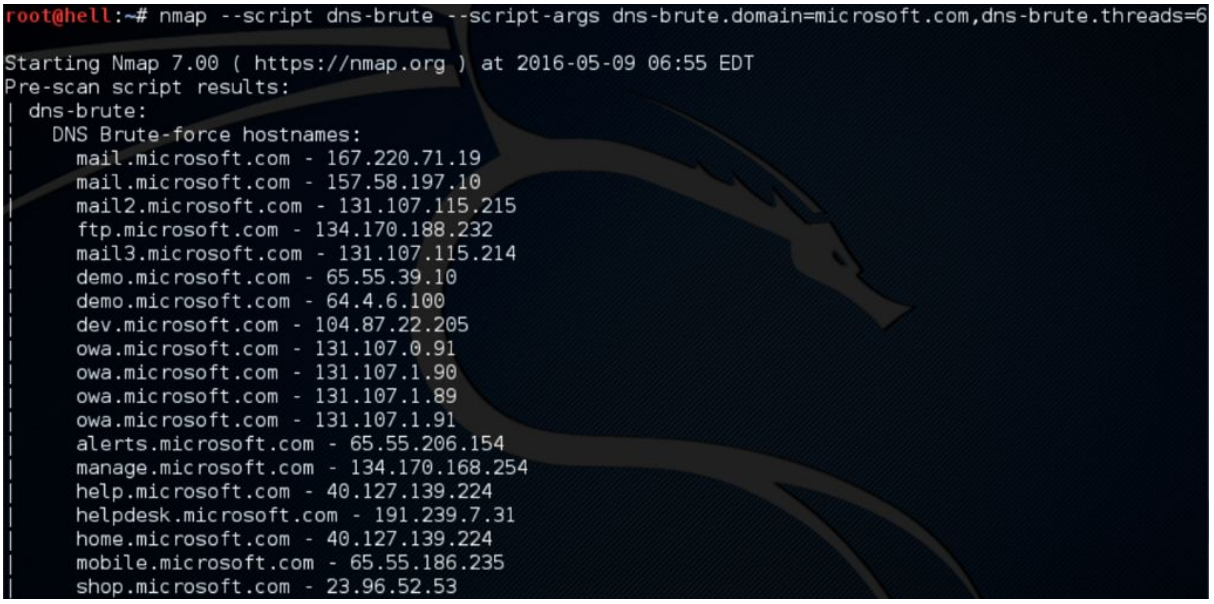


Ilustración 68 - Ejemplo de enumeración de subdominios

En esta ilustración podemos ver un ejemplo de la búsqueda de subdominios utilizando la herramienta “nmap” para el dominio de “microsoft.com” utilizando fuerza bruta. En los resultados podemos apreciar cómo encuentra todos los subdominios asociados al dominio principal de “microsoft.com” junto a sus direcciones IP.

Una vez enumerados todos los subdominios se pueden pasar a los dos puntos siguientes, los cuales son el escaneo de puertos y las capturas de pantalla de los sitios que se encuentran.

1.2. Escaneo de puertos.

Una vez enumerados todos los subdominios el siguiente paso es el escaneo de puertos para identificar más vectores de ataque. Los resultados obtenidos del escaneo de puertos pueden ser el indicativo de la seguridad de una empresa. Por poner un ejemplo, una empresa que ha cerrado todos los puertos excepto el 80 y el 443 (que son los principales puertos de web HTTP y HTTPS) suele ser síntoma de tener una seguridad buena. En cambio una empresa la cual tiene muchos puertos abiertos suele tener una seguridad débil y por lo tanto ser mucho más vulnerable a ataques.

Hay dos herramientas comunes utilizadas para hacer el escaneo de puertos, que son “Nmap” (la cual ha aparecido en la ilustración con el ejemplo) y “Masscan”. Ambas herramientas nos sirven para el propósito de escanear los puertos, sin embargo “nmap” es más antigua y puede ser más lenta que la otra en caso de no saber optimizarla.

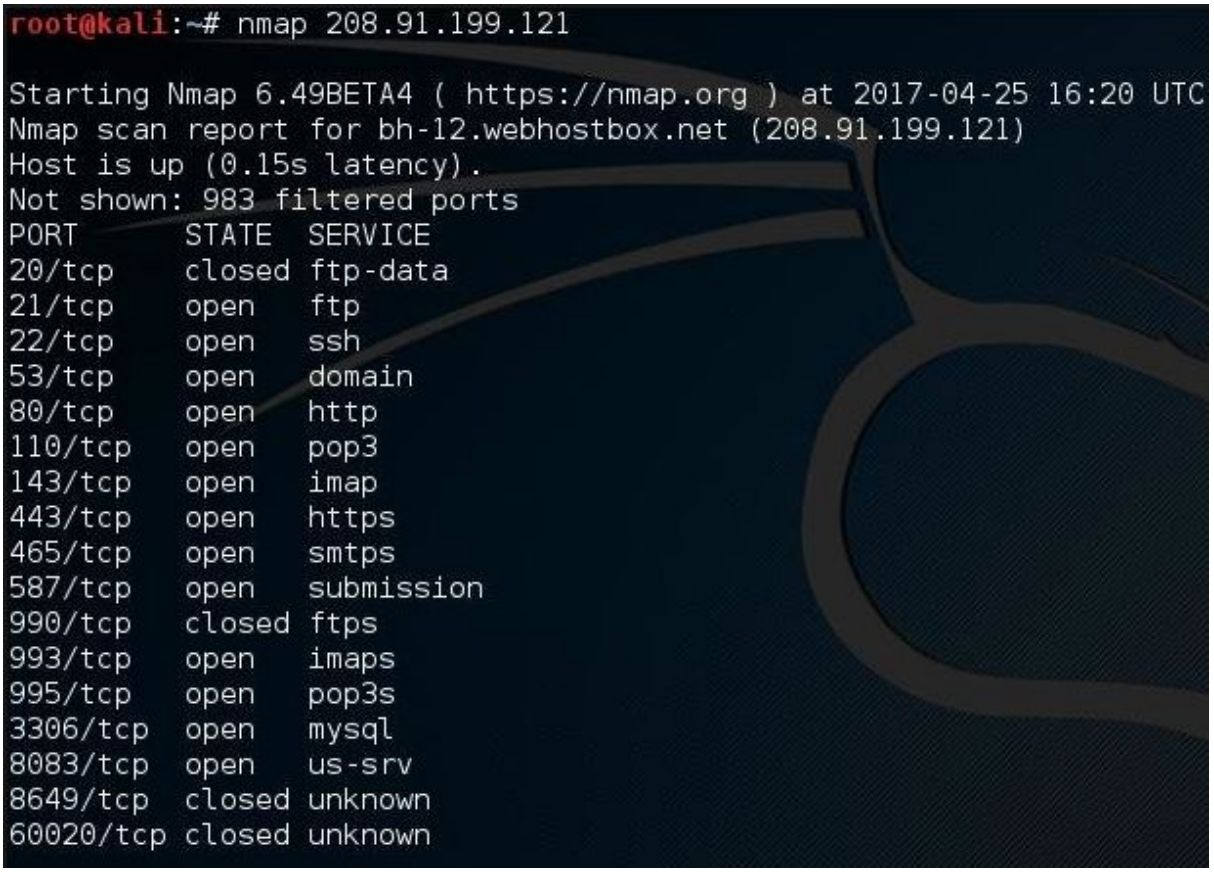


Ilustración 69 - Ejemplo de escaneo de puertos

Un punto a tener en cuenta cuando se está haciendo el escaneo de puertos de una lista de subdominios es la dirección IP de dichos subdominios. Si se encuentra que todos los subdominios tienen un rango de IPs parecido excepto uno, quizás sea interesante investigar este dominio diferente. La diferencia del rango en la IP puede ser sinónimo de una aplicación de terceros la cual no comparte el mismo nivel de seguridad que el resto de aplicaciones de la empresa.

1.3. Capturas de pantalla.

Un buen paso para hacer después de tener la lista de subdominios es hacer capturas de pantalla de estos. Este método es de gran ayuda ya que da una visión global del alcance del programa. Una vez se revisan las capturas de pantalla hay varios puntos para prestar atención. El primero es buscar mensajes de error comunes de servicios que se sabe que están asociados con un subdominio.

El segundo es buscar contenido sensible. Por ejemplo, si todos los subdominios encontrados en "*.asd.example.com" devuelven un error 403 de acceso denegado excepto un subdominio, el cual tiene un login a una página diferente, se debería investigar dicha página ya que es probable que esté implementando un comportamiento propio y distinto del resto de páginas. También se debería vigilar por páginas de logins administrativos, páginas de instalación por defecto, etc.

Hay muchas herramientas distintas las cuales se utilizan para hacer las capturas de pantalla, como por ejemplo "HTTPScreenShot" y "Gowitness". Estas herramientas se pueden utilizar con una lista de direcciones IP las cuales va a hacer las capturas de pantalla, separándolas por grupos según los mensajes de estas, incluyendo también las cabeceras HTTP.

Una vez revisados todos los subdominios y teniendo una imagen global de estos, el siguiente paso es buscar contenido interesante.

1.4. Descubrimiento de contenido.

Hay diferentes formas de realizar la investigación para el descubrimiento de contenido. La primera de todas es intentar descubrir archivos y directorios mediante fuerza bruta, pero el éxito de esta forma depende de la wordlist que se esté usando. Como ya he comentado anteriormente, hay muchas wordlists colgadas en la red creadas por distintas personas para este fin.

Una vez se tiene un listado de nombres de archivos y directorios (wordlist) hay varias herramientas que permiten testear, dados un dominio y un wordlist, la existencia de archivos y directorios junto a la confirmación de la respuesta del servidor. Algunas de estas herramientas podrían ser "Gobuster" y "Burp Suite Pro".

Cuando es necesario ir más allá de la fuerza bruta de archivos y directorios, “Google Dorking” proporciona un descubrimiento de contenido interesante. Una búsqueda de Google Dork es un string de búsqueda el cual utiliza operadores de búsqueda avanzados para buscar información que no está disponible en una página web. Google Dorking (También conocido como Google Hacking) puede devolver información que es difícil de encontrar mediante búsquedas normales. Utilizar esto puede ahorrar mucho tiempo cuando se encuentran parámetros en las URL asociados a vulnerabilidades. Exploit DB tiene una base de datos de Google Dorks para casos de uso específicos.

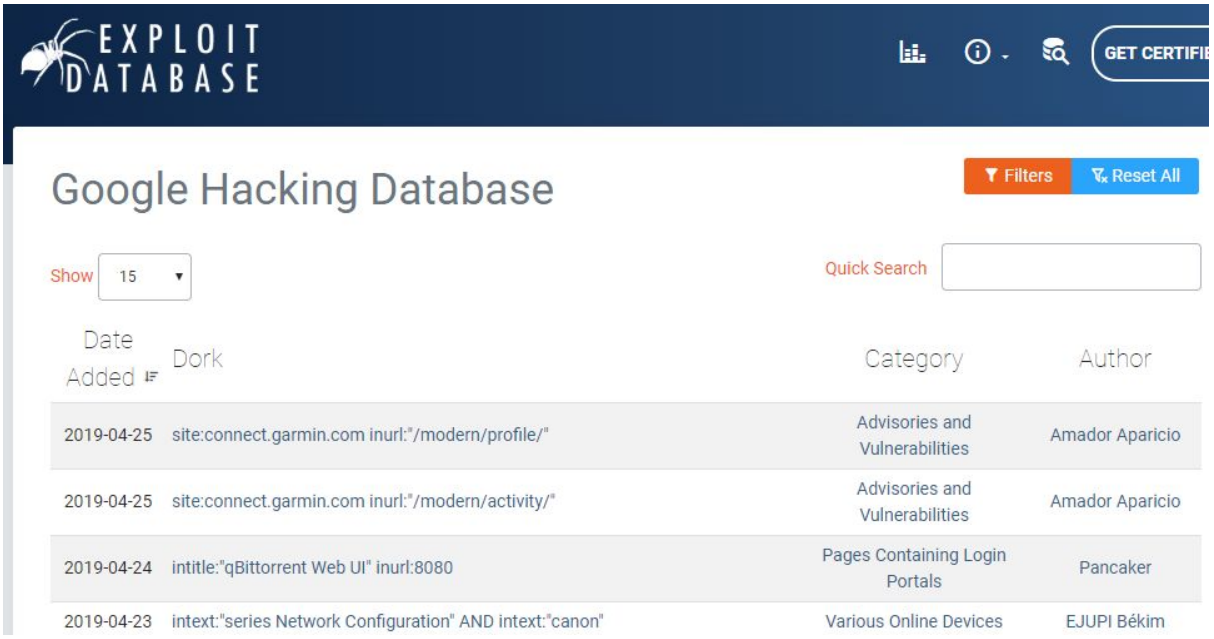


Ilustración 70 - Google Dorks DB en Exploit DB

Otra forma de encontrar contenido interesante es mirar el GitHub de la compañía. Se puede encontrar repositorios de open source o información útil sobre la tecnología que utilizan. Adicionalmente, se puede revisar repositorios de código y encontrar librerías de terceros las cuales utiliza una aplicación. También es posible encontrar un proyecto abandonado o una vulnerabilidad en las aplicaciones o librerías de terceros que afecte el sitio. Los repositorios de código también pueden dar una visión de cómo la compañía ha manejado vulnerabilidades anteriores.

1.5. Errores anteriores.

Una de las últimas etapas del reconocimiento es la búsqueda de errores (bugs) anteriores, mediante informes de hackers, reports de la compañía, CVEs, exploits publicados, foros, etc. Aunque el código esté actualizado no significa que todas las vulnerabilidades estén arregladas. Cuando un error se arregla y se aplica un fix, significa que hay un código nuevo, por lo tanto puede significar que haya más errores en este.

Una vez cubiertas todas las partes del reconocimiento, vamos a pasar a describir el testeo de la aplicación como tal.

6.1.2.2. Testeo de la aplicación

A la hora de testear una aplicación no hay una aproximación concreta que sea mejor o peor. La metodología y las técnicas empleadas van a depender del tipo de aplicación que se está testeando, parecido a que la dimensión del programa define el reconocimiento. Vamos a ver una visión general de las consideraciones e ideas que hay que tener cuando se está testeando un nuevo sitio.

2.1. La pila de tecnología.

Una de las primeras cosas a tener en cuenta cuando se testea la aplicación es identificar las tecnologías que está utilizando. Estas tecnologías pueden ser, por ejemplo: JavaScript frameworks, frameworks de servidor, servicios de terceros, archivos remotos, etc. Existe un plugin para Firefox llamado “Wappalyzer” el cual es genial para identificar estas tecnologías.

Una vez se está en el paso de reconocer las tecnologías del sitio, hay que entender la funcionalidad de este y fijarse en patrones de diseño interesantes. Por poner un ejemplo, un sitio que utiliza AngularJS se puede testear a introducir algún código en concreto a ver si renderiza números. Si un sitio se ha creado con ASP.NET con protección XSS, sería mejor buscar otras vulnerabilidades antes que probar XSS. Con esto quiero decir que es importante conocer las tecnologías que existen y sobre todo ver las que el sitio está utilizando. Un mayor entendimiento de cómo funcionan estas tecnologías puede llevar a un fallo en la seguridad y poder explotarlo.

Varios puntos a tener en cuenta:

- Formatos de contenido que un sitio espera o acepta. Por poner un ejemplo, los archivos XML tienen diferentes estructuras y tamaños y el XML parsing puede ser asociado a vulnerabilidades XXE (External entity Vulnerability). Sitios que aceptan la carga de archivos .docx, .xlsx, .pptx o otros XML son para tener en cuenta.
- Herramientas o servicios de terceros que fácilmente están mal configurados. En caso de leer informes de vulnerabilidades de otros hackers, hay que intentar entender cómo encontraron la vulnerabilidad y cómo aplicarla al testeo.
- Parámetros codificados y cómo una aplicación los maneja. Las rarezas en los parámetros pueden ser indicativo de múltiples servicios que interactúan en el backend, que podrían ser explotados.
- Mecanismos de autenticación implementados personalizados. Pequeñas diferencias en cómo una aplicación maneja la redirección de URLs, codificación y parámetros de estado pueden llevar a vulnerabilidades importantes.

2.2. Mapeo de funcionalidades.

Una vez se enumeran y entienden las diferentes tecnologías utilizadas por el sitios, podemos pasar al mapeo de funcionalidades. En este punto el testeo puede ir de una de las

siguientes formas: Buscar marcadores de vulnerabilidades, definir un objetivo concreto para el testeo o seguir una checklist.

Cuando se está buscando marcadores de vulnerabilidades hay que buscar comportamiento asociado a vulnerabilidades. Por ejemplo, ¿el sitio permite crear webhooks con URLs? Esto puede llevar a vulnerabilidades SSF. ¿Se pueden subir archivos? Donde y como se renderizan estos archivos pueden llevar a una vulnerabilidad de ejecución de código remoto. Es de buena práctica que cuando se encuentra algo interesante parar y empezar el testeo de la aplicación para encontrar algún indicador de alguna vulnerabilidad existente. Puede ser un mensaje devuelto inesperado, un retraso en el tiempo de espera de la respuesta, etc.

Por otro lado, cuando se define y se trabaja para llegar a un objetivo, está bien decidir que hacer antes de testear la aplicación. El objetivo podría ser el encontrar una vulnerabilidad en concreto. Con esta meta se ignoran las otras posibilidades y se centra completamente en el objetivo principal.

Por último, se puede seguir una serie de checklists ya definidos en las metodologías, como por ejemplo la de OWASP. Estas metodologías proporcionan una lista de puntos para el testeo cuando se está haciendo un review de la aplicación. Seguir una checklist ya creada y verificada por cierta clase de organismo o profesionales, puede ayudar a pasar por alto vulnerabilidades por no testear alguna funcionalidad específica o olvidar alguna metodología general.

2.3. Encontrar vulnerabilidades.

En este punto ya se debería tener un amplio conocimiento de la aplicación y el cómo funciona, es hora de empezar a testear. En vez de establecer un objetivo personal o seguir un checklist como se ha podido hacer antes, es recomendable buscar comportamiento que pueda indicar que hay una vulnerabilidad para empezar. Para ello se pueden utilizar herramientas las cuales son escáneres automáticos como puede ser “Burp”. El problema de estas herramientas es que, entre otras cosas hace mucho ruido y las aplicaciones no lo permiten usar.

En caso de empezar el testeo y no encontrar nada interesante durante el mapeo de funcionalidades es recomendable empezar a usar la web como un cliente. Crear contenido, usuarios, equipos, subir archivos, lo que sea que permita hacer la aplicación. Mientras se está haciendo este apartado, es común introducir “payloads” donde se acepte la entrada de contenido, para buscar anomalías o comportamiento inesperado del sitio. Estos payloads se componen de una serie de caracteres especiales para intentar romper la integridad del sitio y lo que se renderiza. Este tipo de payloads se llaman polyglot.

Asimismo mientras se crea contenido como usuario o cliente y este se renderiza en un panel de administrador, se puede utilizar otro tipo de payload destinado a encontrar XSS

(Cross-site scripting). Por otro lado, si la aplicación está utilizando una plantilla, se pueden introducir payloads asociados con dicha plantilla.

Vamos a dar un repaso a algunas de las vulnerabilidades a tener en cuenta y en que deberíamos fijarnos para detectarlas.

- CSRF. Los tipos de peticiones HTTP que cambian datos y si tienen CSRF tokens que los estén validando.
- IDORs. Donde haya parámetros de ID que puedan ser manipulados.
- Application logic. Oportunidades para repetir peticiones desde dos cuentas de usuario separadas.
- XXE. Cualquier XML que acepte peticiones HTTP.
- Open redirects. Cualquier URL que tenga un parámetro de redireccionamiento.
- CRLFs, XSS, etc. Cualquier petición que reproduzca parámetros URL en la respuesta.
- SQLi. Cuando se agrega algún parámetro y cambia la respuesta.
- RCEs. Cualquier tipo de carga de archivo o manipulación de imagen.
- Race conditions. Cuando hay retraso en el procesado de datos.

6.2. VULNERABILIDADES

En el apartado de introducción se ha hablado que mediante el informe anual de la empresa Acunetix de 2019, aproximadamente el 60% de las aplicaciones web dispone de vulnerabilidades.

Viendo otro informe, concretamente el "Automated Code Analysis: Web Application Vulnerabilities in 2017", todas las aplicaciones web que han sido analizadas por la empresa Positive Technologies contenían vulnerabilidades de distinta gravedad. Además del 94% de las aplicaciones contenían una vulnerabilidad muy grave, y el 85% tenían una vulnerabilidad explotable.

Para hacer este informe, Positive Technologies evaluó 33 aplicaciones. Algunas de ellas estaban disponibles públicamente en el momento del análisis, mientras que otras funcionaban solo internamente para funciones comerciales. Todas las aplicaciones eran susceptibles a errores de código o debilidades de configuración.

De todas las vulnerabilidades identificadas, los fallos de cross-site scripting (XSS) fueron los más frecuentes, presentes en el 82% de las aplicaciones, seguidas por HTTP response splitting con un 58% y lectura de archivos arbitrarios con un 52%.

Además de permitir ataques contra usuarios, las vulnerabilidades descubiertas en el 70% de las aplicaciones sentaban las bases para lanzar ataques de denegación de servicio (DoS). Esta amenaza de nivel medio era la más común, seguida de otras cuatro de alta gravedad, incluida la lectura arbitraria de archivos (61%), el control del sistema operativo (55%), el acceso no autorizado a la base de datos (45%) y la eliminación o modificación de archivos del servidor (42%).

Podemos ver en el siguiente gráfico la cantidad de vulnerabilidades por tipo desde 1999 hasta 2019.

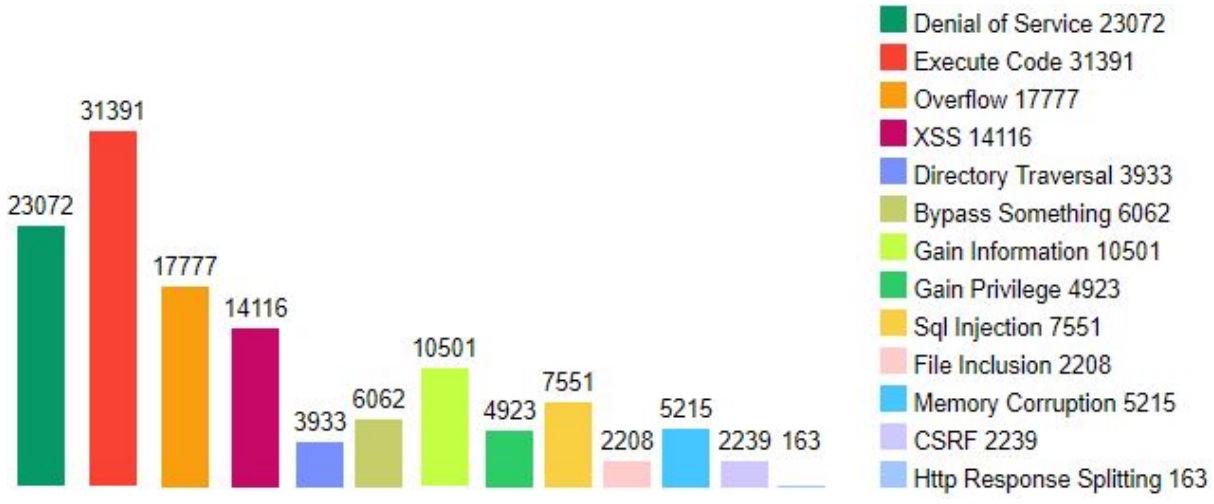


Ilustración 71 - Vulnerabilidades por tipo - Fuente: CVE Details

En el transcurso de casi 20 años las vulnerabilidades más frecuentes han cambiado mucho. Para hacernos una idea de a que nos podemos enfrentar en los últimos tiempos, podemos ver una gráfica con las vulnerabilidades más frecuentes en 2018.

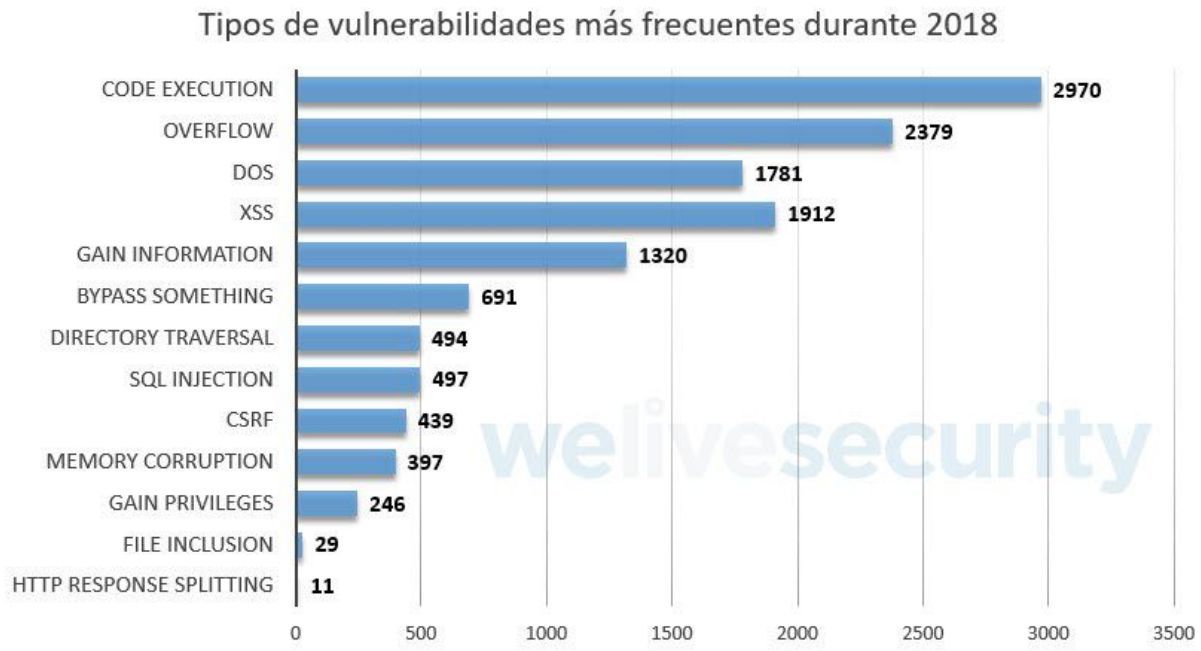


Ilustración 72 - Vulnerabilidades más frecuentes en 2018 - Fuente: welivesecurity.com

Podemos ver que de las vulnerabilidades más frecuentes en 2018 fueron la ejecución de código (23%), ataques de overflow (18%) y de XXS (15%). Cabe destacar también que el 79% de las vulnerabilidades de ejecución de código eran graves. No es de extrañar entonces que la explotación de vulnerabilidades sea uno de los vectores de compromiso más utilizados.

Podemos ver más datos sobre vulnerabilidades web mediante el documento OWASP Top 10. El Top 10 de OWASP es un informe actualizado periódicamente que describe los problemas de seguridad para la seguridad de las aplicaciones web, y se centra en los 10 riesgos más críticos. El informe es elaborado por un equipo de expertos en seguridad de todo el mundo. OWASP se refiere al Top 10 como un "documento de concienciación" y recomienda que todas las empresas incorporen el informe en sus procesos para minimizar y mitigar los riesgos de seguridad.

OWASP Top 10:
https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

En este apartado se va dar una visión general de las vulnerabilidades más importantes que existen. Cada una de las vulnerabilidades explicadas contiene un código QR y un enlace los cuales derivan a un vídeo explicativo de dicha vulnerabilidad, grabados exclusivamente para este trabajo.

6.2.1. CRLF Injection

[Enlace al vídeo]

Esta vulnerabilidad ocurre cuando un atacante es capaz de inyectar datos en una petición a un servidor, debido a la falta de filtrado de datos de entrada por parte del mismo. En este caso, la web afectada permite pasar directamente valores a los campos de respuesta (Location, Set-Cookie..) sin sanearlos lo que a su vez nos permite diversos tipos de ataque como XSS, Cache-Poisoning, Cache-based defacement,page injection..

Las siglas CRLF vienen de CR (Carriage return o retorno de carro) y de LF (Line Feed o salto de línea). Ambos son caracteres "invisibles" que indican el final de línea para importantes protocolos como HTTP, MIME o NNTP. En código ASCII el CR tiene valor 13 y el LF valor 10 y a veces se escriben de la manera siguiente "r\n\".

El ataque CRLF ocurre cuando un atacante consigue, modificando un parámetro HTTP o URL, inyectar el código correspondiente a CRLF al servido, lo que le permite manipular muchos aspectos de la propia petición.

El método más efectivo para evitar estos es aplicar filtros sobre cualquier dato que entre a nuestro servidor o a nuestra web, examinando y saneando las diferentes variables que puedan aprovecharse de CRLF. Para ayudarnos en la tarea de localizar puntos vulnerables en nuestra web siempre podemos utilizar scanners de vulnerabilidades automatizados que suelen encontrar este tipo de vulnerabilidades.

6.2.2. Cross-Site Request Forgery

[Enlace al vídeo]

El ataque Cross-Site Request Forgery (CSRF) es una ataque que a diferencia del Cross-Site Scripting, se basa en la confianza de una web en el usuario que va a ser víctima del ataque. Aprovechando esa confianza y a través de los permisos efectivos en ese momento de ese usuario en la web, se pueden ejecutar scripts para que la víctima ejecute acciones sin ser consciente de ello.

Este tipo de ataques se centra no tanto en el robo de datos ya que el atacante no puede ver la respuesta a la petición manipulada sino en el cambio de estado de los mismos. Dependiendo del nivel de acceso de la víctima el atacante podría desde cambiar datos de la cuenta de la víctima hasta tomar control de la aplicación web pasando por transferir dinero,etc…

Para la mayoría de servicios web, las peticiones incluyen las credenciales del usuario autenticado ya sea a través de cookies de sesión, IP, credenciales de dominio,etc... por ello si la aplicación web no cuenta con las medidas para contrarrestar CSRF, no tendría manera de distinguir entre las peticiones legítimas de la víctima y las que a través de un ataque CSRF se lancen en su nombre en ese mismo momento en el que está autenticado. Aunque la mayoría de scripts actualmente de CSRF son estáticos existen teóricamente también ataques dinámicos que pueden adaptarse a las sesiones que tengamos abiertas en nuestro navegador.

Es posible incluso almacenar un ataque CSRF en el propio servidor, a través de un tag de IMG o IFRAME en el código, esto se denomina "Stored CSRF Flaws". En este caso es más probable que el ataque sea exitoso ya que la víctima estará autenticada en el momento de ejecutar ese código en esa determinada web. Este tipo de ataque se ha encontrado principalmente en foros donde cualquier usuario puede postear fácilmente imágenes y tendrá un gran número de víctimas potenciales de su script.

Otra variante de un ataque CSRF sería la de “Login CSRF” en este caso un atacante podría forzar a una víctima a loguearse con las credenciales del atacante en un servicio, en el caso que la víctima no detecte esto continuará trabajando normalmente y a posteriori el atacante

podrá entrar con sus propias credenciales y ver actividad, archivos,etc.. que la víctima haya podido dejar en el servicio.

La prevención de este ataque se realiza añadiendo métodos de autenticación adicionales para las peticiones. De esta manera se pueden detectar peticiones no autorizadas. Algunas de las técnicas que implementan esto son las siguientes:

Synchronizer token pattern

Esta técnica incluye un token único y secreto para cada petición web (o sesión) ,el token se genera aleatoriamente y es comprobado por el servidor. De esta manera un atacante no puede conocer cuál es el valor correcto y por tanto no puede enviar una petición efectiva al servidor.

Cookie-to-Header Token

Este sistema es usado por la mayoría de aplicaciones web basadas en JavaScript. En el momento del login la aplicación web crea una cookie con un token único y aleatorio para la sesión. El JavaScript en el cliente lee el valor y lo añade a la cabecera de cada petición HTTP. La seguridad de este método se basa en que solo el JavaScript original puede leer este token y por tanto no se podrá crear una cabecera HTTP por otro script utilizandolo.

6.2.3. Cross-Site Scripting

[Enlace al vídeo]

Cross-site scripting (XSS) es un tipo de vulnerabilidad informática o agujero de seguridad típico de las aplicaciones web, que puede permitir a una tercera persona inyectar en páginas web visitadas por el usuario código JavaScript o en otro lenguaje similar (como por ejemplo VBScript), se puede evitar usando medidas como CSP Política del mismo origen.

Es posible encontrar una vulnerabilidad de Cross-Site Scripting en aplicaciones que tengan entre sus funciones presentar la información en un navegador web u otro contenedor de páginas web. Sin embargo, no se limita a sitios web disponibles en Internet, ya que puede haber aplicaciones locales vulnerables a XSS, o incluso el navegador en sí.

XSS es un vector de ataque que puede ser utilizado para robar información delicada, secuestrar sesiones de usuario, y comprometer el navegador, subyugando la integridad del sistema. Las vulnerabilidades XSS han existido desde los primeros días de la web.

Esta situación es usualmente causada al no validar correctamente los datos de entrada que son usados en cierta aplicación, o no sanear la salida adecuadamente para su presentación como página web.

Esta vulnerabilidad puede estar presente de las siguientes formas:

Directa (también llamada Persistente): este tipo de XSS comúnmente filtrado, y consiste en insertar código HTML peligroso en sitios que lo permitan; incluyendo así etiquetas como <script> o <iframe>.

Indirecta (también llamada Reflejada): este tipo de XSS consiste en modificar valores que la aplicación web utiliza para pasar variables entre dos páginas, sin usar sesiones y sucede cuando hay un mensaje o una ruta en la URL del navegador, en una cookie, o cualquier otra cabecera HTTP (en algunos navegadores y aplicaciones web, esto podría extenderse al DOM del navegador).

6.2.4. HTML Injection

[Enlace al vídeo]

La inyección de HTML es un tipo de problema de inyección que se produce cuando un usuario puede controlar un punto de entrada y puede inyectar código HTML en una página web vulnerable. Esta vulnerabilidad puede tener muchas consecuencias, como por ejemplo la divulgación de las cookies de sesión de un usuario, que podrían usarse para hacerse pasar por la víctima o, más generalmente, puede permitir que el atacante modifique el contenido de la página que ven las víctimas.

Esta vulnerabilidad se produce cuando la entrada del usuario no está correctamente saneada y la salida no está codificada. Una inyección permite al atacante enviar una página HTML maliciosa a la víctima. El explorador objetivo no podrá distinguir (confiar) la legitimidad de las partes maliciosas y, en consecuencia, analizará y ejecutará todo como legítimo en el contexto de la víctima.

Existe una amplia gama de métodos y atributos que podrían usarse para representar contenido HTML. Si estos métodos se proporcionan con una entrada no confiable, entonces existe un alto riesgo de XSS, específicamente una inyección de HTML. El código HTML malicioso se puede inyectar, por ejemplo, a través de innerHTML, que se utiliza para representar el código HTML insertado por el usuario. Si las cadenas no se sanean correctamente, el problema podría llevar a una inyección de HTML basada en XSS. Otro método podría ser document.write().

La propiedad innerHTML establece o devuelve el HTML interno de un elemento. Un uso incorrecto de esta propiedad podría permitir que un atacante pueda inyectar código HTML malicioso.

6.2.5. Insecure Direct Object References

[Enlace al vídeo]

Insecure Direct Object References o también llamadas IDOR se producen cuando una aplicación proporciona acceso directo a objetos en función de la entrada proporcionada por el usuario. Como resultado de esta vulnerabilidad, los atacantes pueden omitir la autorización y acceder a los recursos en el sistema directamente, por ejemplo, registros o archivos de bases de datos.

Las IDOR permiten a los atacantes eludir la autorización y acceder a los recursos directamente al modificar el valor de un parámetro utilizado para apuntar directamente a un objeto. Dichos recursos pueden ser entradas de la base de datos que pertenecen a otros usuarios, archivos en el sistema y más. Esto se debe al hecho de que la aplicación toma la entrada proporcionada por el usuario y la utiliza para recuperar un objeto sin realizar suficientes controles de autorización.

Para probar esta vulnerabilidad, el usuario primero debe mapear todas las ubicaciones en la aplicación donde se usa la entrada del usuario para hacer referencia a objetos directamente. Por ejemplo, ubicaciones donde se utiliza la entrada del usuario para acceder a una fila de la base de datos, un archivo, páginas de aplicaciones y más. A continuación, el usuario debe modificar el valor del parámetro utilizado para hacer referencia a los objetos y evaluar si es posible recuperar objetos que pertenecen a otros usuarios o si no pasar por alto la autorización.

La mejor manera de probar las referencias directas de objetos sería tener al menos dos (a menudo más) usuarios para cubrir diferentes objetos y funciones. Por ejemplo, dos usuarios que tienen acceso a diferentes objetos (como información de compra, mensajes privados, etc.) y (si corresponde) usuarios con diferentes privilegios (por ejemplo, usuarios administradores) para ver si hay referencias directas a la funcionalidad de la aplicación. Al tener varios usuarios.

6.2.6. Open Redirect Vulnerabilities

[Enlace al vídeo]

Una Open Redirect Vulnerability es cuando una aplicación o servidor web utiliza un enlace enviado por el usuario para redirigir al usuario a un sitio web o página determinada. Aunque parezca una acción inofensiva, permitir que un usuario decida a qué página desea ser redirigido, si se explota, esta técnica puede tener un impacto grave, especialmente cuando se combina con otras vulnerabilidades.

Dado que el nombre de dominio en una URL suele ser el único indicador para que un usuario reconozca un sitio web legítimo de uno no legítimo, un atacante puede abusar de esta confianza para explotar una vulnerabilidad de redireccionamiento abierta en el sitio web vulnerable y redirigir al usuario a un página maliciosa para ejecutar más ataques.

Los impactos de esta vulnerabilidad pueden ser muchos, y varían desde el robo de información y las credenciales hasta la redirección a sitios web maliciosos que contienen contenido controlado por un atacante, que en algunos casos incluso causa ataques XSS. Por lo tanto, aunque una redirección abierta puede parecer inofensiva al principio, los impactos de la misma pueden ser graves en caso de que puedan explotarse.

La forma más fácil y efectiva de prevenir esta vulnerabilidad sería no permitir que el usuario controle a dónde lo redirige la página. Si tiene que redirigir al usuario basándose en las URL, siempre debe usar una ID que se resuelva internamente a la URL correspondiente. Si se quiere que el usuario pueda emitir redirecciones, debe usar una página de redirección que requiera que el usuario haga clic en el enlace en lugar de simplemente redirigirlos. También debe verificar que la URL comience con http:// o https:// y también invalidar todas las demás URL para evitar el uso de URI maliciosos como javascript:.

6.2.7. Remote Code Execution

[Enlace al vídeo]

Remote Code Execution es una vulnerabilidad que puede ser explotada si la entrada del usuario se inyecta en un archivo o una cadena y se ejecuta (evalúa) mediante el analizador del lenguaje de programación. Por lo general, este comportamiento no está pensado por el desarrollador de la aplicación web. Una evaluación remota de código o remote code execution puede derivar a un compromiso total de la aplicación web vulnerable y también del servidor web. Es importante tener en cuenta que casi todos los lenguajes de programación tienen funciones de evaluación de códigos.

Una evaluación de código puede ocurrir si permite la entrada del usuario dentro de las funciones que están evaluando el código en el lenguaje de programación respectivo. Esto se puede implementar a propósito, por ejemplo, para acceder a las funciones matemáticas del lenguaje de programación para crear una calculadora, o accidentalmente porque no se espera una entrada controlada por el usuario del desarrollador dentro de esas funciones. Generalmente no se aconseja hacerlo. De hecho, se considera una mala práctica utilizar la evaluación de códigos.

Un atacante que puede ejecutar una falla de este tipo generalmente puede ejecutar comandos con los privilegios del lenguaje de programación o el servidor web. En muchos lenguajes puede emitir comandos del sistema, escribir, eliminar o leer archivos o conectarse a bases de datos.

Como regla general, hay que evitar utilizar la entrada del usuario dentro del código evaluado. La mejor opción sería no usar funciones como eval. Se considera una mala práctica y, con frecuencia, puede evitarse por completo. Además, nunca debe permitir que un usuario edite el contenido de los archivos que pueden ser analizados por los lenguajes respectivos. Esto incluye no permitir que un usuario decida el nombre y las extensiones de los archivos que puede cargar o crear en la aplicación web.

6.2.8. Server Side Request Forgery

[Enlace al vídeo]

En un ataque SSRF, el atacante puede abusar de la funcionalidad del servidor para leer o actualizar los recursos internos. El atacante puede suministrar o modificar una URL a la que el código que se ejecuta en el servidor leerá o enviará datos, y al seleccionar cuidadosamente las URL, el atacante podrá leer la configuración del servidor, como los metadatos de AWS, y conectarse a servicios internos como a bases de datos con http habilitado o realizar solicitudes posteriores a los servicios internos que no están destinados a ser expuestos.

La aplicación de destino puede tener una funcionalidad para importar datos desde una URL, publicar datos en una URL o leer datos de una URL que pueda manipularse. El atacante modifica las llamadas a esta funcionalidad suministrando una URL completamente diferente o manipulando cómo se construyen las URL (recorrido de la ruta, etc.).

Cuando la solicitud manipulada va al servidor, el código del lado del servidor recoge la URL manipulada e intenta leer los datos a la URL manipulada. Al seleccionar las URL de destino, el atacante puede leer datos de servicios que no están directamente expuestos en Internet:

- Metadatos del servidor en la nube: los servicios en la nube como AWS proporcionan una interfaz REST donde se pueden extraer configuraciones importantes y, a veces incluso claves de autenticación.
- Interfaces HTTP de la base de datos: la base de datos NoSQL como MongoDB proporciona interfaces REST en los puertos HTTP. Si se espera que la base de datos solo esté disponible para uso interno, la autenticación puede estar deshabilitada y el atacante puede extraer datos.
- Interfaces REST internas.
- Archivos: el atacante puede leer archivos utilizando file:// URIs.

6.2.9. SQL Injection

[Enlace al vídeo]

Inyección SQL es un método de infiltración de código intruso que se vale de una vulnerabilidad presente en una aplicación en el nivel de validación de las entradas para realizar operaciones sobre una base de datos.

El origen de la vulnerabilidad radica en la incorrecta comprobación o filtrado de las variables utilizadas en un programa que contiene, o bien genera, código SQL. Es, de hecho, un error de una clase más general de vulnerabilidades que puede ocurrir en cualquier lenguaje de programación o script que esté embebido dentro de otro.

Se conoce como Inyección SQL, indistintamente, al tipo de vulnerabilidad, al método de infiltración, al hecho de incrustar código SQL intruso y a la porción de código incrustado.

Se dice que existe o se produjo una inyección SQL cuando, de alguna manera, se inserta o "inyecta" código SQL invasor dentro del código SQL programado, a fin de alterar el funcionamiento normal del programa y lograr así que se ejecute la porción de código "invasor" incrustado, en la base de datos.

6.2.10. Subdomain Takeover

[Enlace al vídeo]

Un subdomain takeover es un proceso de registro de un nombre de dominio no existente para obtener el control sobre otro dominio. El escenario más común de este proceso es el siguiente:

1. El nombre de dominio (por ejemplo, sub.example.com) usa un registro CNAME para otro dominio (por ejemplo, sub.example.com CNAME anotherdomain.com).
2. En algún momento en el tiempo, anotherdomain.com caduca y está disponible para su registro por cualquier persona.
3. Como el registro CNAME no se elimina de la zona DNS de example.com, cualquier persona que registre anotherdomain.com tendría control total sobre sub.example.com hasta que el registro DNS esté presente.

Las implicaciones de un subdomain takeover pueden ser bastante significativas. Los atacantes pueden enviar correos electrónicos de suplantación de identidad (phishing) desde el dominio legítimo, realizar scripts entre sitios (XSS) o dañar la reputación de la marca asociada al dominio.

El subdomain takeover no se limita a los registros CNAME. Los registros NS, MX e incluso A también se ven afectados.

6.2.11. Template Injection

[Enlace al vídeo]

Los template engines son generalmente utilizados por las aplicaciones web para presentar datos dinámicos a través de páginas web y correos electrónicos. La incorporación incorrecta de las entradas del usuario en las plantillas permite utilizar template injection, una vulnerabilidad frecuentemente crítica que es extremadamente fácil de confundir con XSS, o que se omite por completo. A diferencia de XSS, la inyección de plantillas se puede usar para atacar directamente las partes internas de los servidores web y, a menudo, obtener la ejecución remota de código (RCE), convirtiendo cada aplicación vulnerable.

Template injection puede surgir tanto a través del error del desarrollador como a través de la exposición intencional de las plantillas en un intento por ofrecer una funcionalidad rica, como lo hacen comúnmente las wikis, blogs, aplicaciones de marketing y sistemas de gestión de contenido. La inyección intencional de plantillas es un caso de uso tan común que muchos motores de plantillas ofrecen un modo de "espacio aislado" para este propósito expreso.

6.2.12. XML External Entity Vulnerability

[Enlace al vídeo]

Un ataque de entidad externa XML es un tipo de ataque contra una aplicación que analiza la entrada XML. Este ataque ocurre cuando la entrada XML que contiene una referencia a una entidad externa es procesada por un analizador XML débilmente configurado. Este ataque puede llevar a la divulgación de datos confidenciales, denegación de servicio, falsificación de solicitudes del lado del servidor, escaneo de puertos desde la perspectiva de la máquina donde se encuentra el analizador y otros impactos del sistema.

El estándar XML 1.0 define la estructura de un documento XML. El estándar define un concepto llamado entidad, que es una unidad de almacenamiento de algún tipo. Hay diferentes tipos de entidades, la entidad externa, que puede acceder al contenido local o remoto a través de un identificador del sistema declarado. Se supone que el identificador del sistema es un URI que el procesador XML puede anular (acceder) al procesar la entidad. El procesador XML reemplaza las ocurrencias de la entidad externa nombrada con los contenidos que el identificador del sistema hace referencia. Si el identificador del sistema contiene datos contaminados y el procesador XML no hace referencia a estos datos contaminados, el procesador XML puede revelar información confidencial que normalmente no es accesible por la aplicación. Los vectores de ataque similares aplican el uso de DTD externas, hojas de estilo externas, esquemas externos, etc. que, cuando se incluyen, permiten ataques de estilo de inclusión de recursos externos similares.

Los ataques pueden incluir la divulgación de archivos locales, que pueden contener datos confidenciales, como contraseñas o datos privados del usuario, mediante el uso de archivos, esquemas o rutas relativas en el identificador del sistema. Dado que el ataque ocurre en relación con la aplicación que procesa el documento XML, un atacante puede usar esta aplicación confiable para pivotar hacia otros sistemas internos, posiblemente revelando otro contenido interno a través de solicitudes http (s) o lanzando un ataque CSRF a cualquier servicio interno no protegido. Otros ataques pueden acceder a recursos locales que pueden no dejar de devolver datos, lo que podría afectar la disponibilidad de la aplicación si no se liberan demasiados procesos o subprocesos.

7. CONCLUSIONES Y TRABAJO FUTURO

Las amenazas y los riesgos existentes en los sistemas y en las aplicaciones varían de forma constante, aparecen nuevas vulnerabilidades y nuevos atacantes que ponen en jaque a las organizaciones.

Este proyecto consta de estudiar las distintas vulnerabilidades existentes a día de hoy y mediante unos casos prácticos testear estas diferentes vulnerabilidades para ver como afecta y qué consecuencias tienen en una aplicación web.

Como resultado de las pruebas de pentesting realizadas podemos decir que los programadores somos humanos y cometemos errores, los cuales se transforman en vulnerabilidades y vectores de ataque en las aplicaciones web que creamos. Hasta el mínimo detalle puede hacer que una página sea insegura y presente problemas. Para evitar esto hay que volver a dar un repaso al código para detectar dichos fallos y poder corregirlos antes de que sean explotados. Otra forma de proceder puede ser realizar pentesting contra la propia página web y pasar algún software de detección de vulnerabilidades automático y así ahorrar mucho tiempo.

Este apartado de vulnerabilidades web y en general la ciberseguridad no se estudia en todo el grado de informática y, desde mi punto de vista, es algo muy importante como para no invertir un mínimo de tiempo en unos estudios superiores.

Este proyecto se puede ampliar como trabajo futuro estudiando más detalladamente las vulnerabilidades u otras menos conocidas, así como realizar más casos de pentesting, un estudio detallado de estos, un informe de vulnerabilidades, etc. Como bien se ha estudiado la página web creada en una asignatura optativa, se podría realizar el mismo proceso en otras aplicaciones o webs creadas en otros créditos del grado para detectar los distintos errores y poder introducir el código necesario para arreglarlos.

8. REFERENCIAS

HackerOne
https://www.hackerone.com/

Web Hacking 101 - Peter Yaworsky [*Libro*]

Owasp Top Ten - OWASP
https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

CVE Details
https://www.cvedetails.com/

Open-Source Security Testing Methodology Manual
http://www.isecom.org/research/

SiteLock Blog
https://www.sitelock.com/blog/

Exploit Database
https://www.exploit-db.com/

PORTSWIGGER - Web security
https://portswigger.net/web-security

Hacking Ético Blog
https://hacking-etico.com/