Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

USO DE LA APLICACIÓN WEBGOAT PARA EL APRENDIZAJE DE LAS VULNERABILIDADES MÁS EXPLOTADAS EN APLICACIONES WEB

Brayner Estuardo Navarro Vásquez
Asesorado por el Ing. Sergio Eduardo Mancilla Escobar

Guatemala, noviembre de 2018

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

USO DE LA APLICACIÓN WEBGOAT PARA EL APRENDIZAJE DE LAS VULNERABILIDADES MÁS EXPLOTADAS EN APLICACIONES WEB

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

BRAYNER ESTUARDO NAVARRO VÁSQUEZ
ASESORADO POR EL ING. SERGIO EDUARDO MANCILLA ESCOBAR

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, NOVIEMBRE DE 2018

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Pedro Antonio Aguilar Polanco
VOCAL I	Ing. Angel Roberto Sic García
VOCAL II	Ing. Pablo Christian de León Rodríguez
VOCAL III	Inga. José Milton de León Bran
VOCAL IV	Br. Oscar Humberto Galicia Nuñez
VOCAL V	Br. Carlos Enrique Gómez Donis
SECRETARIA	Inga. Lesbia Magalí Herrera López

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Pedro Antonio Aguilar Polanco
EXAMINADOR	Ing. César Augusto Fernández Cáceres
EXAMINADOR	Ing. Herman Igor Véliz Linares
EXAMINADOR	Ing. William Estuardo Escobar Argueta
SECRETARIA	Inga. Lesbia Magalí Herrera López

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

USO DE LA APLICACIÓN WEBGOAT PARA EL APRENDIZAJE DE LAS VULNERABILIDADES MÁS EXPLOTADAS EN APLICACIONES WEB

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha abril de 2018.

Brayner Estuardo Navarro Vásquez

Guatemala, 05 de julio de 2018

Ingeniero Carlos Azurdia
Tutor de trabajos de graduación
Escuela de Ciencias y Sistemas
Facultad de Ingeniería

Respetable ingeniero Azurdia:

Por este medio informo, que como asesor del trabajo de graduación del estudiante universitario de la carrera de Ingeniería en Ciencias y Sistemas, Brayner Estuardo Navarro Vásquez, carné: 201113843, DPI :2274288701202, hago constar que ha finalizado todos los capítulos del trabajo de investigación titulado: "**USO DE LA APLICACIÓN WEBGOAT PARA EL APRENDIZAJE DE LAS VULNERABILIDADES MÁS EXPLOTADAS EN APLICACIONES WEB**", el cuál he tenido la oportunidad de revisar y doy mi aprobación al mismo.

Agradeciendo su atención a la presente,

Atentamente,

Msc. Sergio Eduardo Mancilla Escobar
Asesor de trabajo de graduación
Colegiado: 13,412
Sergio Eduardo Mancilla Escobar
Ingeniero en Ciencias y Sistemas
Colegiado No. 13412

Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 31 de julio de 2018

Ingeniero
Marlon Antonio Pérez Türk
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Pérez:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **BRAYNER ESTUARDO NAVARRO VÁSQUEZ** con carné **201113843** y **CUI 2274 28870 1202,** titulado **"USO DE LA APLICACIÓN WEBGOAT PARA EL APRENDIZAJE DE LAS VULNERABILIDADES MÁS EXPLOTADAS EN APLICACIONES WEB"** y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo aprobado.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación

USAC · FACULTAD DE INGENIERÍA EN CIENCIAS Y SISTEMAS · COORDINACION TESIS

ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÌA
ESCUELA DE INGENIERÍA EN
CIENCIAS Y SISTEMAS
TEL: 24188000 Ext. 1534

El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del asesor con el visto bueno del revisor y del Licenciado en Letras, del trabajo de graduación, ***"USO DE LA APLICACIÓN WEBGOAT PARA EL APRENDIZAJE DE LAS VULNERABILIDADES MÁS EXPLOTADAS EN APLICACIONES WEB"*** *realizado por el estudiante, BRAYNER ESTUARDO NAVARRO VÁSQUEZ, aprueba el presente trabajo y solicita la autorización del mismo.*

"ID Y ENSEÑAD A TODOS"

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION DE INGENIERIA EN CIENCIAS Y SISTEMAS

Ing. Marlon Antonio Pérez Türk
Director
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 09 de noviembre de 2018

ACTO QUE DEDICO A:

Jehová Dios	Por darme la vida, conocimiento y la sabiduría necesaria para completar esta parte importante de mi vida.
Mi padre	Mynor Rubén Navarro (q. e. p. d.), por ser el mayor ejemplo en mi vida y que aún guía con los consejos y experiencias enseñados. Además por todo el apoyo y el amor brindado en cada momento de mi vida.
Mi madre	Ingrid Anabella Vásquez, por su amor y apoyo incondicional en cada momento de mi vida, de la cual ha sido parte fundamental, debido a los consejos y ejemplo proporcionado.
Mis hermanas	Katherine y Jennifer Navarro Vásquez, por permanecer a mi lado cada momento que he vivido y siempre estar apoyándome independientemente del tipo de situación que enfrente.
Ing. Sergio Mancilla	Por el valioso apoyo brindado durante la elaboración de este trabajo y consejos que me ayudaron a llegar a este punto.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Por ser mi casa de estudios, la cual me brindó la oportunidad de ser un profesional de éxito.
Facultad de Ingeniería	Por la oportunidad de ser parte de ella y desarrollarme como profesional.
Mis catedráticos	Por los conocimientos transmitidos a lo largo de mi carrera.
Mis amigos y compañeros	Que me apoyaron y con quienes compartí un sinfín de experiencias a lo largo de mi carrera, tanto buenas como malas.
Mis familiares	Abuelos, abuelas, tíos, tías, primos y primas por siempre estar atentos a mi avance en la carrera y sus distintas formas de apoyo.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

GLOSARIO

Arañas Comúnmente conocidas como *spiders*. Programa informático que inspecciona páginas web de forma metódica y automatizada.

ASCII Sistema de codificación de caracteres alfanuméricos en el que se asigna un número del 0 al 127.

Black-box Caja negra. Término usado para referirse a que no se conoce el detalle del funcionamiento de un proceso, solo el resultado que ofrece.

Cookies Información enviada desde un sitio web, que se almacena en el navegador web. Por lo regular contiene las preferencias del cliente.

DTD Definición de tipo de documento.

HTML *Hyper Text Markup Language* (lenguaje de marcas de hipertexto).

HTTP *Hyper Text Transfer Protocol* (protocolo de transferencia de hipertexto).

HTTPS *Hyper Text Transfer Protocol Secure* (protocolo seguro de transferencia de hipertexto).

J2EE	Java 2 Enterprise Edition (Java 2da. edición empresarial).
OWASP	*Open Web Application Security Project* (proyecto abierto de seguridad en aplicaciones Web).
Plugin	Complemento adicional que se agrega a una aplicación, para permitir una función adicional.
Proxy	Aplicación o equipo intermedio que intercepta las solicitudes y respuestas entre un cliente y un servidor web.
Query	Consulta SQL.
Sistema operativo	Software diseñado para el manejo de procesos básicos de un ordenador. Permite el uso de otros programas.
SQL	Lenguaje estructurado de consultas.
TCP/IP	Conjunto de protocolos de red en los que se basa en internet, que permiten transmisión de datos entre ordenadores.
URL	*Uniform Resource Locator* (localizador uniforme de recursos).

XML	*Extensible Markup Language* (lenguaje de marcado extensible).
XSS	*Cross Site Scripting* (ejecución de comandos en sitios cruzados).
XXE	*XML External Entity* (entidades externas XML).

RESUMEN

El presente trabajo de graduación se enfoca en el conocimiento básico que debe tener cualquier persona que desarrolle aplicaciones web, para determinar las posibles vulnerabilidades que una aplicación posee y la forma de mitigarlas, de manera que se eviten los riesgos que una entidad corre al enfrentarse a un ataque cibernético de inyección, utilizando los elementos débiles de la misma. El elemento esencial para entender esto es realizar prácticas que ayuden a comprender el funcionamiento del ataque, lo que se realiza mediante el uso de la aplicación WebGoat.

En el capítulo uno se da una introducción a la aplicación WebGoat, herramienta proporcionada por OWASP (proyecto abierto de seguridad en aplicaciones web), con el fin de realizar pruebas de vulnerabilidades que suelen presentarse en las aplicaciones web. Además se mencionan todos los elementos necesarios para el uso, los beneficios que la aplicación ofrece y las maneras en que se puede utilizar.

En el capítulo dos se muestra un resumen de los elementos que componen el protocolo HTTP, el cual es usado actualmente en todas las aplicaciones web, que son accedidas localmente o por internet. Además se abarca el método de comunicación que usa este protocolo, de manera que se entienda la comunicación que existe entre un cliente y un servidor, utilizando mensajes de solicitudes y respuestas para entenderse.

En el capítulo tres se abarcan tres vulnerabilidades que comúnmente son encontradas en las aplicaciones web. Se analiza la forma en que funcionan, que métodos son utilizados para ser explotadas en un ataque, se realizan prácticas de ataques sobre estas vulnerabilidades para una comprensión más profunda del comportamiento esperado y por último se presentan las maneras que estas debilidades se mitigan para que la información que maneje la aplicación se encuentre protegida y no sufra ningún percance.

En el capítulo cuatro se realizan pruebas de ataques de inyección sobre una aplicación ya diseñada, obtenida en internet. Mediante el establecimiento de los resultados esperados y los diferentes objetivos a cumplir, se establecen los distintos ataques de inyección a realizar. Posteriormente se analizan los resultados obtenidos y la validación del cumplimiento con los objetivos establecidos antes de realizar la prueba, de manera que se determine si la aplicación posee vulnerabilidades que pongan en riesgo la información que contiene.

OBJETIVOS

General

Analizar una forma de aprendizaje práctica sobre las vulnerabilidades más explotadas, existentes en aplicaciones web.

Específicos

1. Presentar el uso de una aplicación web, para el aprendizaje de vulnerabilidades web.

2. Analizar las vulnerabilidades más explotadas actualmente y comprender los riesgos que implican.

3. Conocer cuáles son las mejores prácticas para la mitigación de vulnerabilidades.

INTRODUCCIÓN

En la actualidad el uso de las aplicaciones web ha aumentado debido a la facilidad de uso y ventajas que ofrecen. Por lo que el manejo de todo tipo de información se ha vuelto esencial en este tipo de *software*. Esto conlleva que la información sea el activo más importante a proteger que posee una entidad actualmente, ya que ofrece una ventaja competitiva hacía otras entidades similares en el mercado en el que se desenvuelve.

La seguridad de las aplicaciones es un punto esencial para cualquier organización, esto surge por la necesidad constante de proteger los activos importantes de una empresa, por lo que cualquier aplicación debe cumplir con las siguientes cualidades mínimas al tratar con información sensible: seguridad y confiabilidad. El conocer y analizar las amenazas más explotadas en las aplicaciones web, brindará una ventaja a la aplicación debido a que se tendrá un marco de trabajo en donde se entienda el daño que llega a causar una debilidad en una aplicación y como se mitiga para reducir los riesgos.

Se abarcan las vulnerabilidades más explotadas en los sitios web como lo son: inyección de datos, perdida de autenticación, exposición de datos sensibles, entre otros más. Mediante la práctica se busca entender la forma en que las vulnerabilidades crean la posibilidad de poner en riesgo la información sensible de cualquier entidad al ser explotadas en un ataque, los riesgos que se corren al sufrir un ataque que haya culminado de manera exitosa y la forma de mitigar las diferentes contingencias posibles a enfrentar.

Se utiliza la aplicación WebGoat, la cual fue diseñada de manera insegura, para usarse en el aprendizaje de lecciones de seguridad. Aplicación cuyo fin es realizar ejercicios prácticos para entender de una mejor forma como ocurren los ataques que posiblemente afecten un sistema web informático. Además de indicar cuales son las mejores prácticas y recomendaciones a tomar, buscando que las vulnerabilidades que una aplicación posea se reduzcan y por lo tanto el riesgo al sufrir un ataque también.

1. WEBGOAT

El crecimiento que ha tenido el internet y los sistemas informáticos ha sido muy notorio. Debido a actividades que anteriormente se hacían de forma manual, ahora son automatizadas por medio de sistemas, lo que hace la vida más sencilla y cómoda. La era de tecnología que se está desarrollando ha logrado mejorar la calidad de vida de muchas personas, desde evitar largas colas para realizar un pago en los bancos, hasta tener la opción de trabajar desde el hogar con una conexión a internet.

Por todo esto, en la actualidad las actividades realizadas por las empresas, entidades bancarias, personas particulares, personas jurídicas e inclusive por entidades del gobierno requieren contar con sistemas informáticos que cumplan las exigencias del mercado, además de ser seguros para realizar las distintas labores que implican el modelo de negocio que poseen.

La seguridad en las aplicaciones es importante tomando en cuenta que un fallo en una aplicación crea la posibilidad de generar funcionamientos no deseados, lo que provoca daños económicos y judiciales, tanto para los dueños de la aplicación como para los usuarios que se benefician de la misma. En función de la importancia y el carácter sensible de la información que almacene el sistema informático, existe la posibilidad de ser mal usada si es obtenida por personas mal intencionadas.

Es de notar el alza que han tenido las aplicaciones web como sistemas informáticos, en los diferentes modelos de negocio, por tanto, es importante que al momento de ser desarrolladas se consideren el cómo las vulnerabilidades son explotadas en un ataque y la manera de mitigarlas.

Una de las mejores formas de aprender sobre las vulnerabilidades que presenta una aplicación web es realizar ejercicios prácticos, que enseñen como se realizan los ataques y cuáles son las vulnerabilidades más atacadas.

1.1. ¿Qué es WebGoat?

Webgoat es una aplicación creada de manera insegura de forma intencional para aprendizaje, por la comunidad del proyecto abierto de seguridad en aplicaciones web OWASP (por las siglas en inglés *Open Web Application Security Project*). Esta aplicación está diseñada para enseñar lecciones de seguridad en aplicaciones web. En la figura 1 se observa el logo de la aplicación WebGoat.

Figura 1. **Logo de WebGoat**

Fuente: Logo de WebGoat, Github.
https://raw.githubusercontent.com/wiki/WebGoat/WebGoat/images/wg_logo_snag.png.
Consulta: 14 de abril de 2018.

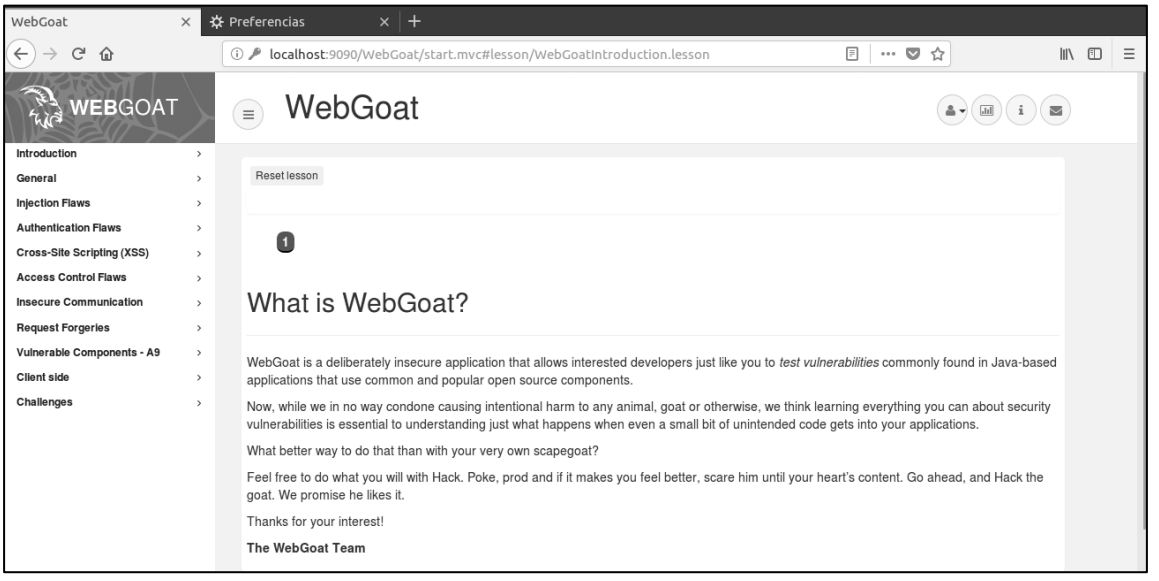
La aplicación WebGoat proporciona una plataforma de prueba para realizar ejercicios prácticos de cada una de las lecciones que contiene. Las cuales se listan a continuación:

- Introducción.
  - WebGoat.
  - WebWolf.
- General.
  - HTTP básico.
  - *Proxies* HTTP.
- Defectos de inyecciones.
  - Inyección SQL.
  - Inyección SQL avanzada.
  - Mitigación de inyección SQL.
  - XXE.
- Defectos de autenticación.
  - Evasión de autenticación.
  - Json web *tokens*.
- Cross site scripting (XSS).
- Defectos de acceso de control.
  - Referencias inseguras de objetos directos.
  - Falla de control a nivel de función.
- Comunicaciones inseguras.
  - *Login* inseguro.
- Falsificación de solicitud entre sitios.
- Componentes vulnerables.
- Lado del cliente.
  - Filtrado del lado del cliente.
  - Evasión de restricciones de *front-end*.

- Manipulación de HTML.

De manera que se evalúa que se haya cumplido el objetivo de la lección en curso, tomando en cuenta que la persona que realiza las pruebas tiene el mismo rol que cualquier usuario común tendría al usar la aplicación. En la figura 2 se muestra la página principal, al momento de montar la aplicación.

Figura 2. **Lecciones Webgoat**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 23 de abril de 2018.

Webgoat ilustra las fallas típicas que se encuentran en las aplicaciones Web. Entre estas fallas se enumeran las siguientes:

- Manejo de *querys* en petición *get*.
- Uso de texto para colocar valores de validación, en vez de parámetros.
- Manejo interno de errores.

El sistema tiene como finalidad enseñar un enfoque estructurado para probar y explotar dichas vulnerabilidades, en el contexto de una evaluación de seguridad en aplicaciones web.

A continuación se enumeran, algunas características que destacan de la aplicación WebGoat:

- Es una aplicación realizada en Java web.
- Las simulaciones de ataques realizadas son remotas.
- Las pruebas de ataque están basadas en el concepto de *black- box*[1].
- Crea la posibilidad de obtener el código para entender cómo fue desarrollada la aplicación.

1.2. Propósito de WebGoat

La seguridad en aplicaciones web resulta un concepto complicado de entender y aplicar en el desarrollo de las mismas. Una de las razones es que no es tan fácil contar con aplicaciones web completas, que proporcionen la posibilidad de realizar búsquedas y pruebas de ataques a vulnerabilidades, debido a que tiene que ocurrir en un ámbito seguro y legal, con el consentimiento del dueño de la aplicación. Es importante recalcar que independientemente de las intenciones (buenas o malas) buscar vulnerabilidades sin permiso en una aplicación crea problemas legales.

[1] *Blac-box:* término usado para indicar que el proceso es transparente para el usuario, de manera que no se conoce el funcionamiento, solo se esperan resultados.

Por esto el propósito del proyecto WebGoat es simple: crear un ambiente web interactivo que permita enseñar seguridad en aplicaciones web mediante lecciones que permiten adquirir las nociones básicas de cómo realizar un ataque e implementar una defensa para protegerse.

Según la página oficial de la aplicación[2], las personas que hacen uso de WebGoat obtendrán las siguientes aptitudes al finalizar las lecciones:

- Comprender los procesos de interacción de alto nivel dentro de una aplicación web.
- Determinar información en datos visibles del cliente que son útiles al realizar un ataque.
- Identificar y comprender los datos y las interacciones del usuario que crean circunstancias que exponen la aplicación a experimentar un ataque.
- Realice pruebas contra esas interacciones para exponer defectos en la operación de la aplicación.
- Ejecutar ataques contra la aplicación para demostrar y explotar vulnerabilidades.

1.3. Aplicaciones complementarias para el uso de WebGoat

Existe una gran cantidad de herramientas y de tipos diferentes que ayudan a una persona enfocada en el estudio de la seguridad web. A continuación se listan los tipos de aplicaciones más utilizadas para el estudio y análisis de la seguridad web:

[2] OWASP Webgoat: https://www.owasp.org/index.php/WebGoat_User_Guide_Objectives. Consulta: 25 de abril de 2018.

- Programas para realizar escaneo de vulnerabilidades. Por ejemplo: Vega[3], Skipfish[4].
- Programas para analizar los distintos protocolos (*sniffer*). Por ejemplo: WireShark[5], Microsoft Network Monitor[6].
- Herramientas de descifrado de contraseñas. Por ejemplo: RainbowCrack[7], *Cain and Abel*[8].

En este caso para realizar los diferentes ejercicios que presenta WebGoat, se necesita una herramienta que permita capturar los paquetes que envía un navegador a una página en específica, utilizando el protocolo HTTP.

A continuación se listan los programas recomendados, para los ejercicios prácticos solamente es necesario instalar uno:

- Owasp Zap[9].
- Tamper Data[10].
- WebScarab[11].

[3] Subgraph: https://subgraph.com/vega/. Consulta: 25 de abril de 2018.
[4] Google code: https://code.google.com/p/skipfish/. Consulta: 25 de abril de 2018.
[5] Wireshark: https://www.wireshark.org/. Consulta: 25 de abril de 2018.
[6] Microsoft: https://www.microsoft.com/en-us/download/details.aspx?id=4865. Consulta: 25 de abril de 2018.
[7] RainbowCrack: http://project-rainbowcrack.com/. Consulta: 25 de abril de 2018.
[8] OXID.IT: http://www.oxid.it/ca_um/. Consulta: 25 de abril de 2018.
[9] OWASP: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project. Consulta: 25 de abril de 2018.
[10] Mozilla Dev: http://tamperdata.mozdev.org/. Consulta: 25 de abril de 2018.
[11] SourceForge: https://sourceforge.net/projects/owasp/files/WebScarab/. Consulta: 25 de abril de 2018.

1.3.1. Owasp Zap

Es una de las herramientas más potentes que posee el proyecto OSWAP. Creada con el fin de monitorizar la seguridad en aplicaciones web, lo que ayuda en las auditorias de seguridad. Las características esenciales que hacen atractivo el uso de Zap son las siguientes:

- Gratuito y *open source*.
- Multiplataforma.
- Facilidad de uso.
- Permite asignar prioridades al tráfico de red.
- Soporte en la comunidad de internet.
- Multilenguaje

Las funciones que existe la posibilidad de realizar utilizando esta aplicación son las siguientes:

- Comprobar peticiones entre cliente y servidor.
- Localizar recursos en un servidor.
- Escaneo automático.
- Escaneo pasivo.
- Lanzamiento de ataques en simultáneo.
- Uso de certificados SSL dinámicos.
- Análisis en sistemas de autenticación.
- Tienda de *plugins*, para agregar más funciones a la herramienta.

1.3.2. Temper Data

Es un complemento para Firefox que permite ver y modificar las solicitudes HTTP, antes de que sean enviadas al servidor. Permite visualizar la información que envía el navegador web, como por ejemplo: *cookies* y campos de formulario ocultos. Este complemento permite utilizarse en los siguientes escenarios:

- Test de seguridad sobre aplicaciones web.
- Rastreo de solicitudes y respuestas.

Para utilizarlo se instala el complemento en el navegador web Firefox.

1.3.3. WebScarab

Proyecto desarrollado por OWASP como marco de trabajo para el análisis de aplicaciones web que se comunican usando los protocolos HTTP y HTTPS. Dentro de las características que posee se mencionan las siguientes:

- Desarrollado en Java.
- Multiplataforma.
- Distintos modos de operación, dependiendo de los *plugins* con los que se cuente.
- Permite capturar paquetes tanto de envío como de recepción generadas por un navegador web y modificarlas antes que lleguen al destino.
- Permite revisar las conversaciones (peticiones y respuestas) capturadas por el *software*.

Las funciones que permite realizar este *software* son las siguientes, las cuales dependen de los *plugin* que se posean:

- Extraer *scripts* y comentarios de páginas HTML, en el momento que son percibidas por el *proxy* u otros *plugins*.
- Función de *proxy* para observar el tráfico entre el navegador y los servidores web que consulta. Es posible observar tráfico HTTP y HTTPS.
- Intercepción manual, permite modificar peticiones y respuestas HTTP y HTTPS antes de que alcance el servidor o el navegador.
- Modificación de campos ocultos mediante hacerlos visibles y editables.
- Simulador de ancho de banda.
- Arañas o *spiders*, para obtener nuevas URLs en el sitio que se consulta y la información cuando se le indica.
- Análisis de identificadores de sesión.

1.4. Requerimientos de instalación de WebGoat

La aplicación WebGoat, es una aplicación web J2EE, por lo tanto es compatible con cualquier plataforma que posea una máquina virtual de Java instalada. Adicional cuenta con programas que permiten la instalación en los siguientes sistemas operativos: Linux, Os X Tiger y Windows. Los programas que se permite utilizar para instalar WebGoat se listan a continuación:

- Docker
- Git
- Vagrant

2. COMPONENTES DE UN ATAQUE

El aprender como mitigar las vulnerabilidades que presenta una aplicación web, implica conocer y estudiar los elementos esenciales que influyen en el uso de las aplicaciones, además de los componentes que participan entre la comunicación de un cliente hacia un servidor. De manera que es importante saber la forma en que se realiza la comunicación dentro de una red informática, para asegurar que los datos sean enviados y recibidos correctamente.

El modelo OSI, llamado así debido a las siglas en inglés *Open System Interconnection* (Interconexión de Sistemas Abiertos). Es una herramienta utilizada para enseñar cómo se realiza el envío de datos de un dispositivo a otro usando una red para la comunicación. Este modelo ayuda a entender cómo se realizan las diferentes tareas dentro de una red, de manera que la transmisión de datos sea fiable y se dé sin errores. En la figura 3 se muestra las capas que componen al modelo y una breve explicación de que labor realizan.

Figura 3. **Modelo OSI**

Fuente: INGESOR. *Introducción al modelo OSI*. https://i1.wp.com/www.elingesor.com/wp-content/uploads/2016/05/modelo_osi.jpg. Consulta: 21 de mayo de 2018.

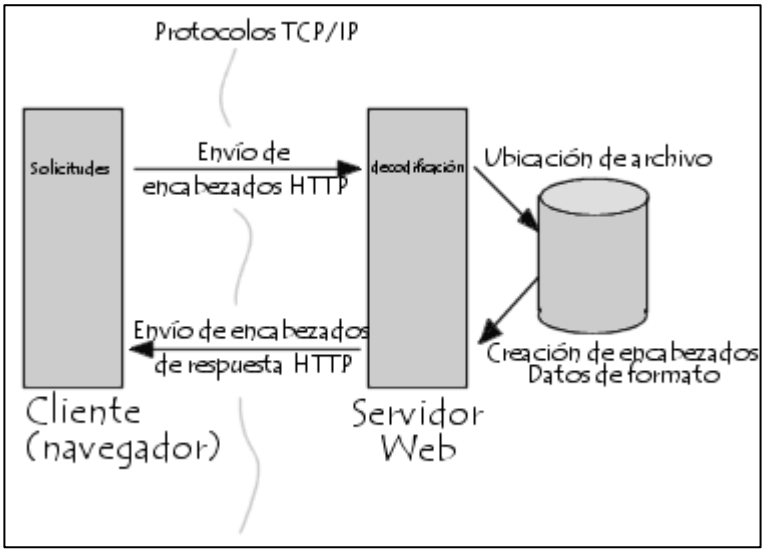
A continuación se da una breve definición de los conceptos que son esenciales conocer para llevar a cabo los ejercicios prácticos de la herramienta WebGoat.

2.1. Protocolo HTTP

El protocolo de transferencia de hipertexto o HTTP (por las siglas en inglés de *HyperText Transfer Protocol*) es el encargado de los intercambios de información entre un cliente web y el servidor web al que se conecta. El protocolo HTTP fue diseñado en la década de 1990, caracterizado por la evolución que tuvo con el paso del tiempo. Pertenece a la capa de aplicación del modelo TCP/IP.

Está basado en comunicación simple conformada por solicitudes y respuestas. El cliente web se conecta al servidor, posteriormente envía solicitudes con los datos necesarios para recibir una respuesta similar del servidor, conformada por el estatus de la misma y el resultado de la petición. Dado a la versatilidad del protocolo HTTP, permite transmitir imágenes, videos, datos o contenidos a los servidores e inclusive para la transmisión de documentos, no solo documentos de hipertexto (HTML). En la figura 4 se muestran los elementos que influyen en la comunicación del protocolo HTTP.

Figura 4. **Comunicación protocolo HTTP**



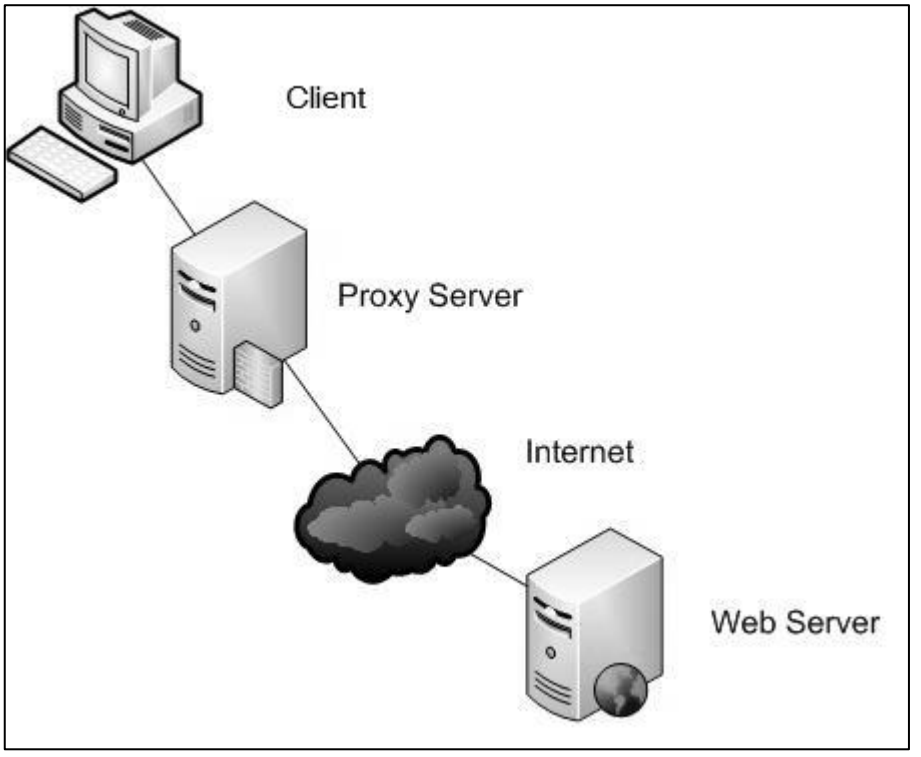
Fuente: El protocolo HTTP. https://img-17.ccm2.net/-P-kXsfhNd-6KT1HjIb3_8YKn3w=/377x/61c0586238c04582855627961b329882/ccm-encyclopedia/internet-images-comm.gif. Consulta: 06 de mayo de 2018.

2.1.1. Actores en la comunicación HTTP

A continuación se listan los elementos que comúnmente participan en la comunicación del protocolo HTTP, como se observa en la figura 5:

- Cliente.
- Servidor
- *Proxy*

Figura 5. **Actores comunicación HTTP**



Fuente: ¿Qué es un servidor *proxy* y porque debería implementarlo en mi empresa? https://www.muycomputerpro.com/wp-content/uploads/2014/01/Proxy_5515FA94.jpg. Consulta: 06 de mayo de 2018.

2.1.1.1. Cliente

El cliente será cualquier herramienta que represente a un usuario realizando solicitudes. Por lo general esta labor es realizada por un navegador web, debido a que es la aplicación más usada por usuarios finales. El cliente es por lo regular quien inicia la comunicación mediante una petición al servidor, en excepciones muy escasas sucede al revés.

En el proceso de comunicación el cliente realiza una petición con un objetivo en específico, utilizando la semántica que maneja el servidor al que consulta. Luego el cliente tiene que tener la capacidad de procesar las respuestas, de manera que confirme si se llevó a cabo de manera exitosa la solicitud enviada. Además tiene que tener la facultad de interpretar la manera de responder del servidor para mostrar la información solicitada al usuario.

2.1.1.2. Servidor web

El servidor será el encargado de obtener las peticiones de los clientes, procesarlas y devolver el resultado. Siempre se mantiene a la escucha de una conexión mediante un puerto destinado con este objeto. Debe encargarse de interpretar los elementos del mensaje con el objetivo de la solicitud recibida, luego responde con uno o varios mensajes al cliente. Conceptualmente se habla de un servidor como entidad, debido a que existe la posibilidad de que lo formen varios elementos para el manejo de peticiones.

2.1.1.3. Proxies

Entre el cliente y el servidor en ocasiones existen dispositivos que manejan y gestionan las solicitudes HTTP, entre estos se encuentran los *proxies* que manejan los paquetes en la capa de aplicación. Los *proxies* tienen distintas funciones, a continuación se enumeran las más básicas:

- Cache.
- Filtrado.
- Balanceo de carga.
- Autenticación.
- Log de eventos.

Los *proxies* se explicarán más a detalle en el apartado 2.2 de este capítulo.

2.1.2. Métodos de petición

El protocolo HTTP cuenta con una serie predeterminada de métodos de petición, en ocasiones nombrados como verbos, para ser usados al momento de realizar una solicitud. El protocolo es flexible, debido a que permite añadir nuevos métodos con el fin de conseguir nuevas funcionalidades. Cada método se caracteriza porque el nombre fue colocado con relación a la acción a realizar sobre el recurso del servidor solicitado. Es importante recalcar que este recurso dependerá de los servicios web que se consuman.

La definición formal de los métodos se encuentra en el documento RFC 2616[12], que presenta los estándares de la definición de la versión HTTP/1.1, la más usada actualmente. En los siguientes apartados se explican y detallan los métodos que se utilizan para la comunicación mediante el protocolo HTTP de los cuales dependen los demás elementos de la comunicación.

2.1.2.1. Get

Método encargado de solicitar la representación de un recurso en específico. En este método los parámetros se envían en la URL, al momento de hacer la solicitud. Devuelve la cabecera de los metadatos y el recurso en sí. En la figura 6 se muestra un ejemplo de la petición y posteriormente en la figura 7 un ejemplo de la respuesta obtenida por el servidor, usando el método.

[12] Documento de definición HTTP/1.1: https://tools.ietf.org/html/rfc2616. Consulta: 05 de mayo de 2018.

Figura 6. **Ejemplo de petición *get***

```
GET /index.html HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.michelletorres.mx
Accept-Language: es-mx
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
```

Fuente: Michelletorre.mx. http://michelletorres.mx/peticiones-http-get-post-put-delete-etc/. Consulta: 06 de mayo de 2018.

Figura 7. **Ejemplo de respuesta *get***

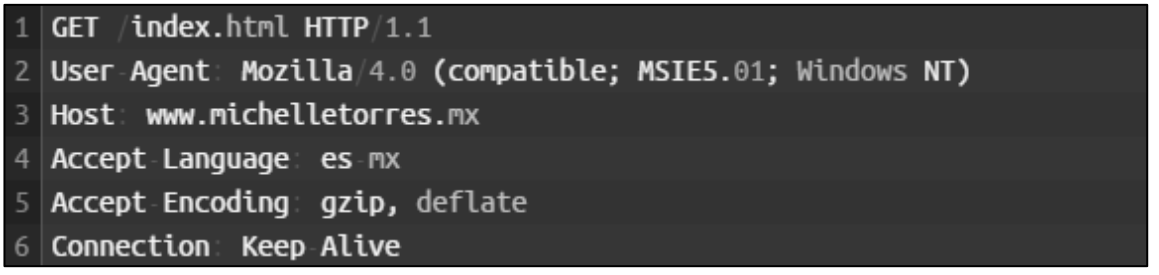
```
Ejemplo respuesta del servidor:
HTTP/1.1 200 OK
Date: Wed, 08 Nov 2017 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Mon, 22 Jul 2014 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Vary: Authorization,Accept
Accept-Ranges: bytes
Content-Length: 88
Content-Type: text/html
Connection: Closed

<html>
<body>
<h1>Howdy, Michelle!</h1>
</body>
</html>
```

Fuente: Michelletorre.mx. http://michelletorres.mx/peticiones-http-get-post-put-delete-etc/. Consulta: 06 de mayo de 2018.

2.1.2.2. Head

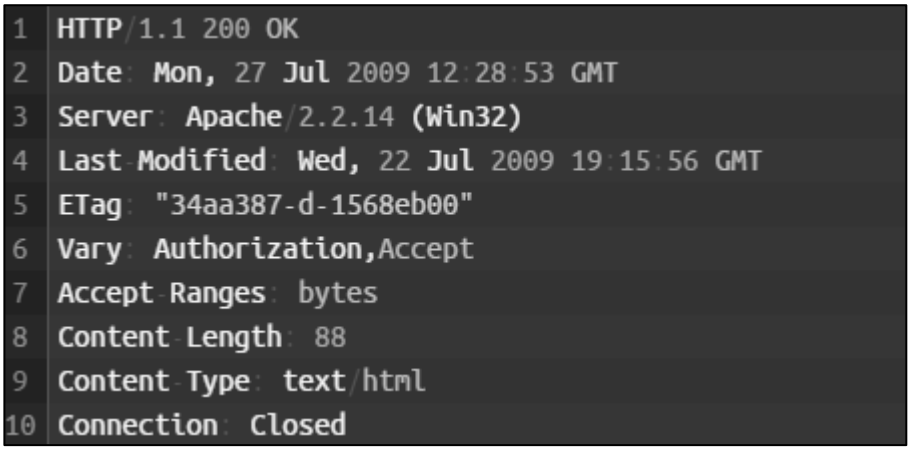
Método que genera una solicitud igual que una petición *get*, con la diferencia que no espera el cuerpo de la respuesta, solo la línea de estado y la sección del encabezado con los metadatos. En las figura 8 se muestra un ejemplo de la petición y posteriormente en la figura 9 un ejemplo de la respuesta obtenida por el servidor, usando el método.

Figura 8. **Ejemplo de petición *head***



Fuente: Michelletorre.mx. http://michelletorres.mx/peticiones-http-get-post-put-delete-etc/. Consulta: 06 de mayo de 2018.

Figura 9. **Ejemplo de respuesta *head***

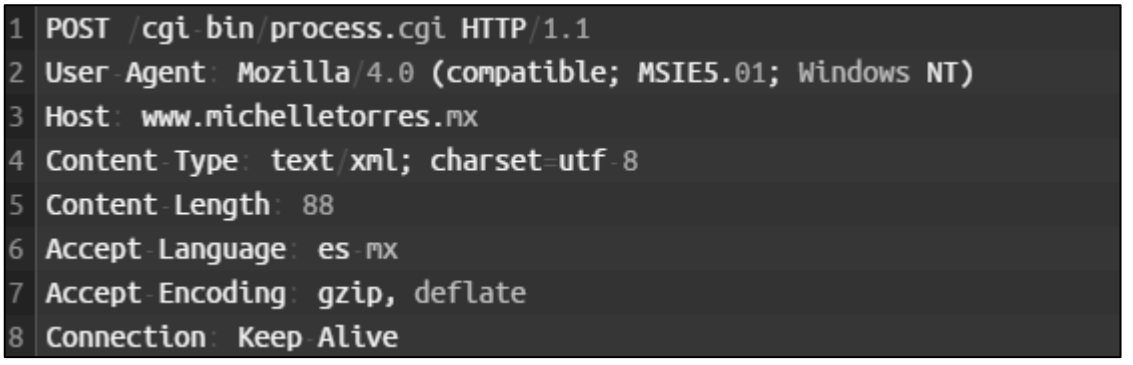


Fuente: Michelletorre.mx. http://michelletorres.mx/peticiones-http-get-post-put-delete-etc/. Consulta: 06 de mayo de 2018.

2.1.2.3. Post

Método que envía una entidad en la solicitud, la cual contiene valores, que se encuentran en el cuerpo de la petición y está dirigida a un recurso específico. Esto provoca que el servidor realice procesos utilizando los valores enviados. Utilizado comúnmente en el envió de formularios. Una de las ventajas de usar el método *post*, es que los datos que envía la petición no es posible visualizarlos a simple vista para el usuario. En la figura 10 se muestra un ejemplo de la petición usando el método.

Figura 10. **Ejemplo de petición *post***



Fuente: Michelletorre.mx. http://michelletorres.mx/peticiones-http-get-post-put-delete-etc/. Consulta: 06 de mayo de 2018.

2.1.2.4. Put

Método que se encarga de crear o reemplazar como se representa el recurso de destino existente, con la carga útil que posee la petición. El cambio se realiza mediante una conexión *socket* que se establece con el servidor. En la figura 11 se muestra un ejemplo de la petición realizada y posteriormente en la figura 12 un ejemplo de la respuesta obtenida por el servidor, usando el método.

Figura 11. **Ejemplo de petición *put***

```
PUT /index.htm HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
Host: www.michelletorres.mx
Accept-Language: es-mx
Connection: Keep-Alive
Content-type: text/html
Content-Length: 182

<html>
<body>
<h1>Howdy, Michelle!</h1>
</body>
</html>
```

Fuente: Michelletorre.mx.http://michelletorres.mx/peticiones-http-get-post-put-delete-etc/.
Consulta: 06 de mayo de 2018.

Figura 12. **Ejemplo de respuesta *put***

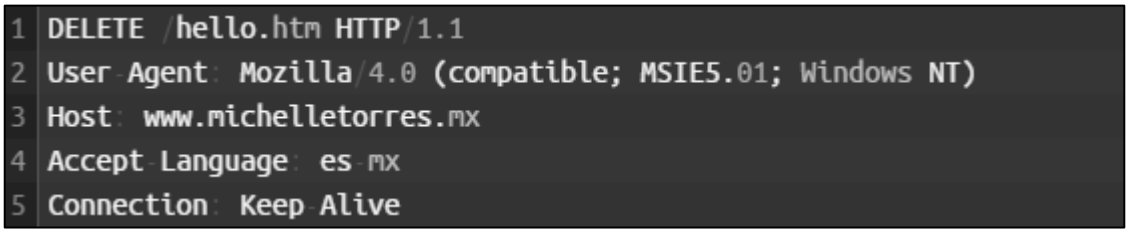
```
HTTP/1.1 201 Created
Date: Mon, 27 Nov 2017 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Content-type: text/html
Content-length: 30
Connection: Closed
```

Fuente: Michelletorre.mx. http://michelletorres.mx/peticiones-http-get-post-put-delete-etc/.
Consulta: 06 de mayo de 2018.

2.1.2.5. Delete

Método que borra las representaciones de un recurso en específico. En la figura 13 se muestra un ejemplo de la petición y posteriormente en la figura 14 se muestra un ejemplo de la respuesta obtenida por el servidor, usando el método.

Figura 13. **Ejemplo de petición *delete***



Fuente: Michelletorre.mx. http://michelletorres.mx/peticiones-http-get-post-put-delete-etc/. Consulta: 06 de mayo de 2018.

Figura 14. **Ejemplo de respuesta *delete***

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Content-type: text/html
Content-length: 30
Connection: Closed

<html>
<body>
<h1>URL deleted.</h1>
</body>
```

Fuente: Michelletorre.mx. http://michelletorres.mx/peticiones-http-get-post-put-delete-etc/. Consulta: 06 de mayo de 2018.

2.1.2.6. Connect

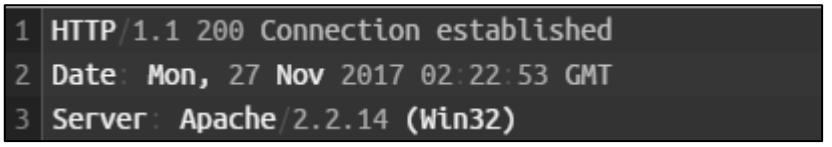
Método que se utiliza para establecer un túnel de tipo TCP/IP hacia un servidor que se identifica con el recurso. Regularmente es utilizado para comprobar si un *proxy* brinda acceso a un host bajo ciertas condiciones. En la figura 15 se muestra un ejemplo de la petición y posteriormente en la figura 16 un ejemplo de la respuesta obtenida por el servidor, usando el método.

Figura 15. **Ejemplo de petición *connect***

```
CONNECT www.michelletorres.mx HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

Fuente: Michelletorre.mx. http://michelletorres.mx/peticiones-http-get-post-put-delete-etc/. Consulta: 06 de mayo de 2018.

Figura 16. **Ejemplo de respuesta *connect***



Fuente: Michelletorre.mx. http://michelletorres.mx/peticiones-http-get-post-put-delete-etc/. Consulta: 06 de mayo de 2018.

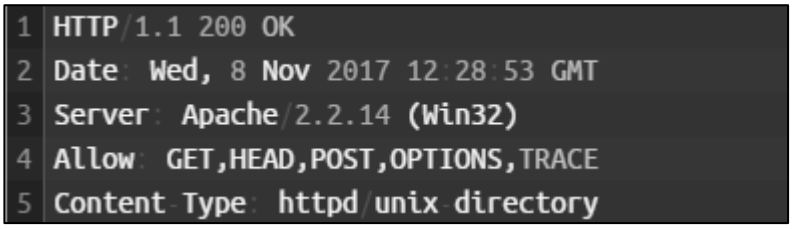
2.1.2.7. Options

Método utilizado para detallar las opciones en la comunicación con el recurso de destino. Se utiliza con el fin de comprobar un servicio web y los métodos HTTP que soporta. En la figura 17 se muestra un ejemplo de la petición y posteriormente en la figura 18 un ejemplo de la respuesta obtenida por el servidor, usando el método.

Figura 17. **Ejemplo de petición *options***

Fuente: Michelletorre.mx. http://michelletorres.mx/peticiones-http-get-post-put-delete-etc/. Consulta: 06 de mayo de 2018.

Figura 18. **Ejemplo de respuesta *options***

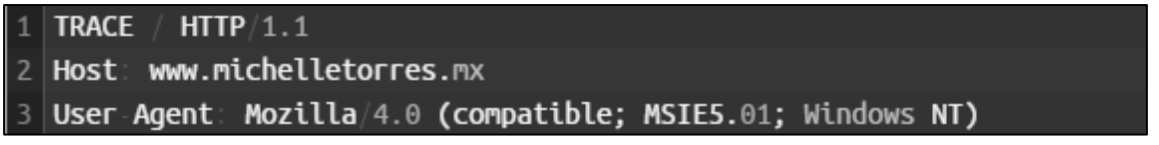


Fuente: Michelletorre.mx. http://michelletorres.mx/peticiones-http-get-post-put-delete-etc/. Consulta: 06 de mayo de 2018.

2.1.2.8. Trace

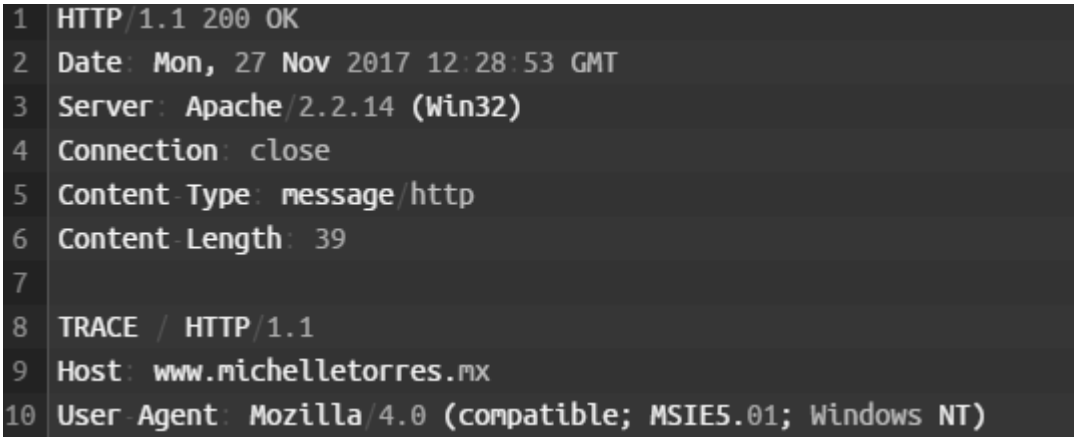
Método encargado de realizar una prueba repetitiva a lo largo de la ruta hacia el recurso de destino, para conocer los elementos intermedios que influyen en la petición. En la figura 19 se muestra un ejemplo de la petición y posteriormente en la figura 20 un ejemplo de la respuesta obtenida por el servidor, usando el método.

Figura 19. **Ejemplo de petición *trace***



Fuente: Michelletorre.mx. http://michelletorres.mx/peticiones-http-get-post-put-delete-etc/. Consulta: 06 de mayo de 2018.

Figura 20. **Ejemplo de respuesta *trace***



```
HTTP/1.1 200 OK
Date: Mon, 27 Nov 2017 12:28:53 GMT
Server: Apache/2.2.14 (Win32)
Connection: close
Content-Type: message/http
Content-Length: 39

TRACE / HTTP/1.1
Host: www.michelletorres.mx
User-Agent: Mozilla/4.0 (compatible; MSIE5.01; Windows NT)
```

Fuente: Michelletorre.mx. http://michelletorres.mx/peticiones-http-get-post-put-delete-etc/. Consulta: 06 de mayo de 2018.

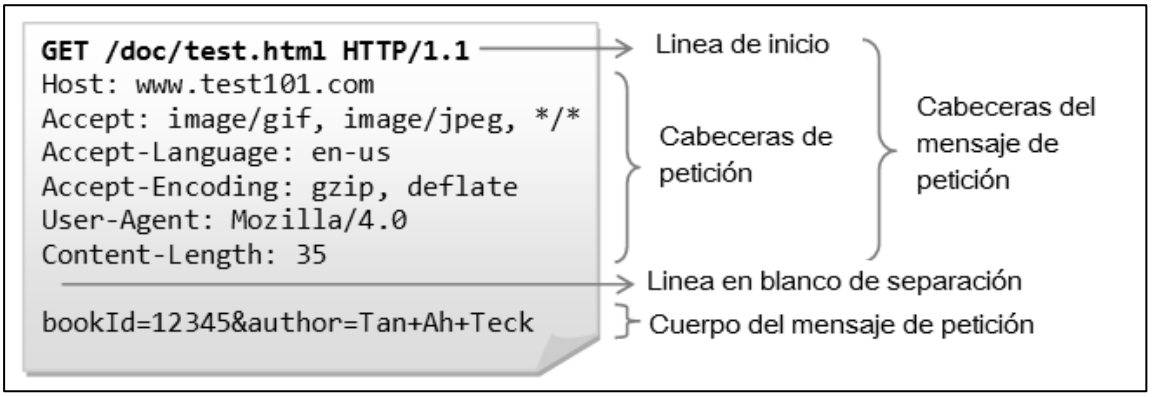
2.1.3. Mensajes de solicitud HTTP

Cuando un cliente necesita obtener información de un servidor web, lo solicita mediante el uso de un mensaje de petición HTTP. Este mensaje está compuesto por texto, codificado en ASCII y existe la posibilidad de que este constituido por múltiples líneas. Uno de los elementos importantes para realizar una solicitud es el método a utilizar, mayormente se utilizan los métodos predefinidos del protocolo HTTP/ 1.1, analizados en el apartado anterior.

Sin embargo, no es el único elemento de la solicitud. Un mensaje HTTP completo se compone de los siguientes elementos, tal y como se observa en la figura 21:

- Línea de inicio
- Encabezados.
- Cuerpo.

Figura 21. **Componentes de solicitud HTTP**



Fuente: Github. https://github.com/TonyChagolla/ejemplo. Consulta: 06 de mayo de 2018.

2.1.3.1. Línea de inicio

Parte inicial del mensaje de solicitud HTTP que describe la información básica de la petición a ser implementada. Tiene la siguiente estructura:

[Método] [URL] [Versión]

A continuación se detallan los valores mencionados anteriormente:

- Método HTTP**:** en este apartado se coloca la acción que se necesita ejecutar con la petición. Por ejemplo: enviar un archivo al servidor para realizar cambios en el servidor.
- URL: el objetivo de una petición se especifica mediante la dirección completa incluyendo el protocolo y puerto, dependiendo del contexto de la petición.
- Versión HTTP: indica la estructura que los mensajes deben tener y cuál es el formato que espera de respuesta, en base a la versión HTTP a usar.

2.1.3.2. Encabezados HTTP

En esta parte de la petición se envía información útil para el servidor si es un mensaje de solicitud o al cliente si es un mensaje de repuesta. Mensajes que son generados al momento realizar una comunicación entre un cliente y un servidor web. Existe una gran cantidad de encabezados que son permitidos en un mensaje HTTP, la mayoría de estos no son obligatorios a excepción del *host* (elemento que envía la petición). Es importante recalcar que al utilizar un encabezado se debe cumplir con los estándares, para que sea interpretado por el receptor (cliente o servidor).

A continuación se describen los encabezados más usados en la tabla I:

Tabla I. **Encabezados más usados**

Cabecera	Descripción
Referer	Cuando el usuario hace *click* en un enlace, el cliente envía la URL de la página de referencia en esa cabecera.
User-Agent	Información sobre el agente de usuario (*software*) que hace la solicitud. Muchas aplicaciones utilizan la información en esta cabecera, cuando está presente, para descubrir que explorador está haciendo la solicitud (Internet Explorer 6 o Internet Explorer 9 o Chrome, entre otros).
Accept	Describe los tipos de medio (*media-types*) que el agente de usuario está dispuesto a aceptar. Esta cabecera es utilizada para la negociación de contenido.
Accept-Language	Describe el idioma que el agente de usuario prefiere.
Cookie	La información de *cookies* generalmente ayuda a un servidor a dar seguimiento o identificar a un usuario.
If-Modified-Since	Esta contiene una fecha de cuando el agente de usuario obtuvo (y almacenó en caché) el recurso. El servidor solo envía de vuelta el recurso completo si ha sido modificado desde esa fecha.

Fuente: Http Conciso: Mensajes HTTP. Scott Allen. https://code.tutsplus.com/es/tutorials/http-succinctly-http-messages--net-33699. Consulta: 22 de abril de 2018.

2.1.3.3. Cuerpo del mensaje

Esta parte de un mensaje HTTP contiene la información que se utiliza para dar respuesta de parte del emisor al receptor (cliente-servidor o servidor-cliente). Algunos de los elementos que lleva el cuerpo del mensaje se listan a continuación:

- Contenido de formulario HTML.

- Archivos.
- Documentos.

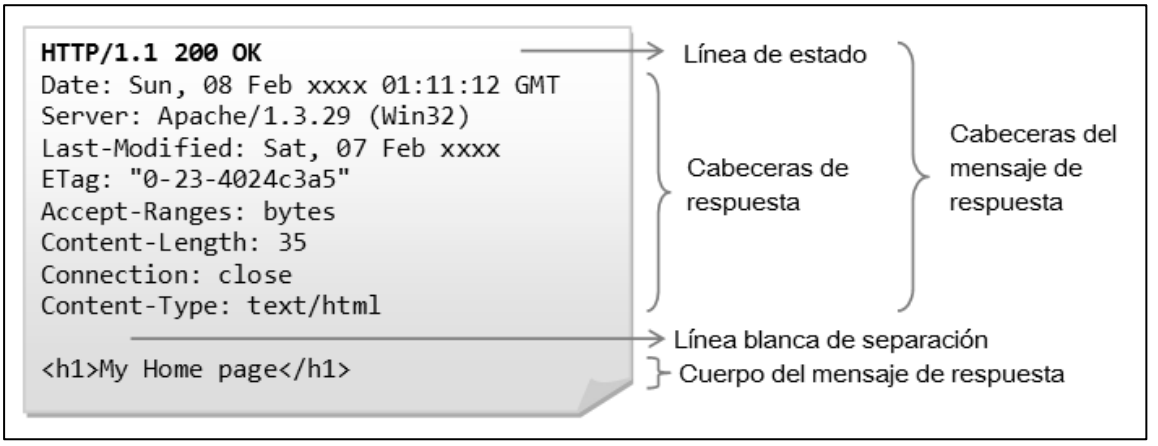
El cuerpo es opcional dentro del mensaje HTTP por lo tanto el encabezado indica si este existe y el tamaño que posee. Normalmente no es usado en peticiones que solicitan información del servidor como por ejemplo la petición *get*. En cambio las solicitudes de modificación de información si lo llevan como es el caso de la petición *post*.

2.1.4. Mensaje de respuesta HTTP

Un mensaje de este tipo tiene una estructura similar a la que posee el mensaje de solicitud HTTP. Las secciones que posee son las siguientes, tal y como se observa en la figura 22:

- Línea de inicio o de estado.
- Encabezados.
- Cuerpo.

Figura 22. **Componentes de respuesta HTTP**



Fuente: Github. https://github.com/TonyChagolla/ejemplo. Consulta: 06 de mayo de 2018.

Como se observa en la figura 22 el único que elemento que cambia con respecto a la solicitud HTTP es la línea de inicio. El encabezado y el cuerpo tienen la misma definición expuesta anteriormente.

2.1.4.1. Línea de inicio o de estado

Parte inicial del mensaje de respuesta HTTP que describe la información básica de la respuesta obtenida por parte del servidor. Tiene la siguiente estructura:

[Versión] [Estado] [Razón]

A continuación se detallan los valores mencionados anteriormente:

- Versión: indica la estructura y formato de la respuesta, que se está recibiendo.
- Estado: indica el estado del resultado generado por la solicitud del cliente.

- Razón: indica el detalle del estado de la respuesta.
- Códigos de estado: el código de estado es un número predefinido en el protocolo HTTP para identificar que sucedió con la petición hecha. Están subdivididos en rangos de los cuales, cada rango pertenece a una categoría que cataloga el tipo de respuesta que se obtiene. En la tabla II se observan los rangos establecidos para los códigos de estado y categorías a las que pertenece el mensaje de respuesta que el servidor regresa.

Tabla II. **Rangos de códigos de estado**

Rango	Categoría
100-199	Mensajes del tipo informativo.
200-299	Mensajes de solicitud exitosa.
300-399	Mensajes de redirección.
400-499	Mensajes de error del cliente
500-599	Mensajes de error del servidor

Fuente: elaboración propia.

Existe una gran cantidad de códigos de estado posibles en el protocolo HTTP, de los cuales se mencionan los más usados y conocidos en la tabla III.

Tabla III. **Códigos de estado más usados**

CÓDIGO	ESTADO	RAZÓN
100	Continuar	Repuesta que indica que al momento todo está bien y el cliente debe continuar con la solicitud.
101	Cambio de protocolo	Respuesta que se obtiene en base al encabezado *upgrade*, en una solicitud. Indica si el servidor soporta el cambio de protocolo solicitado por el cliente.
200	*OK*	Indica que la comunicación fue realizada de manera exitosa
201	Creado	Indica que la solicitud tuvo éxito y se ha creado un nuevo recurso como resultado de ello.
202	Aceptado	Indica que la solicitud fue recibida de manera exitosa pero aún no se ha procesado. Esto indica que la solicitud está siendo procesada por el servidor pero el protocolo no será capaz de devolver el resultado de la misma.
204	No contenido	Indica que la petición fue procesada exitosamente, pero la respuesta no tiene contenido, aunque se utilizan los encabezados en caché.
301	Movido permanentemente	Este mensaje indica que existe comunicación con el servidor, pero el recurso al que se busca acceder fue movido permanentemente a otra dirección, por lo que es necesario que el cliente re direccione a esta definitivamente.
302	Movido temporalmente	Este mensaje indica que existe comunicación con el servidor pero el recurso al que se desea acceder ha sido movido temporalmente a otra dirección, por lo que es necesario que el cliente re direccione por un tiempo.
303	Ver otros	Este mensaje indica al cliente, que para obtener el recurso solicitado se direccionara una petición *get* a otra URL.
304	No modificado	Este mensaje se usa con fines de almacenamiento en caché. Le indica al cliente que la respuesta no ha sido modificada. Entonces, el cliente continua usando la misma versión de respuesta en caché.

Continuación de la tabla III.

400	Solicitud incorrecta	Este mensaje significa que el servidor no fue capaz de interpretar la solicitud dada una sintaxis inválida.
401	No autorizado	Este mensaje indica que es necesario realizar una autenticación para obtener la respuesta de la petición.
403	Prohibido	El cliente no posee los permisos necesarios para el contenido solicitado, por lo tanto el servidor rechaza la solicitud.
404	No encontrado	Este mensaje indica que no se encontró el recurso solicitado.
405	Método no permitido	Este mensaje indica que el método utilizado en la solicitud es identificado por el servidor pero ha sido deshabilitado.
407	Autenticación *proxy* requerida	Este mensaje es similar al 401, con la diferencia que indica que la autenticación se realiza en el *proxy* que usa el cliente.
500	Error interno de servidor	Este mensaje indica que el servidor ha encontrado una situación que no sabe manejar, por lo tanto, no cumple con la solicitud.
501	No implementado	Indica que el método solicitado no es compatible con el servidor, por lo tanto no lo maneja.
503	Servicio no disponible	Indica que el servidor no es capaz de manejar la solicitud. Este error suele presentar cuando un servidor está en mantenimiento o sobrecargado.
505	Versión HTTP no compatible	Indica que la versión http utilizada no es compatible con el servidor.

Fuente: elaboración propia.

Los códigos de estado son una parte esencial en la comunicación del protocolo HTTP, ya que indican al cliente que ocurrió con la solicitud hecha al servidor.

2.2. Servidor *proxy*

Un servidor *proxy* es un equipo o *software* encargado de ser intermediario entre un dispositivo inicial y un destino final, por lo regular el destino inicial es una computadora y el destino final un servidor. Impide la comunicación directa entre el equipo de origen y el equipo de destino.

2.2.1. Características principales de un *proxy*

Un *proxy* es usado para múltiples funciones dependiendo de las necesidades que se tengan. A continuación se detallan las funciones más comunes de un *proxy*:

- Filtrar: una de las características más esenciales y usadas de un *proxy*, se refiere al bloqueo de sitios, contenido y recursos que existe la posibilidad de que sean maliciosos. Regularmente usado en empresas para mantener la seguridad en la red.
- Autenticar: otra característica esencial de un *proxy* se refiere al proceso de conexión por medio de los usuarios, de manera que se manejen diferentes roles de acceso. Por lo tanto, evitará que un rol acceda a un recurso al cual no tiene permiso.
- *Logs*: un proxy permite almacenar los accesos a internet de los usuarios, durante un tiempo determinado, sin que ellos se den cuenta y de esta forma determinar que uso se le está dando a la red.
- Almacenamiento de cache: un *proxy* tiene la capacidad de guardar el contenido total de las solicitudes realizadas, de manera que cuando otro usuario realice una solicitud similar no tenga que realizar el llamado externo. De esta forma se carga la información que posee el *proxy* del sitio web, lo que ayuda a agilizar la navegación.

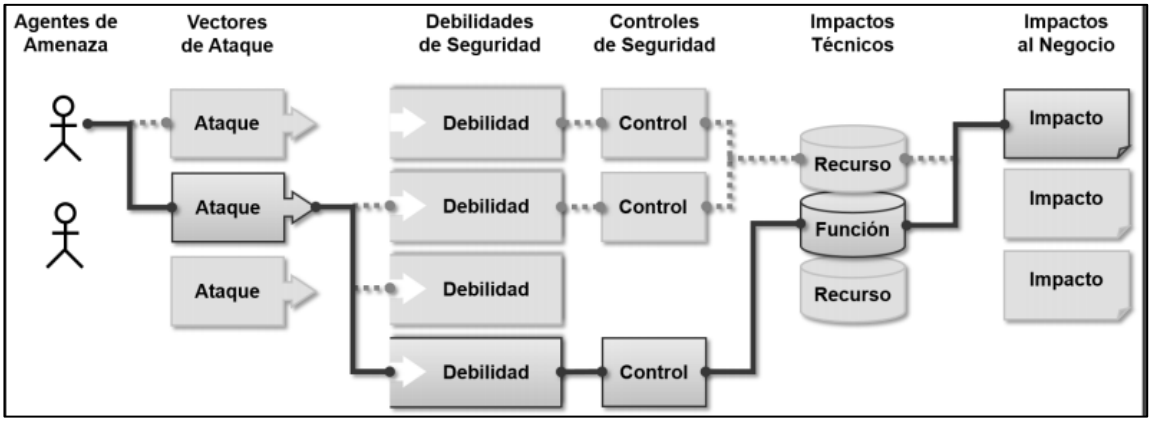
- Conexión compartida: con una adecuada configuración, el *proxy* se encargará de distribuir equitativamente el tráfico hacia internet. De manera que existan varios usuarios navegando sin que ninguno tenga pérdida de velocidad en la navegación.
- Listas negras: son listas donde se encuentran todos los sitios a los que no se tiene permiso de acceso debido a varias razones. Esta lista se configura y administra para agregar o quitar elementos.
- Bloqueo de IP: esta función restringe el acceso a una entidad en particular mediante la IP. Esto es de utilidad cuando existe alguien mal intencionado que desea acceder a la red o algún elemento interno que esté haciendo mal uso de la misma.
- Archivos no permitidos: esta función elige si los usuarios tienen permitido realizar descargas de archivos. Esto sirve para proteger la red de virus maliciosos que se obtienen al obtener archivos de sitios poco confiables.
- Control de usuarios: permite restringir quien tiene los permisos para conectarse a la red o no. De manera que aunque una entidad no deseada se logre infiltrar a la red, no accederán a internet porque el *proxy* denegará el acceso.

Estas son algunas funcionalidades que posee un *proxy*, las cuales es posible resumir en el manejo de permisos dentro de una red, de manera que se evite accesos no permitidos y el uso indebido.

3. ATAQUES A VULNERABILIDADES

El desarrollo de aplicaciones web, conlleva el considerar los distintos ataques de los que existe la posibilidad que la aplicación afronte. Además de reconocer que estos ataques surgen debido a las debilidades que existen al momento de la elaboración de los distintos módulos que comprenden una aplicación. De manera que en una aplicación existen varias rutas a través de las cuales un atacante utiliza para perjudicar el negocio o la organización. En la figura 23 se muestra un esquema del funcionamiento de un ataque.

Figura 23. **Esquema de funcionamiento de un ataque**



Fuente: OWASP Top 10-2017. https://www.owasp.org/images/5/5e/OWASP-Top-10-2017-es.pdf. Consulta: 21 de mayo de 2018.

De manera que para mitigar las vulnerabilidades que una aplicación posee, se debe considerar cada uno de los caminos posibles a explotar por un atacante. Hay que reconocer que unos caminos son más difíciles que otros e influye en el riesgo que representan a una entidad o empresa. Esto también se determina mediante la estimación de impacto que un ataque produce.

3.1. Inyección SQL

Este ataque consiste en la inserción de lenguaje de consulta estructurado, conocido comúnmente como SQL (por las siglas en inglés *Structured Query Language*) usando los datos de entrada que el cliente envía al servidor, desde una aplicación. El objetivo de este ataque es obtener datos sensibles, utilizando la ejecución de operaciones de manipulación e inclusive de definición de datos sobre la base de datos de una determinada aplicación. Además, crea la posibilidad obtener información perteneciente al sistema de manejo de base de datos e inclusive en algunos casos emitir comandos al sistema operativo del servidor donde se aloja la base de datos.

Las maneras de efectuar inyección SQL son las siguientes:

- Se ingresan datos a un programa de una fuente que no es fidedigna.
- Los datos que se envían a una aplicación construyen dinámicamente un *query*.

Un ataque realizado a una aplicación, busca efectuar algún daño en la misma. Las cualidades que se afectan en una aplicación al sufrir un ataque de inyección SQL, son las siguientes:

- Confidencialidad: la información que maneja una aplicación es sensible, por lo que la exposición de la misma representa un gran problema, tanto para el usuario final como para la entidad que la utiliza.
- Autenticación: los usuarios que utilizan un sistema deben acceder a la información sobre la que poseen permisos. Por lo tanto si acceden a información de otro usuario, crea un gran problema de seguridad.
- Autorización: en una aplicación existen roles, sobre las actividades que un usuario tiene permiso de realizar. Si un usuario tiene la posibilidad de sobrepasar los roles, crearía la opción de que se realicen cambios sin tener los permisos para hacerlo.
- Integridad: una aplicación requiere que la información almacenada no sea cambiada sin los procesos creados con este fin. De manera que al momento de realizar transacciones, estas no se vean afectadas.

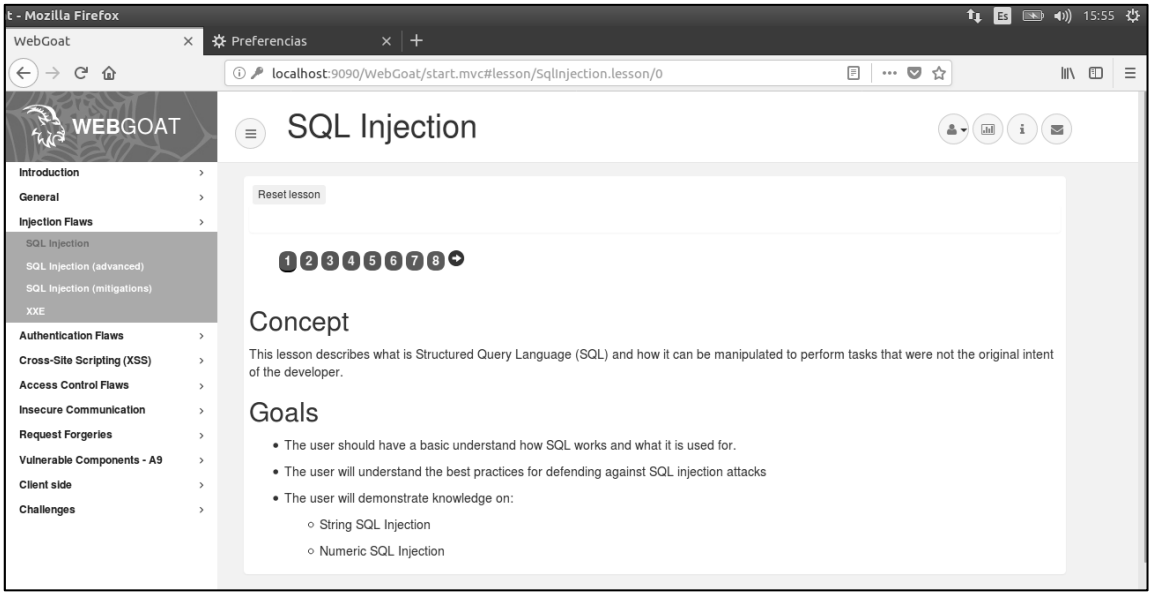
Los ataques de inyección SQL realizan las siguientes acciones para afectar al sistema que es víctima del ataque:

- Suplantar identidades.
- Alterar datos existentes.
- Anular transacciones.
- Cambiar balances.
- Revelar información sensible.
- Volver inaccesibles los datos.
- Eliminar datos.
- Obtener el rol de administrar la base de datos.

3.1.1. Funcionamiento del ataque

La aplicación WebGoat permite realizar pruebas de este ataque de manera que se entienda el funcionamiento y así determinar cómo evitarlo. La lección que abarca los elementos básicos del ataque, se encuentra bajo el apartado *Injection Flaws* (defectos de inyección), el nombre de la lección es *SQL Injection* (inyección SQL). En la figura 24 se observa la lección en la aplicación WebGoat.

Figura 24. **Lección inyección SQL, WebGoat**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 29 de abril de 2018.

La lección inyección SQL en WebGoat cuenta con 8 partes que se encuentran divididas de la siguiente manera:

- 6 partes de contenido.
- 2 partes de ejercicios.

Los objetivos a completar con la lección son los siguientes:

- Comprensión básica de SQL y como se usa.
- Las mejores prácticas para defenderse de la inyección SQL.
- Obtener conocimiento de 2 tipos de validaciones que permiten la inyección SQL:
  - Numérica.
  - Cadena.

La lección de WebGoat ofrece ejemplos de cómo estos ataques se llevan a cabo. El ataque se da al aprovechar 2 vulnerabilidades que se presentan al crear dinámicamente *querys* para las diferentes validaciones que se realizan en una aplicación, como el ingreso de usuarios.

En caso de usar una validación que conlleva cadenas de texto, como se muestra a continuación:

*"select * from users where name = '" + userName + "'";*.

También afecta realizando validaciones con números como se muestra a continuación:

*"select * from users where employee_id = " + userID;*.

La manera en que estas validaciones se utilizan para un ataque es añadiendo texto adicional a los valores enviados del cliente al servidor, como se muestra a continuación:

- userName = Smith' or '1'='1.

- *userName* =' or 1=1 –.
- *userID* = 1234567 or 1=1.
- *UserName = Smith';drop table users; truncate audit_log;--.*

De manera que al momento de recibir los valores el servidor, la consulta que se realiza a la base de datos queda conformada como se muestra a continuación:

- *select * from users where name = 'Smith' or '1' = '1'.*
- *select * from users where name = 'Smith' or TRUE.*
- *select * from users where employee_id = 1234567 or 1=1.*

Esto implica que la base de datos retorne todos los valores contenidos en la tabla a la que se realiza la consulta.

3.1.2. Probando la inyección SQL

La aplicación WebGoat presenta ejercicios para realizar pruebas de inyección SQL. Las cuales están dividas en 2 y se analizan a continuación.

3.1.2.1. Inyección SQL con validaciones de cadena

El ejercicio consiste en realizar una inyección SQL para obtener todos los datos de la tabla de usuarios de la aplicación, se menciona que la forma en que se realiza la validación es la siguiente:

*"select * from users where name = '" + userName + "'";.*

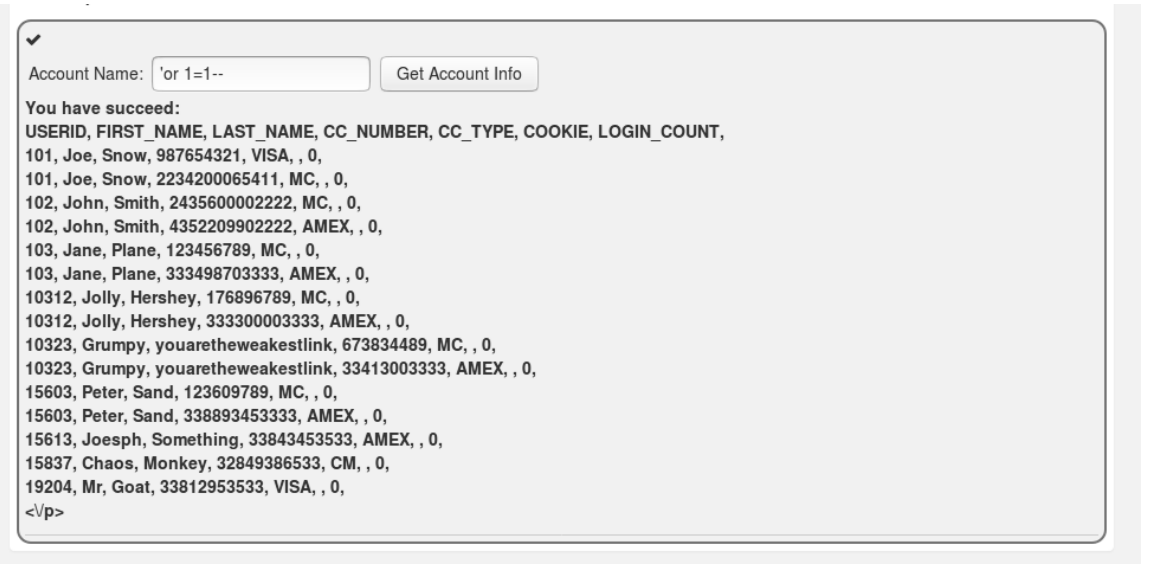
Adicional el ejercicio presenta la opción de ingresar el usuario Smith en el campo de texto que permite el ingreso de valores, para obtener la información de un usuario existente. Como se muestra en la figura 25.

Figura 25. **Ejercicio 1 inyección SQL prueba**

Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 01 de mayo de 2018.

Para obtener todos los valores que se registran en la tabla usuarios, existe la posibilidad de usar alguno de los ejemplos mencionados anteriormente. En la figura 26 se muestra como se completa el ejercicio de forma satisfactoria.

Figura 26. **Ejercicio 1 inyección SQL resuelto**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 1 de mayo de 2018.

De esta manera el primer ejercicio se encuentra finalizado exitosamente. Al obtener toda la información de los usuarios que guarda la aplicación.

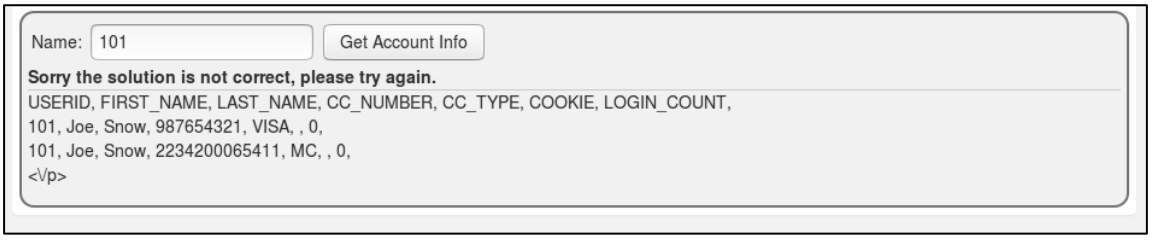
3.1.2.2. Inyección SQL con validaciones de número

El ejercicio consiste en realizar una inyección SQL para obtener todos los datos de la tabla de usuarios de la aplicación, se menciona que la forma en que se realiza la validación es la siguiente:

*"select * from users where employee_id = " + userID;.*

Adicional el ejercicio presenta la opción de ingreso del código de usuario 101 en el campo de texto que permite el ingreso de valores, para obtener la información de un usuario existente. Como se muestra en la figura 27.

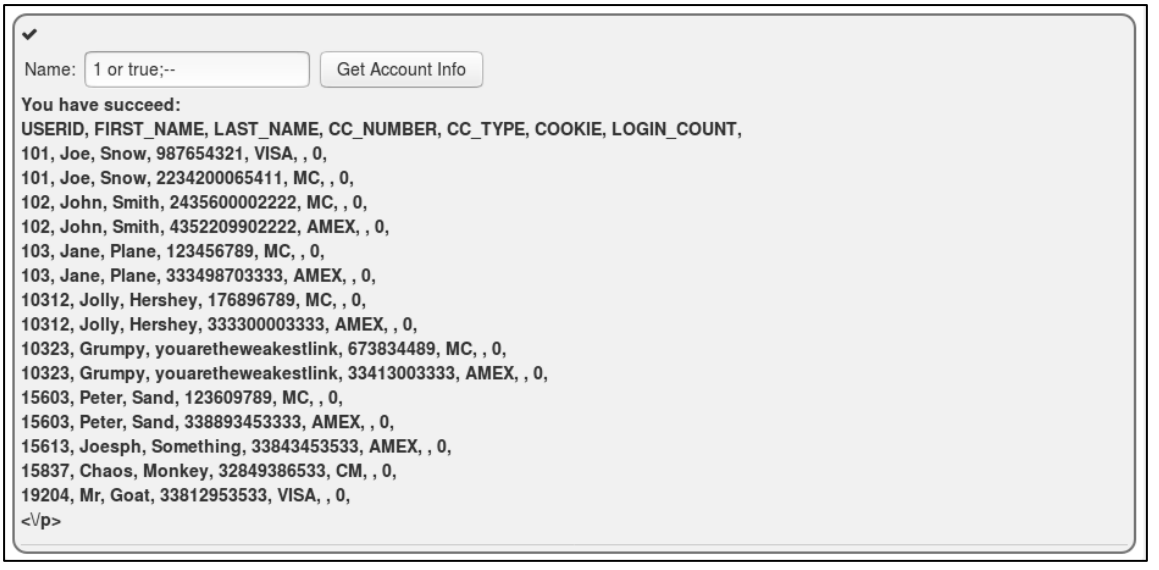
Figura 27. **Ejercicio 2 inyección SQL prueba**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 01 de mayo de 2018.

Para obtener todos los valores que se registran en la tabla usuarios, existe la opción de utilizar alguno de los ejemplos mencionados anteriormente. En la figura 28 se muestra, como se completa el ejercicio de forma satisfactoria.

Figura 28. **Ejercicio 2 inyección SQL resuelto**



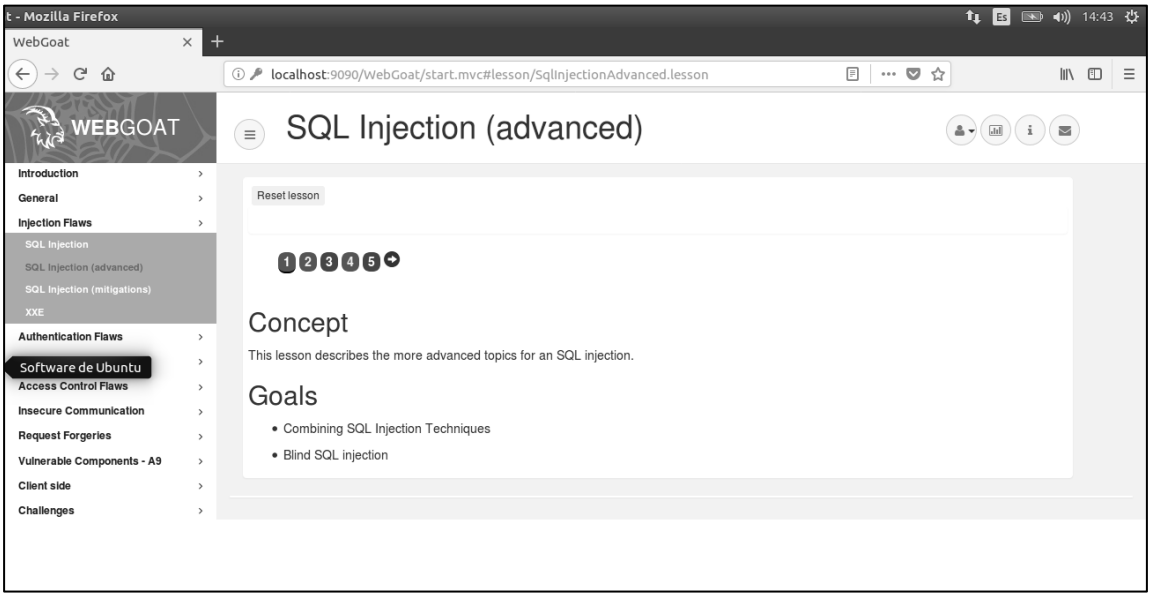
Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 1 de mayo de 2018.

De esta manera el segundo ejercicio se encuentra finalizado exitosamente. Al obtener toda la información de los usuarios que posee la aplicación.

3.1.3. Inyección SQL avanzada

Para realizar un ataque de inyección SQL es importante conocer los caracteres especiales que existen en el lenguaje SQL, de manera que se aprovechen para la obtención de información. La aplicación WebGoat, cuenta con una lección dedicada a esto, bajo el apartado *Injection Flaws* (defectos de inyección), el nombre de la lección es *SQL Injection advanced* (inyección SQL avanzado). Como se observa en la figura 29.

Figura 29. **Lección inyección SQL avanzada**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 1 de mayo de 2018.

3.1.3.1. Caracteres especiales SQL

A continuación en la tabla IV se listan los caracteres especiales que una consulta SQL incluye:

Tabla IV. **Caracteres especiales SQL**

CARÁCTER	USO
/* */	Se usan para realizar comentarios multilínea.
-- , #	Se usan para realizar comentarios de una línea.
;	Permite ejecutar consultas en cadena.
', +, ||	Se usan para concatenación de cadenas.
Char ()	Se utiliza para ingresar caracteres especiales, indicando el número del carácter en ASCII.

Fuente: elaboración propia.

Ejemplos de uso de caracteres especiales para inyección SQL:

- *Select * from users where name = 'admin' **--**and pass = 'pass'.*
- *Select * from users where name = 'admin' --and pass = 'pass'.*
- *Select * from users where name = '+char(27) or 1=1.*

3.1.3.2. Inyección ciega de SQL

Es un tipo de inyección SQL que se basa en realizar consultas a la base de datos de verdadero y falso, determinando la respuesta dada por medio del comportamiento de la aplicación. Este ataque se realiza en aplicaciones web, que han sido diseñadas de manera que devuelvan errores genéricos, los cuales no muestran ninguna información particular de la base de datos. Sin embargo, a pesar de mostrar errores genéricos la inyección SQL no se ha mitigado.

La diferencia entre una inyección SQL normal y una inyección ciega SQL, es que en una inyección normal los mensajes de error se muestran y dan información suficiente para reconocer como funciona la consulta. Mientras que en una inyección SQL ciega se debe formular preguntas a la base de datos booleanas para observar el comportamiento. Debido a que la aplicación no muestra ningún mensaje de error que sea útil, este tipo de ataque es más difícil de realizar.

¿Cómo aprovechar el que una aplicación sea vulnerable a inyección ciega de SQL si no se obtienen valores? El realizar preguntas de verdadero y falso, brinda la posibilidad de obtener información de la base de datos. Por ejemplo: el tipo de base de datos, la versión que posee la base de datos, si se tiene algún usuario genérico, entre otras. De manera que el ataque busca exponer información sensible de la base de datos.

Al obtener información como la mencionada en el párrafo anterior, abre paso a extraer información de tablas en específico e inclusive volcar la base de datos, al obtener usuarios con privilegios de administrador. Aclarando que esto dependerá de que los roles y permisos del usuario de base de datos no estén configurados correctamente, regularmente sucede que el usuario de la aplicación tenga acceso total a la información de la base de datos.

3.1.3.2.1. Ejemplo

Para el ejemplo se realizará una pregunta booleana a la base de datos de una aplicación ficticia, para determinar si la página que se está atacando es susceptible a inyección SQL. Suponiendo que la página a atacar envía una petición de la siguiente manera:

https://my-shop.com?article=4.

De manera que al realizar la consulta a la base de datos, esta quedaría de la siguiente forma:

*select * from articles where article_id = 4.*

Para determinar si la aplicación es vulnerable a inyección SQL, se debe agregar un valor como el siguiente a la variable enviada al servidor: *AND* 1=1. De manera que la consulta a la base de datos sea la siguiente:

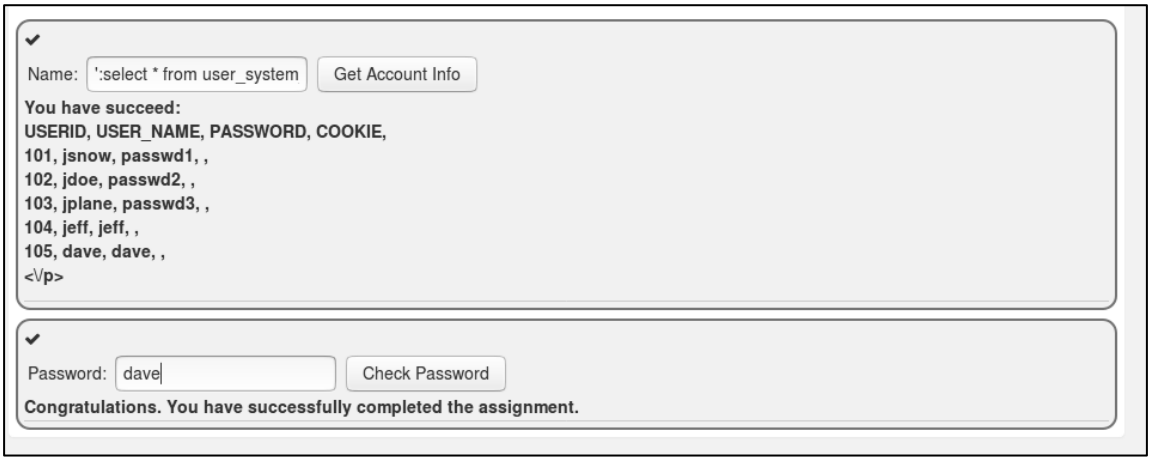
*select * from articles where article_id = 4 AND 1=1.*

Si al momento de cambiar el valor enviado, el servidor responde con la misma página que con el valor original, significa que la página es vulnerable para realizar inyección ciega de SQL. Posterior existe la posibilidad de cambiar el valor enviado por el siguiente: https://my-shop.com?article=4 AND 1=2. De manera que no debe retornar nada debido a que el resultado de la sentencia es falso.

3.1.4. Probando inyección SQL avanzada

La lección de inyección SQL avanzada de la aplicación WebGoat, presenta ejercicios prácticos para realizar pruebas de inyección SQL. El ejercicio indica que existe una tabla en la aplicación con el nombre: *user_system_data*, de la cual se debe obtener el contenido mediante una consulta anidada al valor enviado por un cuadro de texto. Posterior a obtener la información de la tabla, el ejercicio requiere que en el segundo cuadro de texto, se ingrese la contraseña del usuario Dave, para dar por finalizado el ejercicio. La figura 30 muestra el resultado del ejercicio realizado.

Figura 30. **Ejercicio 1 inyección SQL avanzado resuelto**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 1 de mayo de 2018.

Una vez obtenida la información se observa que la contraseña para el usuario dave, la contraseña a usar es dave. Con esto se finaliza el ejercicio.

3.1.5. Mitigación de inyección SQL

Los ataques de inyección SQL son bastante comunes, por lo tanto es importante tener una guía clara, simple y procesable para evitarlos. Las razones por las que estos se producen son las siguientes:

- Vulnerabilidades existentes.
- El atractivo del objetivo.

Por lo tanto se deben mitigar las vulnerabilidades que un sistema presenta, para evitar que el sistema sea víctima de un ataque de inyección SQL. Estas vulnerabilidades se crean cuando en la aplicación se desarrollan SQL dinámicos que incluyen información que el usuario debe enviar. Por lo que se toma en cuenta la opción de que el usuario envíe una consulta dentro de los valores que la aplicación espera.

3.1.5.1. Uso de consultas parametrizadas

Consiste en crear consultas hacia la base de datos de manera que los parámetros se ingresen posteriormente al definir la consulta SQL. Esto hace que las consultas a utilizar sean más fáciles de escribir y de entender que una consulta creada dinámicamente. Definir la consulta SQL, con los datos que se quieren obtener de la base de datos y posteriormente agregar la información enviada por el usuario como parámetros, permite a la base de datos distinguir entre código SQL y datos, no importando el tipo de fuente que represente al usuario.

El usar consultas parametrizadas asegura que si existe un atacante, este no tenga la posibilidad de cambiar la intención de los datos que espera el servidor del cliente. A pesar de no detener el envío de los datos de parte del atacante, el servidor interpretará estos datos como un parámetro completo que será comparado contra una columna en específico de la base de datos. Por lo tanto si la información transmitida tiene alguna consulta para ejecutar no se realizará.

En muy raras ocasiones pasara que las consultas parametrizadas afecten el rendimiento de una aplicación. De darse el caso se presentan las siguientes recomendaciones:

- Realizar validaciones completas de los datos ingresados por el usuario.
- Evitar que el usuario ingrese caracteres que permitan ejecutar instrucciones en la base de datos.

A continuación en las figuras 31, 32 y 33 se presentan ejemplos de consultas parametrizadas, en diferentes lenguajes de programación.

Figura 31. **Consulta parametrizada usando Java**

```
String custname = request.getParameter("customerName"); // This should REALLY be validated too
// perform input validation to detect attacks
String query = "SELECT account_balance FROM user_data WHERE user_name = ? ";

PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

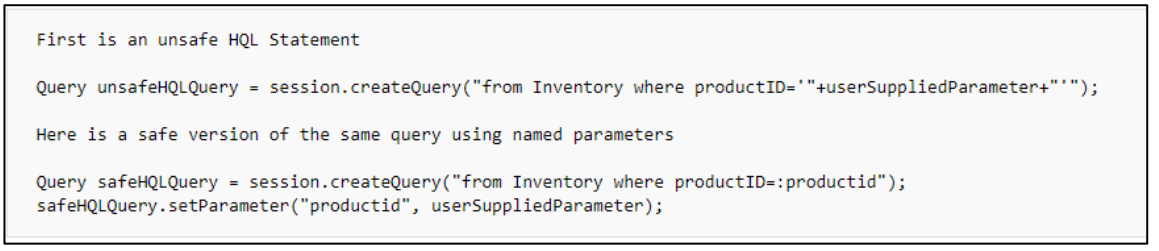
Fuente: OWASP. https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet. Consulta: 05 de mayo de 2018.

Figura 32. **Consulta parametrizada usando C#**

```
String query =
        "SELECT account_balance FROM user_data WHERE user_name = ?";
try {
       OleDbCommand command = new OleDbCommand(query, connection);
       command.Parameters.Add(new OleDbParameter("customerName", CustomerName Name.Text));
       OleDbDataReader reader = command.ExecuteReader();
       // …
} catch (OleDbException se) {
       // error handling
}
```

Fuente: OWASP. https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet. Consulta: 05 de mayo de 2018.

Figura 33. **Consulta parametrizada usando Hibernate**



```
First is an unsafe HQL Statement

Query unsafeHQLQuery = session.createQuery("from Inventory where productID='"+userSuppliedParameter+"'");

Here is a safe version of the same query using named parameters

Query safeHQLQuery = session.createQuery("from Inventory where productID=:productid");
safeHQLQuery.setParameter("productid", userSuppliedParameter);
```

Fuente: OWASP. https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet. Consulta: 05 de mayo de 2018.

3.1.5.2. Uso de procedimientos almacenados

Los procedimientos almacenados son una de las formas de defenderse de la inyección SQL, si estos se usan de la manera correcta. Debido a que también existe la posibilidad de tener procedimientos almacenados mal desarrollados que resultan ser inseguros. Por lo tanto, se requiere que el desarrollador cree procedimientos almacenados usando sentencias SQL parametrizadas, de manera que solo fallen si el programador realiza algo fuera de lo normal.

La diferencia que poseen los procedimientos almacenados con las consultas parametrizadas, es que son creados y almacenados directamente en la base de datos, para posteriormente ser invocados por la aplicación. Es imprescindible que dentro de los procedimientos almacenados no se generen consultas SQL dinámicas debido a que esto los vuelve inseguros. Si es algo que no existe la posibilidad de evitar, los procedimientos almacenados deben validar los parámetros de entrada, de manera que se aseguren que no contienen información que afecte los datos de la base de datos.

Otro factor importante a validar es que el usuario que se use para ejecutar los procedimientos almacenados a nivel de base de datos, este bien configurado. De manera que solo tenga los permisos necesarios para la ejecución de los procedimientos almacenados. Esto evitará que si por algún motivo se llega a modificar el procedimiento almacenado o a acceder al usuario, no se corra riesgo de perder información o la estructura de la base de datos.

A continuación en las figuras 34 y 35 se muestran ejemplos de llamado de funciones en 2 lenguajes distintos de programación.

Figura 34. **Procedimiento almacenado en Java**

```
String custname = request.getParameter("customerName"); // This should REALLY be validated
try {
       CallableStatement cs = connection.prepareCall("{call sp_getAccountBalance(?)}");
       cs.setString(1, custname);
       ResultSet results = cs.executeQuery();
       // … result set handling
} catch (SQLException se) {
       // … logging and error handling
}
```

Fuente: OWASP. https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet. Consulta: 05 de mayo de 2018.

Figura 35. **Procedimiento almacenado en VB.NET**

```
Try
       Dim command As SqlCommand = new SqlCommand("sp_getAccountBalance", connection)
       command.CommandType = CommandType.StoredProcedure
       command.Parameters.Add(new SqlParameter("@CustomerName", CustomerName.Text))
       Dim reader As SqlDataReader = command.ExecuteReader()
       ' …
Catch se As SqlException
       ' error handling
End Try
```

Fuente: OWASP. https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet. Consulta: 05 de mayo de 2018.

3.1.5.3. Validación de lista blanca

Existen ocasiones donde se necesita los valores que el usuario ingresa para establecer el nombre de la tabla o columnas que se desean consultar. En este caso lo que se debe hacer es validar la entrada. De manera que la aplicación se asegure que el valor enviado no contenga algún contenido que cree la posibilidad de perjudicar la base de datos. Por lo que los valores enviados por el usuario se deben validar antes de ejecutarse en la base de datos, para determinar si conlleva la información esperada o se tiene que informar que los valores ingresados no son aceptados por la aplicación. A continuación se muestra un ejemplo en la figura 36.

Figura 36. **Validación de ingreso para nombre de tabla.**

```
String tableName;
switch(PARAM):
  case "Value1": tableName = "fooTable";
                 break;
  case "Value2": tableName = "barTable";
                 break;
    ...
  default      : throw new InputValidationException("unexpected value provided for table name");
```

Fuente: OWASP. https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet. Consulta: 05 de mayo de 2018.

Una vez validado el valor ingresado para el nombre de la tabla, este se agrega directamente al resto de la consulta SQL, ya que se descartó que contenga información perjudicial. Cada vez que la entrada ingresada por el usuario, se convierta a un valor que no sea del tipo cadena, por ejemplo: un número o un valor booleano, se debe realizar. De esta forma se usa el valor convertido para anexar la información a la consulta SQL y así asegurar la información que se manda a la base de datos.

3.2. Inyección XXE

Ataque que se realiza contra una aplicación que tiene como función recibir e interpretar entradas por algún analizador de XML. El ataque se realiza cuando la forma en que los datos son recibidos, no valida que se incluyan entidades externas. Esto permite que en los archivos XML se envíen valores que realicen la lectura de archivos locales, denegación de servicio, mapeo de red interna, entre otros. De manera que la vulnerabilidad se crea cuando se configura de forma inadecuada las librerías utilizadas para interpretar el XML enviado a la aplicación.

XML es un formato de datos ampliamente usado que se encuentra en diferentes elementos informáticos, como lo son servicios web (*Web Services*), además de documentos del tipo: Docx, HTML y XML. A continuación se muestra en la figura 37, un ejemplo de una aplicación Web sencilla que soporta comunicación por medio del uso de XML. Se ejemplifica la solicitud del cliente por medio de un documento XML y la respuesta que el servidor regresa.

Figura 37. **Comunicación XML sencilla**

Fuente: Acunetix. https://www.acunetix.com/blog/articles/xml-external-entity-xxe-vulnerabilities/. Consulta: 10 de mayo de 2018.

En el ejemplo anterior se observa la forma en que se declaran atributos, elementos y texto en un documento XML. Un documento que usa este formato de datos, también es capaz de especificar un grupo de declaraciones de marcado que definen la estructura que este tendrá y el tipo, de manera que el analizador sea capaz de validar la corrección del documento XML antes de que sea procesado. Esto se realiza de 2 maneras:

- Definición de esquema XML (XSD).
- Definición de tipos de datos (DTD).

3.2.1. Definición de esquema XML

Una definición de esquema XML (XSD por las siglas en inglés *XML Schema Definition*), es un documento que define las reglas y restricciones para la estructura de documentos XML. Una XSD se utiliza para validar el contenido que posee un documento XML, debido a que incluye la definición de los elementos que pertenecen al archivo, el tipo de datos que es permitido usar y las restricciones que tienen los valores recibidos dentro del documento. En la figura 38 se muestra un ejemplo de un documento de definición.

Figura 38. **Ejemplo documento XSD**

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="https://www.w3schools.com"
xmlns="https://www.w3schools.com"
elementFormDefault="qualified">

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Fuente: W3schools. https://www.w3schools.com/xml/schema_howto.asp. Consulta: 12 de mayo de 2018.

Como se observa en la figura 38 el documento de definición XSD, propiamente esta hecho en lenguaje XML. Con este ejemplo se limitan los valores que recibirá la aplicación que lo implemente, tanto como la cantidad, como la información que poseen. En la figura 39 se muestra un ejemplo de la implementación de un documento de definición XSD en un documento XML, para definir la estructura que llevará el contenido.

Figura 39. **Ejemplo de uso de XSD en XML**

```
<?xml version="1.0"?>

<note
xmlns="https://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://www.w3schools.com/xml/note.xsd">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Fuente: W3schools. https://www.w3schools.com/xml/schema_howto.asp. Consulta: 12 de mayo de 2018.

En los siguientes apartados se analizan los componentes de una definición de esquema XML.

3.2.1.1. Elementos simples y tipos de datos XSD

En el esquema de definición se establecen los elementos que conforman un documento XML y los tipos de datos que acepta. Un elemento simple es un elemento XML que solo contiene texto, no tiene la opción de contener otros elementos ni atributos. El texto que un elemento contiene es de diferentes tipos, tanto de los predefinidos por el lenguaje o personalizado por el desarrollador. El formato para definir los elementos es el siguiente:

<xs:*element name*="identificador_del_elemento" *type*="tipo_de_datos" />

Los tipos de elemento predefinidos y más utilizados se listan a continuación:

- *xs:string*: usado para valores que contienen cadenas de caracteres.
- *xs:decimal*: usado para valores numéricos que incluyen decimales.
- *xs:integer:* usado para valores numéricos enteros.
- *xs:boolean*: usado para valores de verdadero y falso.
- *xs:date*: usado para valores que especifican una fecha.
- *xs:time*: usado para valores que especifican un tiempo, usando fecha y hora.

En la figura 40 se muestran ejemplos de elementos con los valores que contiene y la respectiva definición.

Figura 40. **Ejemplo de elementos XML**

Fuente: W3schools. https://www.w3schools.com/xml/schema_simple.asp. Consulta: 12 de mayo de 2018.

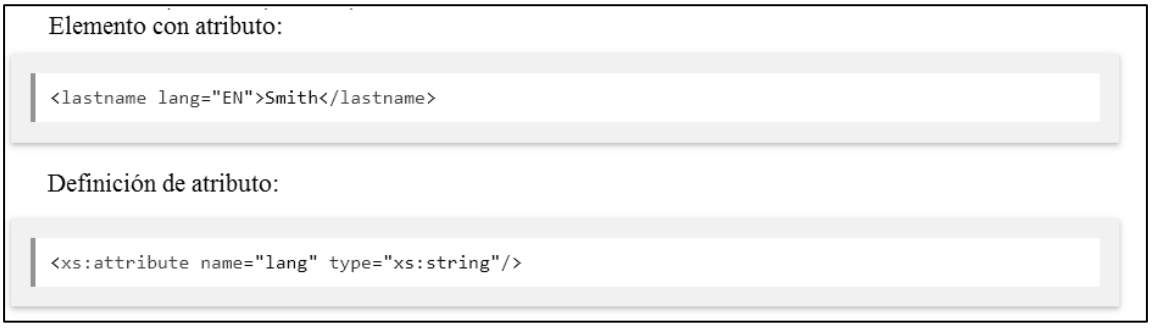
3.2.1.2. Atributos XSD

Un atributo es una característica de un elemento compuesto, este provee información adicional del elemento. Es importante aclarar que los elementos simples no contienen atributos. La definición de un atributo tiene la misma estructura que se usa al definir un elemento simple, especificando un nombre y tipo de atributo, seleccionado de los mencionados anteriormente o personalizado por el desarrollador. A continuación se muestra la estructura:

<xs:*attribute name*="identificador_del_elemento" *type*="tipo_de_datos"/>

En la figura 41 se muestra un ejemplo de un elemento que posee un atributo y la definición del atributo:

Figura 41. **Ejemplo de elemento con atributo**



Fuente: W3schools. https://www.w3schools.com/xml/schema_simple.asp. Consulta: 12 de mayo de 2018.

3.2.1.3. Restricciones XSD

Las restricciones en XSD se utilizan para definir los valores que son aceptados en elementos y atributos, cuando se aplican en elementos también son llamadas facetas. Por lo regular se usan en la creación de tipos de atributos y elementos personalizados, de manera que se adapten a las necesidades que tenga la aplicación que recibe el XML y así se permita solo el ingreso de valores que posean un significado y no sean perjudiciales para la aplicación. En las figuras 42 y 43 se ejemplifican las distintas formas de utilizarlas.

Figura 42. **Restricción de valores de un elemento**

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Fuente: W3schools. https://www.w3schools.com/xml/schema_facets.asp. Consulta: 12 de mayo de 2018.

Figura 43. **Creación de tipos usando restricciones**

```
<xs:element name="car" type="carType"/>

<xs:simpleType name="carType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="Golf"/>
    <xs:enumeration value="BMW"/>
  </xs:restriction>
</xs:simpleType>
```

Fuente: W3schools. https://www.w3schools.com/xml/schema_facets.asp. Consulta: 12 de mayo de 2018.

A continuación en la tabla V se listan las diferentes características que se utilizan para definir restricciones:

Tabla V. **Restricciones de XSD**

Restricción	Descripción
enumeration	Define una lista de valores que son aceptados.
fractionDigits	Específica la cantidad de dígitos que son permitidos en un valor, debe ser un valor mayor o igual a 0.
length	Especifica la cantidad exacta de caracteres o elementos que son permitidos. Debe colocarse un valor igual o mayor a 0.
maxExclusive	Especifica el valor del límite superior para valores numéricos, el valor debe ser menor al indicado.
maxInclusive	Especifica el valor del límite superior para valores numéricos, el valor debe ser menor o igual al indicado.
maxLength	Especifica la cantidad máxima de caracteres que son aceptados, debe indicarse un valor mayor o igual a 0.
minExclusive	Especifica el valor del límite inferior para valores numéricos, el valor debe ser mayor al indicado.
minInclusive	Especifica el valor del límite inferior para valores numéricos, el valor debe ser mayor o igual al indicado.
minLength	Especifica la cantidad mínima de caracteres que son aceptados, debe indicarse un valor mayor o igual a 0.
Pattern	Define la secuencia exacta de caracteres que son aceptados.
totalDigits	Especifica el número exacto de dígitos permitidos. El valor debe ser mayor a 0.
whiteSpace	Especifica cómo son manejados los espacios en blanco (saltos de línea, tabulaciones y espacios).

Fuente: W3schools. https://www.w3schools.com/xml/schema_facets.asp. Consulta: 12 de mayo de 2018.

3.2.1.4. Elementos compuestos

Un elemento compuesto contiene otros elementos y atributos. Existen 4 tipos de elementos compuestos, que se listan a continuación:

- Elementos vacíos.
- Elemento que contiene solamente otros elementos.
- Elemento que contiene solamente texto.
- Elemento que contiene textos y elementos.

- Elementos compuestos vacíos: es un elemento complejo que está compuesto solamente por atributos en la figura 44 se muestra un ejemplo de un elemento complejo vacío.

Figura 44. **Ejemplo de elemento complejo vacío**

```
<product prodid="1345" />
```

Fuente: W3schools. https://www.w3schools.com/xml/schema_complex_empty.asp. Consulta: 13 de mayo de 2018.

En la figura 45 se muestra la definición completa para el elemento del ejemplo anterior, en la cual se describe la restricción usada para el atributo *prodid* del elemento *product*.

Figura 45. **Definición completa elemento *product***

```
<xs:element name="product">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:integer">
        <xs:attribute name="prodid" type="xs:positiveInteger"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

Fuente: W3schools. https://www.w3schools.com/xml/schema_complex_empty.asp. Consulta: 13 de mayo de 2018.

La definición del elemento *product*, es posible realizarla de una manera más compacta, como se muestra en la figura 46.

Figura 46. **Definición compacta elemento *product***

```
<xs:element name="product">
  <xs:complexType>
    <xs:attribute name="prodid" type="xs:positiveInteger"/>
  </xs:complexType>
</xs:element>
```

Fuente: W3schools. https://www.w3schools.com/xml/schema_complex_empty.asp. Consulta: 13 de mayo de 2018.

Existe otra forma para definir el elemento *product*, utilizando un tipo de elemento personalizado, de esta manera es posible usarlo en diferentes elementos, realizando la llamada al tipo de producto. En la figura 47 se muestra la definición del elemento *product*, usando un tipo llamado *prodtype*.

Figura 47. **Definición con tipo elemento *product***

```
<xs:element name="product" type="prodtype"/>

<xs:complexType name="prodtype">
  <xs:attribute name="prodid" type="xs:positiveInteger"/>
</xs:complexType>
```

Fuente: W3schools. https://www.w3schools.com/xml/schema_complex_empty.asp. Consulta: 13 de mayo de 2018.

- Elemento que contiene solamente otros elementos: es un elemento complejo que está compuesto por otros elementos. En la figura 48 se muestra un ejemplo de este tipo de elemento complejo.

Figura 48. **Ejemplo de elemento compuesto por elementos**

```
<person>
  <firstname>John</firstname>
  <lastname>Smith</lastname>
</person>
```

Fuente: W3schools. https://www.w3schools.com/xml/schema_complex_empty.asp. Consulta: 13 de mayo de 2018.

En la figura 49 se muestra la definición del elemento *person*, con los elementos internos: *firstname* y *lastname*.

Figura 49. **Definición de elemento *person***

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Fuente: W3schools. https://www.w3schools.com/xml/schema_complex_empty.asp. Consulta: 13 de mayo de 2018.

En este caso, es importante recalcar que los elementos dentro del elemento *person*, deben aparecer en el orden que muestra la definición.

- Elemento que contiene solamente texto: este tipo de elemento complejo se compone únicamente por contenido simple, como texto y atributos, por lo que se debe usar un elemento simple con la restricción del contenido que se desea. En la figura 50 se observa un ejemplo de elemento de este tipo.

Figura 50. **Ejemplo de elemento compuesto por texto**

```
<shoesize country="france">35</shoesize>
```

Fuente: W3schools. https://www.w3schools.com/xml/schema_complex_text.asp. Consulta: 13 de mayo de 2018.

En la figura 51 se muestra la definición para el elemento complejo *shoesize*, y el atributo *country*.

Figura 51. **Definición de elemento *shoesize***

```
<xs:element name="shoesize">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:integer">
        <xs:attribute name="country" type="xs:string" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Fuente: W3schools. https://www.w3schools.com/xml/schema_complex_text.asp. Consulta: 13 de mayo de 2018.

- Elemento que contiene textos y elementos: es un elemento complejo, que está compuesto por elementos, texto y atributos, también llamado elemento complejo mixto. En la figura 52 se muestra un ejemplo de este tipo de elemento.

Figura 52. **Ejemplo de elemento complejo mixto**

```
<letter>
  Dear Mr.<name>John Smith</name>.
  Your order <orderid>1032</orderid>
  will be shipped on <shipdate>2001-07-13</shipdate>.
</letter>
```

Fuente: W3schools. https://www.w3schools.com/xml/schema_complex_mixed.asp. Consulta: 13 de mayo de 2018.

En la figura 53 se observa la definición del elemento *letter*, con los diferentes componentes que posee.

Figura 53. **Definición elemento *letter***

```
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="orderid" type="xs:positiveInteger"/>
      <xs:element name="shipdate" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Fuente: W3schools. https://www.w3schools.com/xml/schema_complex_mixed.asp. Consulta: 13 de mayo de 2018.

3.2.2. Definición de tipos de datos

Es una definición de tipo de documento XML, comúnmente llamado DTD (por las siglas en inglés *Document Type Definition*). Se utiliza para definir la estructura, elementos y atributos que se permiten en un documento XML que recibe una aplicación, de manera que se envíen solo datos que sean útiles para la misma. Es utilizado como estándar para el manejo de documentos XML, entre grupos de personas independientes y ajenas entre sí.

Existen 2 maneras de definir la estructura de un documento XML, usando DTD. Las cuales se listan a continuación:

- Declaración interna.
- Declaración externa.

Cuando se realiza una definición por declaración interna, la DTD debe estar adentro del archivo XML, dentro de la definición <!DOCTYPE>. En la figura 54 se muestra un ejemplo de definición interna DTD.

Figura 54. **Ejemplo de definición interna DTD**

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

Fuente: W3schools. https://www.w3schools.com/xml/xml_dtd_intro.asp. Consulta: 13 de mayo de 2018.

Cuando se realiza una definición DTD en un archivo externo al documento XML, dentro de la definición <!DOCTYPE> debe contener una referencia al archivo. En la figura 55, se muestra un ejemplo de definición externa DTD y en la figura 56 se muestra el archivo DTD.

Figura 55. **Ejemplo de definición externa DTD**

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Fuente: W3schools. https://www.w3schools.com/xml/xml_dtd_intro.asp. Consulta: 13 de mayo de 2018.

Figura 56. **Ejemplo de archivo DTD**

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

Fuente: W3schools. https://www.w3schools.com/xml/xml_dtd_intro.asp. Consulta: 13 de mayo de 2018.

3.2.2.1. Componentes básicos de un documento XML

A continuación se listan los componentes básicos que poseen todos los documentos XML, vistos desde una definición DTD:

- Elementos.
- Atributos.

- Entidades.
- PCDATA.
- CDATA.

- Elementos

Los elementos en un documento XML y HTML son los bloques de construcción principales y esenciales para el manejo de información. Un elemento tiene la posibilidad de estar vacío o de tener entre los componentes elementos o texto. En la figura 57 se muestran ejemplos de uso de elementos, que existe la posibilidad de aplicar tanto para documento HTML, como para documentos XML.

Figura 57. **Ejemplo de uso de elementos**

Fuente: W3schools. https://www.w3schools.com/xml/xml_dtd_building.asp. Consulta: 14 de mayo de 2018.

- Atributos

Los atributos se utilizan para proveer información adicional acerca de los elementos. Estos se colocan dentro de la etiqueta de un elemento y siempre se declaran en par incluyendo el nombre y el valor. En la figura 58, se observa un ejemplo de un elemento que posee un atributo, que añade información sobre la locación del recurso.

Figura 58. **Ejemplo de uso de atributos**

```
<img src="computer.gif" />
```

Fuente: W3schools. https://www.w3schools.com/xml/xml_dtd_building.asp. Consulta: 15 de mayo de 2018.

- Entidades

Las entidades son caracteres que tienen un significado especial en un documento XML, como por ejemplo el signo de menor que (<), que indica el inicio de una etiqueta XML. Las entidades se expanden cuando un analizador de XML, analiza un documento XML. En la tabla VI se muestran entidades predefinidas en XML.

Tabla VI. **Entidades predefinidas XML**

Referencia de entidad	Carácter
<	<
>	>
&	&
"	"
'	'

Fuente: W3schools. https://www.w3schools.com/xml/xml_dtd_building.asp. Consulta: 15 de mayo de 2018.

- PCDATA

PCDATA significa datos de caracteres analizados. Es texto que es interpretado por un analizador XML, con el fin de buscar entidades y marcado, de manera que las etiquetas que se encuentren en el texto y las entidades se expandan. Se debe tomar en cuenta que los datos a analizar no deben contener ningún carácter de los mostrados en la tabla VI, sino que deben representarse usando la referencia de entidad.

- CDATA

Este tipo de dato aplica para texto que no será interpretado por un analizador, las etiquetas que se encuentren en el texto no se trataran como marcas y las entidades no se expandirán.

3.2.2.2. Elementos DTD

Para una definición DTD los elementos deben declararse siguiendo la sintaxis que se muestra en la figura 59.

Figura 59. **Sintaxis de declaración de elementos DTD**

```
<!ELEMENT element-name category>
or
<!ELEMENT element-name (element-content)>
```

Fuente: W3schools. https://www.w3schools.com/xml/xml_dtd_building.asp. Consulta: 14 de mayo de 2018.

En donde *element-name* es el nombre que poseerá el elemento. *Category* es la categoría que el elemento tendrá. Por último *element-content*, definirá el contenido del elemento, en el cual se especificará si posee un tipo o si posee elementos como hijos. A continuación se listan los valores de categoría que existen para utilizar y la función que cumplen:

- *Empty*: Usado para declarar un elemento vacío.
- *Any:* Usado para indicar que un elemento tiene la posibilidad de contener cualquier combinación de valores analizables.

Adicional para la declaración de los elementos que componen a un elemento padre, existen diferentes símbolos que ayudan a determinar el cardinal de los elementos, estos se listan a continuación.

Tabla VII. **Símbolos para elementos hijos**

Símbolo	Uso
*****	Indica que el elemento hijo tiene 0 o más ocurrencias.
?	Indica que el elemento hijo tiene 0 o 1 ocurrencia.
|	Indica que el elemento hijo se elige entre 2 posibles elementos, obligatoriamente tiene que existir 1 de los 2.

Fuente: elaboración propia.

En la figura 60 se muestra un ejemplo de declaración de elementos hijos, usando los símbolos expuestos en la tabla VII. En donde la definición para el elemento *note* abarca múltiples opciones para representarlo, de manera que el elemento *note* está conformado por alguna de los siguientes elementos: tipo de dato carácter, elemento *to*, elemento *from*, elemento *header*, elemento *message*. De los cuales existen cero o más ocurrencias.

Figura 60. **Ejemplo de declaración de elementos hijos**

```
<!ELEMENT note (#PCDATA|to|from|header|message)*>
```

Fuente: W3schools. https://www.w3schools.com/xml/xml_dtd_elements.asp. Consulta: 16 de mayo de 2018.

3.2.2.3. Atributos DTD

En la definición DTD para documentos XML, los atributos se declaran usando la palabra reservada ATTLIST. En la figura 61 se observa la sintaxis que se sigue para declarar atributos DTD, adicional se muestran y el uso en elementos.

Figura 61. **Declaración de atributos DTD**

```
<!ATTLIST element-name attribute-name attribute-type attribute-value>

DTD example:

<!ATTLIST payment type CDATA "check">

XML example:

<payment type="check" />
```

Fuente: W3schools. https://www.w3schools.com/xml/xml_dtd_attributes.asp. Consulta: 16 de mayo de 2018.

En la tabla VIII se enumeran los valores que existen para *attribute-type*.

Tabla VIII. **Tipos de atributos**

Tipo	Descripción
CDATA	El valor es del tipo carácter.
(en1|en2|..)	El valor debe ser uno de la lista enumerada.
ID	El valor es un identificador único.
IDREF	El valor es un identificador referenciado de otro elemento.
IDREFS	El valor es una lista de identificadores referenciados.
NMTOKEN	El valor es un nombre XML válido.
NMTOKENS	El valor es una lista de nombres XML válidos.
ENTITY	El valor es una entidad.
ENTITIES	El valor es una lista de entidades.
NOTATION	El valor es un nombre de una notación.
XML	El valor es un valor predefinido de XML.

Fuente: W3schools. https://www.w3schools.com/xml/xml_dtd_attributes.asp. Consulta: 16 de mayo de 2018.

En la tabla IX se enumeran los valores que existen para *attribute-value*.

Tabla IX. **Valores de atributos**

Valor	Explicación
value	Es el valor predeterminado del atributo.
#REQUIRED	El valor del atributo es obligatorio.
#IMPLIED	El valor del atributo es opcional.
#FIXED	El valor del atributo es fijo.

Fuente: W3schools. https://www.w3schools.com/xml/xml_dtd_attributes.asp. Consulta: 16 de mayo de 2018.

3.2.2.4. Entidades DTD

Las entidades son usadas para definir accesos rápidos a caracteres especiales, de manera que las etiquetas usadas sean reemplazadas por contenido cuando el documento XML sea analizado. Para declararlas en un documento XML, existen 2 maneras de realizarlo: Externa e Interna. En la figura 62 se muestra la sintaxis para una declaración interna, junto con un ejemplo de uso.

Figura 62. **Declaración interna de entidades**

Fuente: W3schools. https://www.w3schools.com/xml/xml_dtd_entities.asp. Consulta: 17 de mayo de 2018.

En la figura 63 se muestra un ejemplo de la sintaxis para declarar externamente una entidad. Adicional se muestra un ejemplo de uso.

Figura 63. **Declaración externa de entidades**

Sintaxis:

```
<!ENTITY entity-name SYSTEM "URI/URL">
```

Ejemplo:

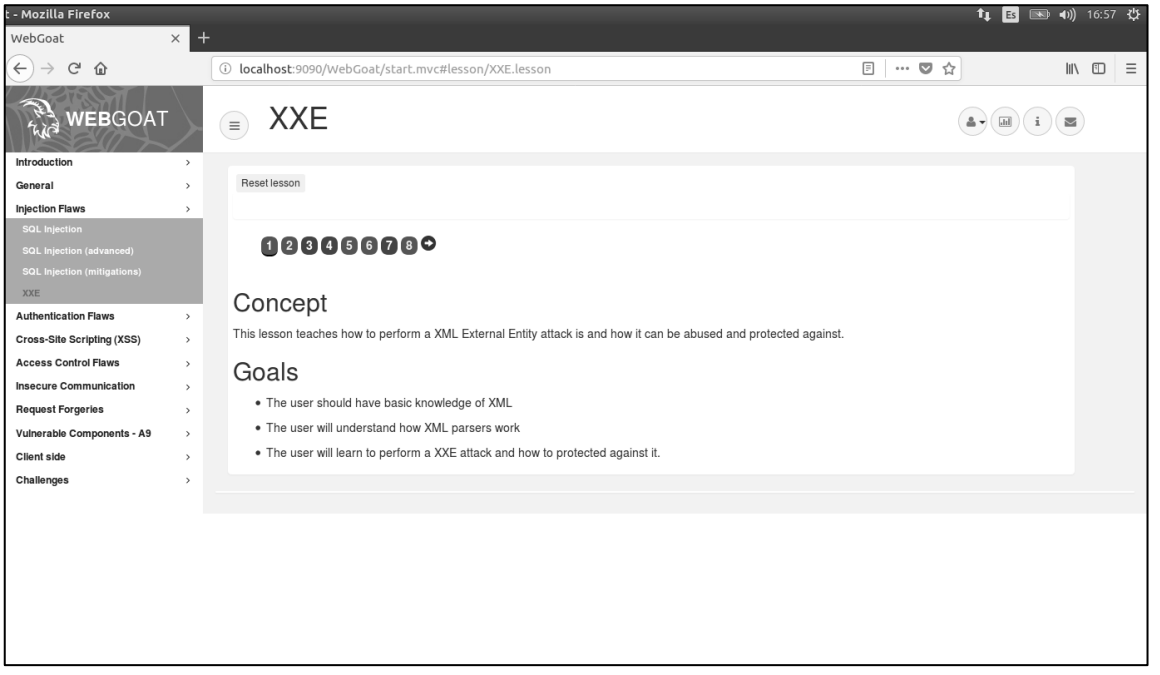
```
DTD Example:

<!ENTITY writer SYSTEM "https://www.w3schools.com/entities.dtd">
<!ENTITY copyright SYSTEM "https://www.w3schools.com/entities.dtd">

XML example:

<author>&writer;&copyright;</author>
```

Fuente: W3schools. https://www.w3schools.com/xml/xml_dtd_entities.asp. Consulta: 17 de mayo de 2018.

3.2.3. Funcionamiento del ataque

La aplicación WebGoat permite realizar pruebas de este ataque, de manera que se entienda el funcionamiento y así determinar cómo evitarlo. La lección que abarca los elementos básicos del ataque, se encuentra bajo el apartado *Injection Flaws* (defectos de inyección), el nombre de la lección es XXE. En la figura 64 se observa la lección en la aplicación WebGoat.

Figura 64. **Lección XXE, WebGoat**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 17 de mayo de 2018.

La lección XXE en WebGoat, cuenta con 8 partes, que se encuentran divididas de la siguiente manera:

- 6 partes de contenido.
- 2 partes de ejercicios.

Los objetivos a completar con la lección son los siguientes:

- Comprensión básica de XML y como se usa.
- Comprensión del funcionamiento de analizadores de XML.
- Conocimiento básico de cómo llevar a cabo un ataque XXE y cómo protegerse del mismo.

La lección abarca información de cómo estos ataques se llevan a cabo. El ataque se da al aprovechar la vulnerabilidad que genera el uso de una DTD mediante la declaración de entidades. Agregando funcionalidades al interprete XML que no están entre el comportamiento habitual, como leer un archivo local y exponer el contenido. Entre los daños que produce se mencionan los siguientes: la divulgación de datos confidenciales, denegación de servicios, falsificación de solicitudes, entre otros.

Es de considerar que el ataque se realiza hacia una aplicación que se encarga de procesar el documento XML, de manera que utilizando esta aplicación confiable se afecta otros sistemas internos, mediante la revelación de contenido interno por medio de solicitudes HTTP o lanzando un ataque CSRF contra cualquier servicio que no tenga protección. Además cuando existe una biblioteca para el procesamiento de XML, que es vulnerable a problemas de corrupción de memoria en el cliente, existe la posibilidad que se explote usando código arbitrario malicioso como que fuera de la aplicación.

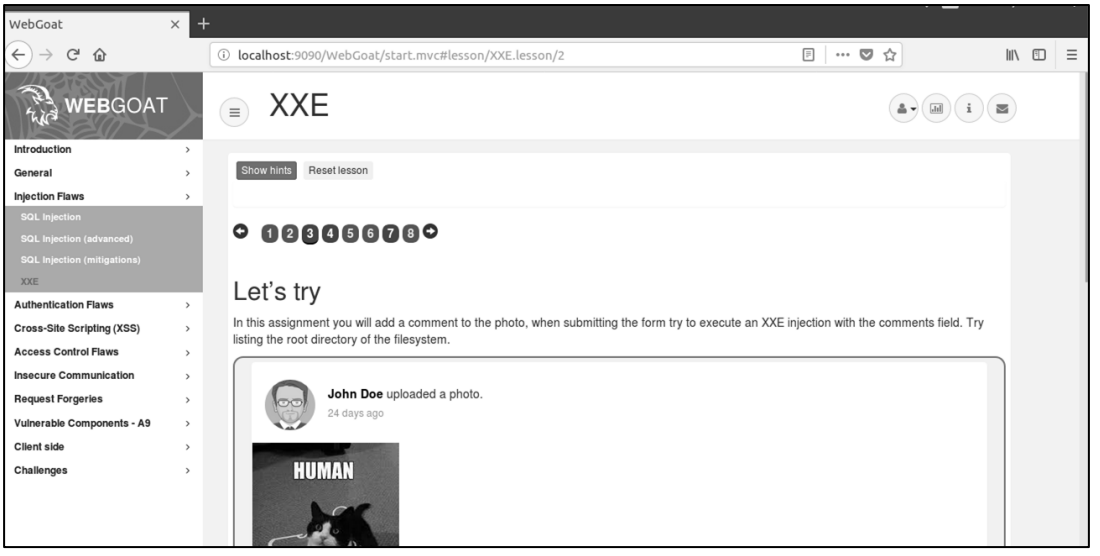
En general se distinguen 3 tipos de ataques XXE, los cuales se enumeran a continuación:

- Clásico: en esta clase se incluye una entidad externa a un DTD local.
- Ciego: no se muestran resultados o errores en la respuesta.
- Error: intentar obtener información del error en un mensaje de error.

3.2.4. Probando inyección XXE

La aplicación WebGoat presenta dos ejercicios prácticos para realizar un ataque XXE, en los cuales se busca atacar una aplicación que recibe información del cliente mediante el uso de un archivo XML. Para realizar los ejercicios se debe hacer uso de una declaración DTD, de manera que se obtenga información del servidor donde la aplicación se encuentra montada. En la figura 65 se muestra el ejercicio número uno de inyección XXE, directamente en la aplicación.

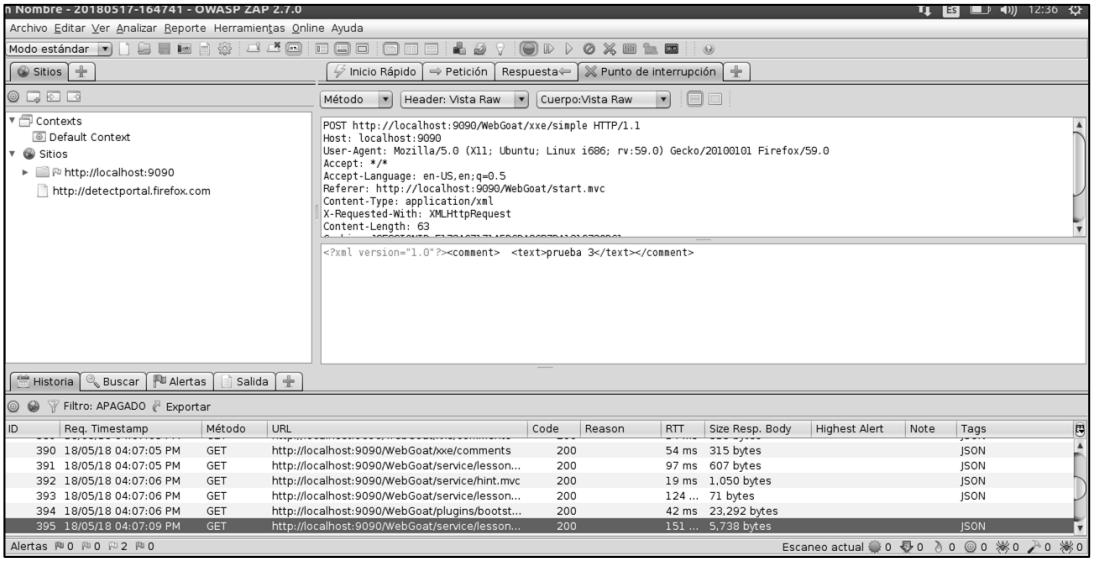
Figura 65. **Ejercicio 1 inyección XXE, WebGoat**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 19 de mayo de 2018.

En el ejercicio se utiliza un cuadro para enviar comentarios sobre la imagen que se ve en la figura 65. Para observar de qué manera se envía la información del cliente al servidor al guardar los comentarios, se intercepta la petición y así se realiza un análisis de la información que el servidor obtiene y como esta genera la posibilidad de ser manipulada para realizar un ataque XXE. En la figura 66 se muestra la solicitud que se genera al enviar un comentario, obtenida con ZAP.

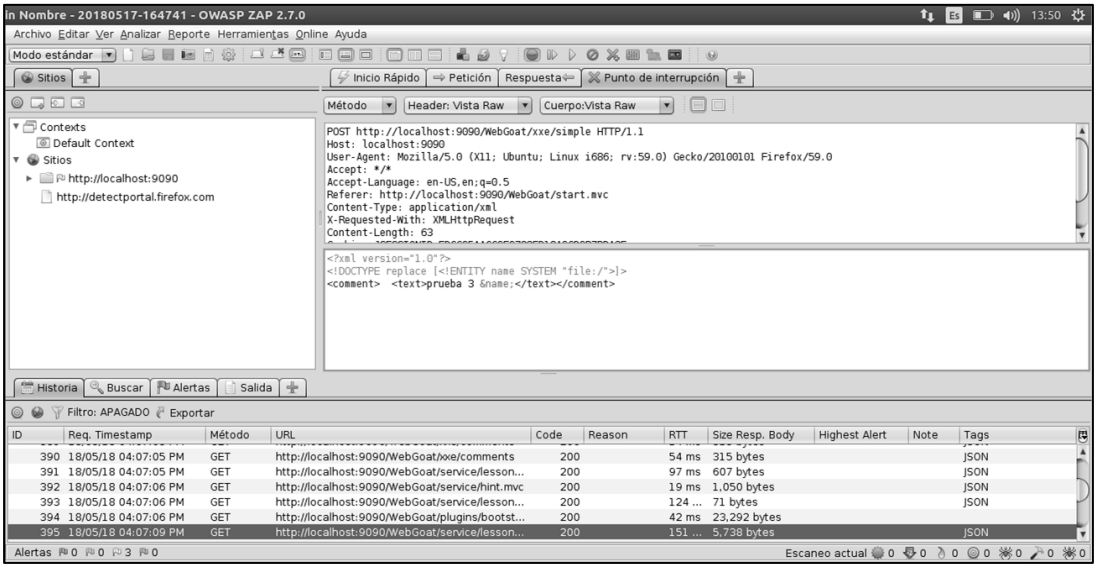
Figura 66. **Intercepción de petición, ejercicio 1 XXE**



Fuente: elaboración propia con base en aplicación OWASP ZAP. Consulta: 19 de mayo de 2018.

Como se observa en la figura 66 el cliente está enviando un XML hacia el servidor con los valores que el usuario ingresa. De manera que el objetivo del ejercicio en la aplicación WebGoat, es aprovechar este XML para listar lo que contiene el fichero principal donde se encuentra montada la aplicación. Lo que se realiza es una definición DTD, que al momento de ser analizada por el intérprete de XML, devuelva la información solicitada del servidor. En la figura 67 se muestra la solución para este ejercicio.

Figura 67. **Solución ejercicio 1 XXE**



Fuente: elaboración propia con base en aplicación OWASP ZAP. Consulta: 19 de mayo de 2018.

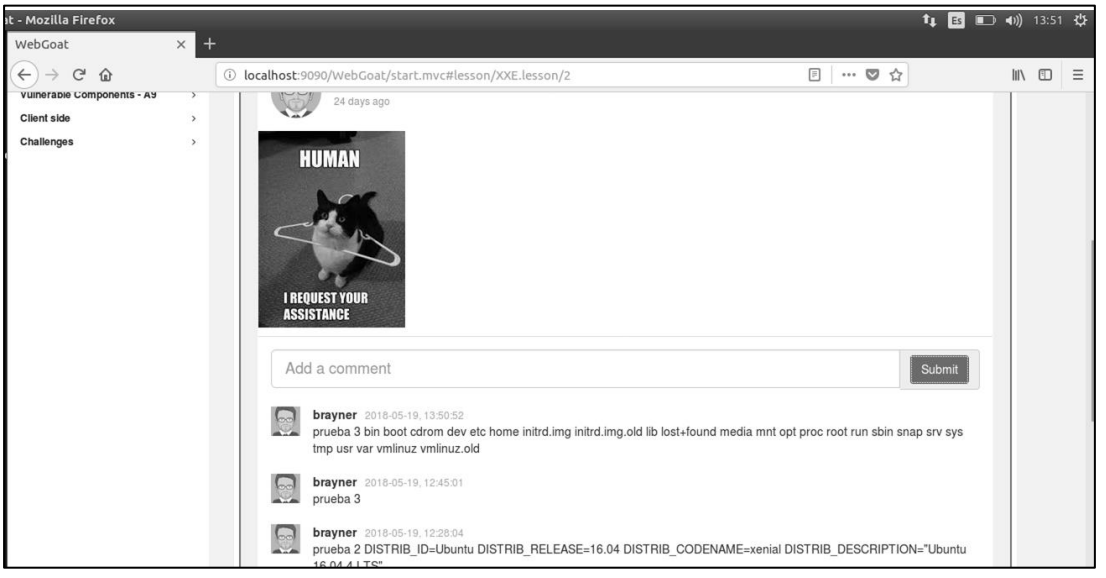
Como se observa en la figura 67, se utiliza una entidad DTD, para obtener la información de lo que contiene el directorio raíz. Esto se realiza mediante la declaración de la entidad *name* que es del tipo *file*. A continuación se muestra la declaración como tal:

<!DOCTYPE replace [<!ENTITY name SYSTEM "file:/">]>

Una vez es declarada solo se necesita llamar en el contenido del XML para obtener la información que contiene la entidad que en este caso será la lista de archivos y carpetas que tiene el directorio raíz del servidor donde se encuentra montada la aplicación. Para expandir la entidad se realiza de la siguiente manera: "&*name*;".

Una vez realizada la modificación en la solicitud que realiza el cliente al servidor, se continúa con el envío. En la figura 68 se muestra la respuesta obtenida del servidor, con la información requerida por el ejercicio.

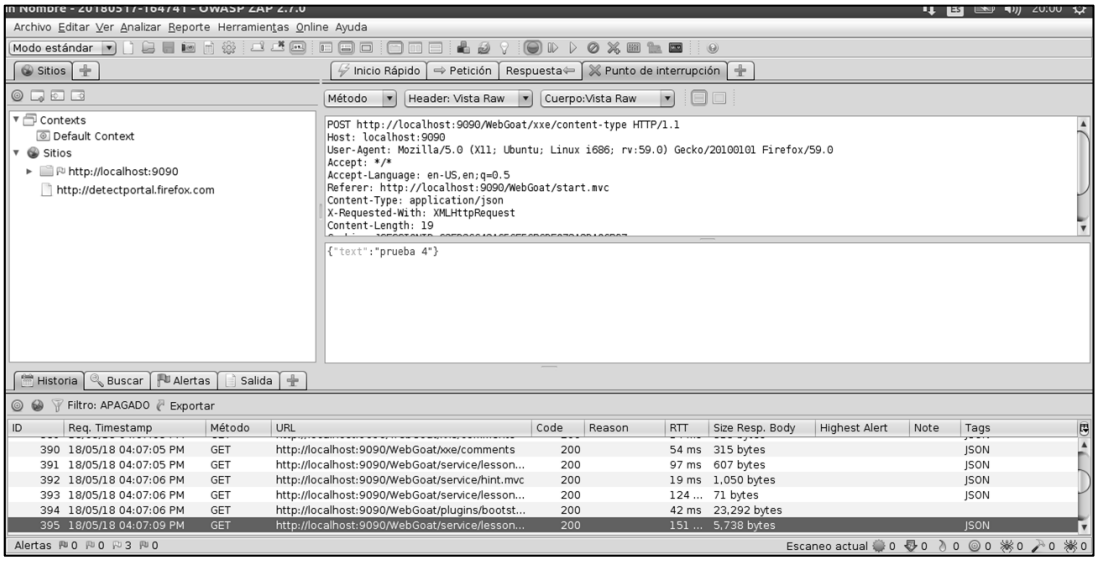
Figura 68. **Respuesta del servidor, ejercicio 1 XXE**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 19 de mayo de 2018.

El ejercicio 2 de inyección XXE tiene el mismo objetivo que el número uno, la diferencia que existe es que la solicitud se realiza usando un servicio *Rest*, de manera que no se tiene un documento XML con la información del cliente, sino en cambio un documento JSON, que el servidor espera recibir. La solicitud interceptada se muestra en la figura 69, usando la aplicación OWASP ZAP.

Figura 69. **Intercepción de petición, ejercicio 2 XXE**

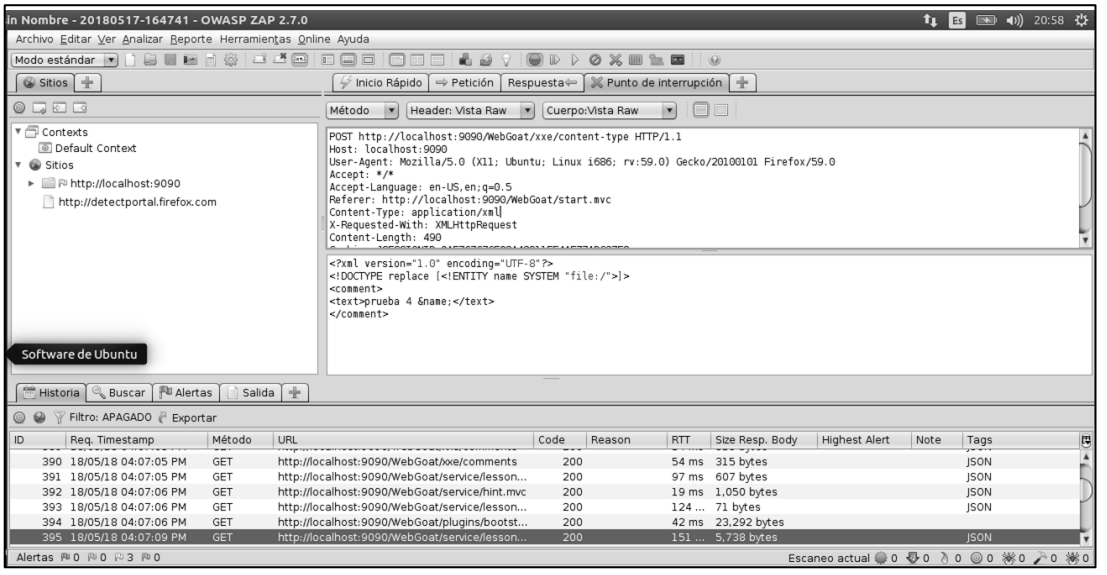


Fuente: elaboración propia con base en aplicación OWASP ZAP. Consulta: 19 de mayo de 2018.

Para completar este ejercicio es necesario cambiar el tipo del contenido que la petición lleva de JSON a XML, de manera que en el nuevo cuerpo de la petición se incluya declaraciones de entidades para obtener la información requerida. El primer paso para lograrlo es cambiar el atributo de la petición *Content-type* a *application-xml*. Luego se realiza el cambio del contenido de la petición transformando el documento JSON a un documento XML válido. Por último se cambia el valor del atributo *conten-length* con el tamaño del nuevo cuerpo de la petición.

En la figura 70 se muestra un ejemplo de cómo realizar la solicitud realizando un ataque XXE, realizando los cambios necesarios para que se envíe un documento XML con una definición DTD, que tiene como objetivo listar los archivos del directorio *root* del servidor donde se encuentra la aplicación WebGoat montada, de manera que se complete el ejercicio.

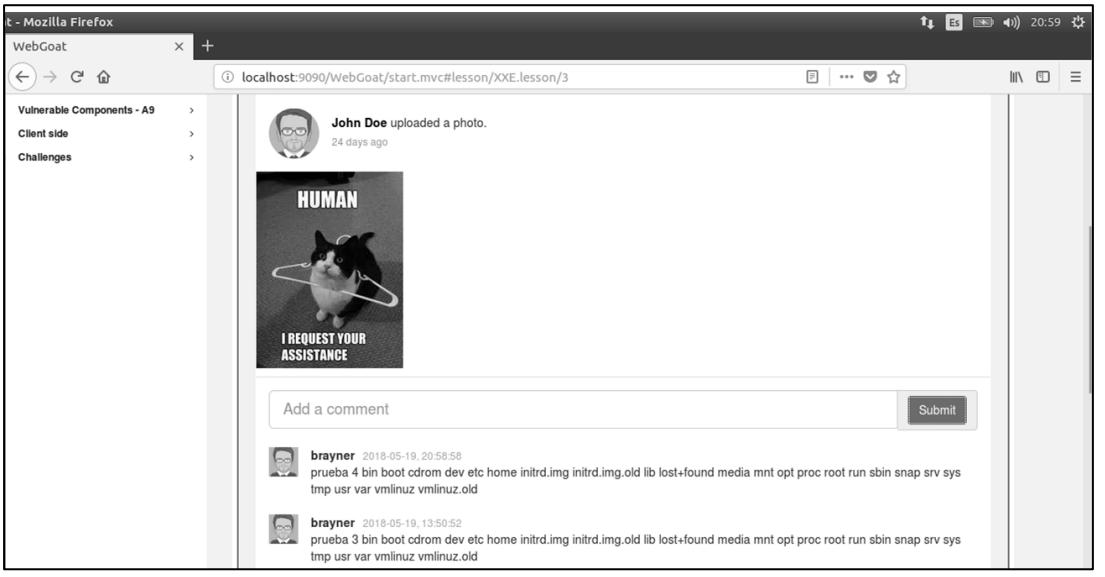
Figura 70. **Solución ejercicio 2 XXE**



Fuente: elaboración propia con base en aplicación OWASP ZAP. Consulta: 19 de mayo de 2018.

Una vez enviada la petición el resultado en la página del ejercicio de la aplicación WebGoat. Como se observa en la figura 71.

Figura 71. **Respuesta del servidor, ejercicio 2 XXE**



Fuente: elaboración propia con base en aplicación WebGoat. Contenido: 19 de mayo de 2018.

3.2.5. Inyección ciega de XXE

En algunos casos los resultados de un ataque XXE no se muestran dentro de la respuesta o resultado que brinda la aplicación, de manera que se complica conocer si la aplicación es vulnerable o no a un ataque de inyección XXE. Otra situación que se presenta de manera similar es cuando el archivo que se está intentando leer contiene algún carácter ilegal XML que hace que el analizador falle.

Para determinar si existe la posibilidad de realizar un ataque de inyección XXE en una aplicación de la cual no se obtiene una respuesta visible, se realiza mediante un ataque ciego a la misma. El ataque ciego se basa en el uso de un servidor externo para guardar un archivo que contenga la definición DTD del ataque que se quiere realizar, para posteriormente ser llamado en la solicitud que se realiza al servidor y así obtener respuesta con la información solicitada al servidor. En la figura 72 se muestra un ejemplo de cómo se realiza un ataque ciego XXE.

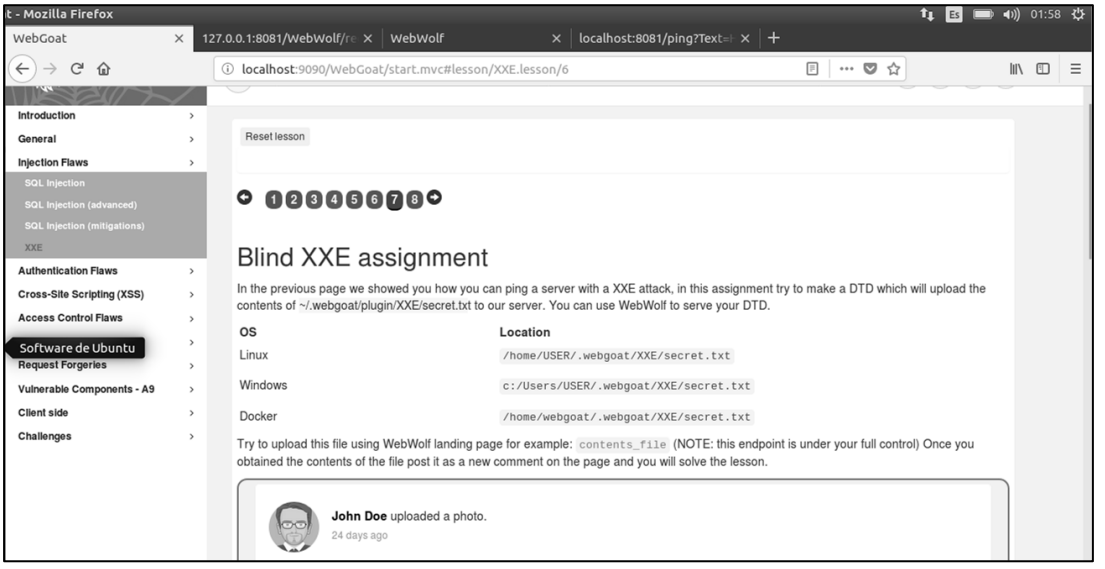
Figura 72. **Ejemplo de inyección ciega XXE**

Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 20 de mayo de 2018.

3.2.6. Probando inyección ciega XXE

La aplicación WebGoat posee un ejercicio para aplicar la inyección ciega XXE, aprovechando la vulnerabilidad que esta posee. El objetivo del ejercicio es obtener la información del siguiente archivo: ~/.webgoat/plugin/XXE/secret.txt. Que se encuentra alojado en la aplicación. Esto se debe realizar usando un archivo DTD alojado en la aplicación complementaria WebWolf. En la figura 73 se muestra el ejercicio de la aplicación WebGoat.

Figura 73. **Ejercicio de inyección ciega XXE, WebGoat**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 20 de mayo de 2018.

3.2.7. Mitigación inyección XXE

Debido a que la inyección XXE se basa en agregar información a un documento XML, que permita obtener datos útiles de la aplicación o el servidor donde se encuentra montada. La mejor manera de protegerse, es validando la entrada recibida de un cliente, en algunos casos se opta por obviar dentro de la aplicación cualquier definición DTD o limitarse a excluir solamente la declaración de entidades, que es donde la vulnerabilidad se presenta.

La forma de deshabilitar las definiciones DTD, varía según el analizador que se utilice. Pero se establece la forma de deshabilitar las definiciones que más se adapta a la mayoría de ellos, se muestra en la figura 74. Esto ayuda a ser más segura la aplicación, debido a que también se prevén de ataques DOS y así evitar que la aplicación sufra una saturación de solicitudes y deje de brindar los servicios que ofrece.

Figura 74. **Deshabilitar definiciones DTD**

```
factory.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
```

Fuente: OWASP.
https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet.
Consulta: 20 de mayo de 2018.

Si se desea consultar el detalle de cómo deshabilitar definiciones DTD o declaraciones DTD en lenguajes de programación específicos, se encuentra más información en la página oficial de prevención de ataques XXE del proyecto OWASP[13]. En caso de ser necesario utilizar definiciones para documentos XML, se recomienda usar XSD, debido a que esta no es vulnerable a ataques XXE y por lo tanto hará de la aplicación y el analizador XML elementos seguros.

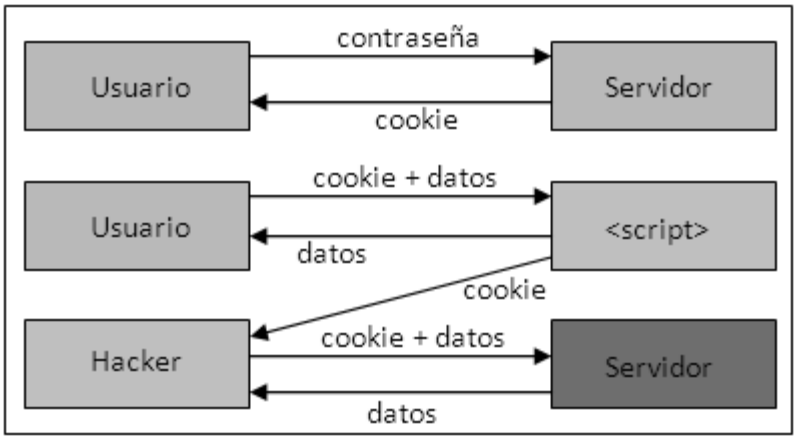
[13] Hoja de prevención de ataques XXE OWASP: https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Prevention_Cheat_Sheet. Consulta: 21 de mayo de 2018

3.3. XSS

Vulnerabilidad que se presenta en aplicaciones web obtiene el nombre de las siglas en inglés *cross-site scripting* (ejecución de comandos en sitios cruzados). Esta debilidad aparece por la falta de validaciones en los campos de entrada que posee una aplicación, además de la evasión de sanear los resultados de salida para la representación como página web. Esto permite que se envíen datos inseguros e inclusive la ejecución de scripts completos en el cliente que utiliza la aplicación.

El ataque a esta vulnerabilidad se realiza mediante el envío de código malicioso a la aplicación web en el cliente, utilizando hipervínculos, que tienen como propósito conducir al usuario a otro sitio, con el fin de obtener información sensible del usuario, sesiones de usuario y comprometer el navegador y la integridad del sistema. Este ataque también es usado para realizar una denegación de servicio (DDos) a un sistema web. En la figura 75 se ejemplifica el flujo que sigue un ataque al aprovechar esta vulnerabilidad.

Figura 75. **Flujo de ataque XSS**



Fuente: UNAM. https://www.seguridad.unam.mx/historico/documento/index.html-id=35. Consulta: 22 de mayo de 2018.

En la figura 75 se muestra como para realizar un ataque XSS, el atacante utiliza la confianza que tiene el usuario en la aplicación para insertar *scripts* que ayudan a obtener datos tanto del servidor como del cliente. Por lo regular el atacante busca que los hipervínculos usados no parezcan sospechosos, e inclusive en ocasiones estos suelen redirigir a páginas similares a la original, para realizar de forma efectiva el robo de información, mediante el acceso a los recursos que cuenta el navegador web y se asocian a la aplicación.

3.3.1. Tipos de ataques XSS

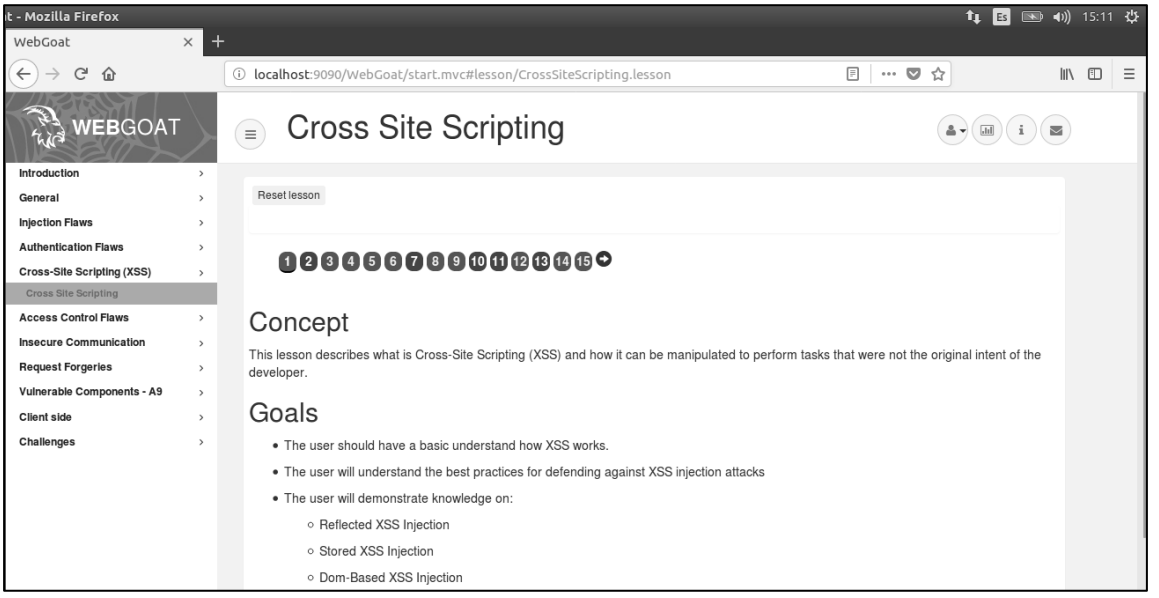
A continuación se listan los diferentes tipos de ataque XSS que existe la posibilidad de encontrar en aplicaciones web:

- Directo: este ataque se base en la inclusión de etiquetas <*script*> y <*frame*> para invadir el código HTML de un sitio. También es conocido como persistente.
- Local: es una variante del ataque directo XSS, consiste en explotar las vulnerabilidades del mismo código que posee una página web. Estas vulnerabilidades se generan por el uso incorrecto del DOM, usado para establecer objetos estandarizados en una página web, en conjunto con JavaScript, con el objetivo de abrir otra página web usando código Java incrustado de manera maliciosa, afectando la página del sistema local. Usando este ataque no se envía ningún código directamente al servidor, debido a que el funcionamiento se da específicamente en el cliente, pero hace parecer que es el servidor quien envía este código malicioso. De manera que aunque el servidor cuente con protección esta no funciona.
- Indirecto: se realiza mediante la modificación de los valores enviados de una página web a otra sin el uso de una sesión, de manera que se envía un mensaje o una ruta en la URL del navegador, en una *cookie* o utilizando cualquier otra cabecera HTTP. También es conocido como reflejado.

3.3.2. Funcionamiento del ataque

La aplicación WebGoat, permite realizar pruebas de este ataque de manera que se entienda el funcionamiento y así determinar cómo evitarlo. La lección que abarca los elementos básicos del ataque, se encuentra bajo el apartado *Cross-Site Scripting* (ejecución de comandos en sitios cruzados), el nombre de la lección es *Cross Site Scripting*. En la imagen 76 se observa la lección en la aplicación WebGoat.

Figura 76. **Lección *Cross Site Scripting*, WebGoat**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 26 de mayo de 2018.

La lección *Cross Site Scripting* en WebGoat, cuenta con 15 partes, que se encuentran divididas de la siguiente manera:

- 10 partes de contenido.
- 5 partes de ejercicios.

Los objetivos a completar con la lección son los siguientes:

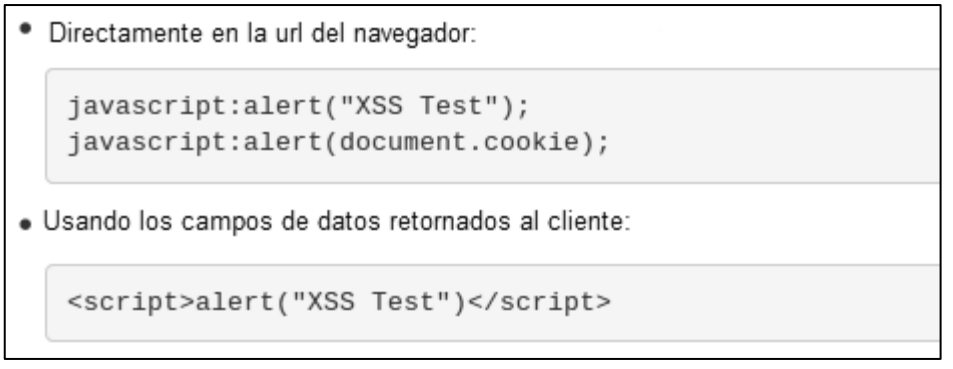
- Comprensión básica de cómo funciona XSS.
- Comprensión de las mejores prácticas para defenderse contra XSS.
- Conocimiento básico de los siguientes tipos de ataque XSS:
  - Inyección XSS reflejada.
  - Inyección XSS almacenada.

- Inyección XSS directa o basada en DOM.

La lección abarca información de cómo estos ataques se llevan a cabo. El ataque se da al aprovechar la vulnerabilidad que genera el uso de etiquetas HTML y de *script* como entradas para información que el cliente debe enviar. En donde la información que se envía no se codifica, ni se realiza saneamiento sobre esta para evitar la aparición de código malicioso de un ataque XSS. Según OWASP es uno de los ataques más frecuentes debido a que existe la posibilidad de causar mucho daño.

Existe una defensa conocida para este tipo de ataque, de la cual se habla en apartados posteriores de este mismo capítulo, a pesar de esto aún existen muchas instancias de este ataque en la web. Esto sucede ya que cada vez es más común el enriquecimiento de los sitios web mediante el uso de llamadas a funciones con JavaScript, que crean la posibilidad de ser comprometidas. Por lo tanto si no se protegen de manera adecuada, los datos confidenciales de un usuario serán robados con fines perjudiciales. En la figura 77 se muestran ejemplos básicos de la inyección XSS.

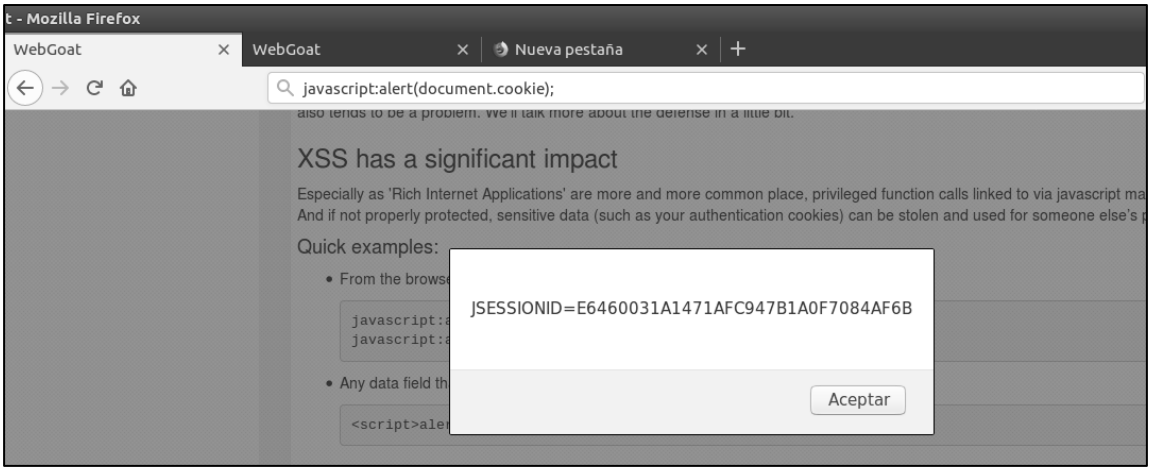
Figura 77. **Ejemplos básicos, XSS**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 26 de mayo de 2018.

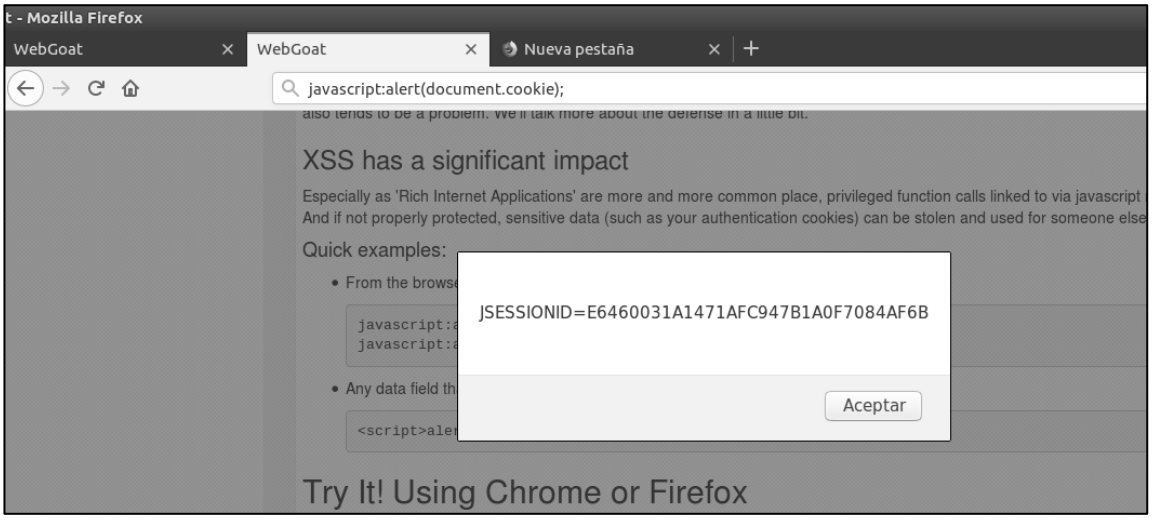
La aplicación WebGoat permite realizar pruebas de este tipo de inyección en el apartado 2 de la lección *Cross Site Scripting*. Las instrucciones son básicas: usar el siguiente comando javascript:alert(document.cookie); en una nueva pestaña utilizando la misma URL de la lección y en la pestaña actual de la lección. Con el fin de determinar si las *cookies* de ambas pestañas son iguales o tienen algún cambio entre sí. En la figura 78 se observa la prueba realizada en la pestaña actual de la aplicación y en la figura 79 se observa la prueba realizada en la segunda pestaña.

Figura 78. **Primera prueba XSS simple**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 27 de mayo de 2018.

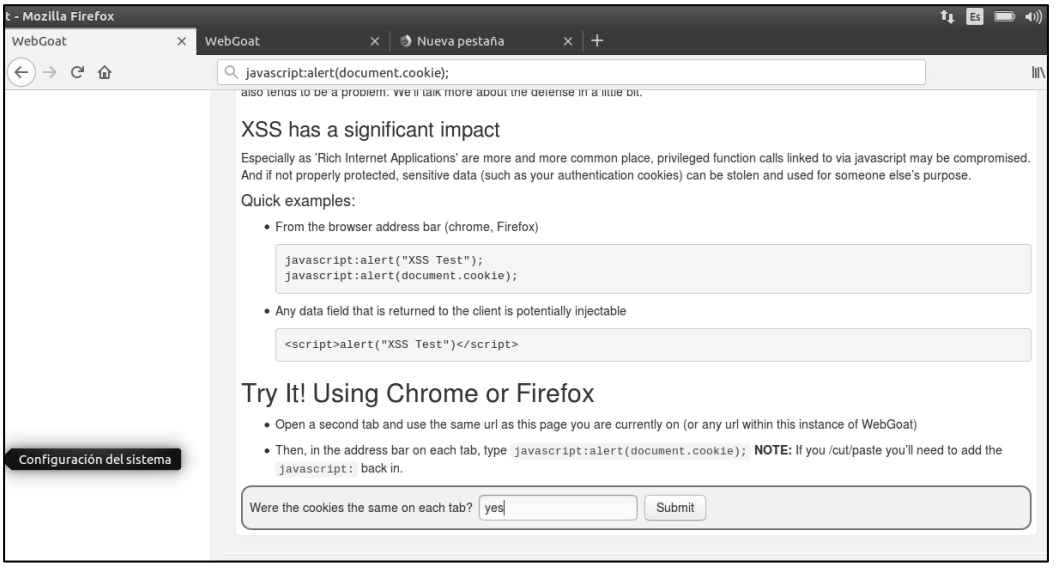
Figura 79. **Segunda prueba XSS simple**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 27 de mayo de 2018.

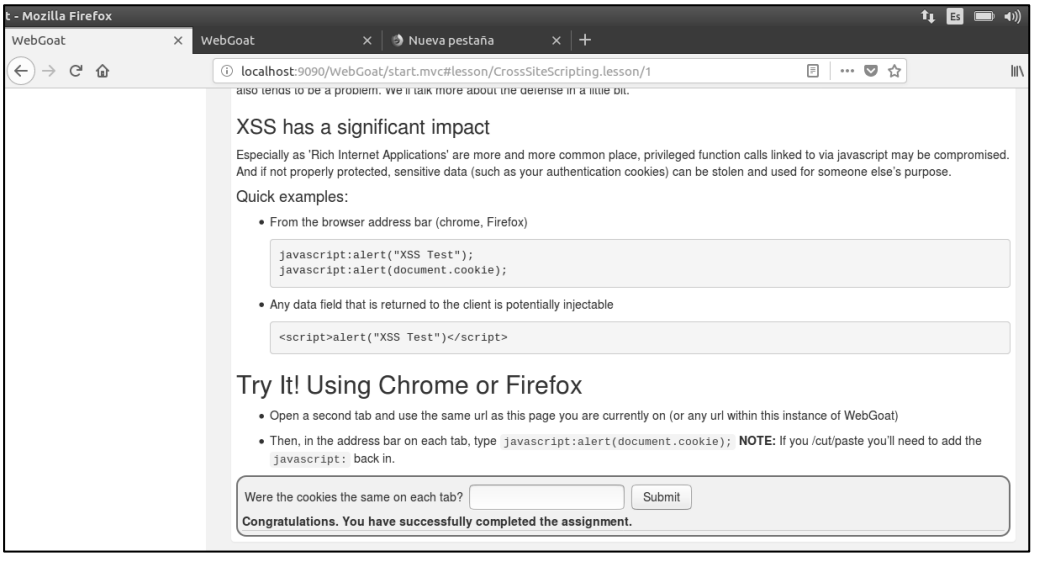
Como se observa en la figura 78 y en la figura 79 el resultado obtenido en ambas pestañas, el contenido que posee la cookie del usuario es el mismo. Por lo tanto se responde que el resultado obtenido en el ejercicio es sí y de esta manera se termina el ejercicio de forma exitosa. En la figura 80 se muestra el ingreso del valor en inglés y en la figura 81 se muestra el resultado obtenido de la aplicación WebGoat.

Figura 80. **Solución ejercicio simple XXS**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 27 de mayo de 2018.

Figura 81. **Resultado de ejercicio simple XXS**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 27 de mayo de 2018.

3.3.2.1. Locaciones más comunes

A continuación se listan los lugares más comunes donde se encuentra la vulnerabilidad XSS:

- Campos de búsqueda que devuelven una cadena de búsqueda al usuario.
- Campos de entrada que devuelven información del usuario.
- Mensajes de error que devuelven datos proporcionadas por el usuario.
- Campos ocultos que poseen valores proporcionados por el usuario.
- Cualquier página que muestre datos proporcionados por el usuario.
- Encabezados HTTP.

3.3.2.2. Riesgos del ataque

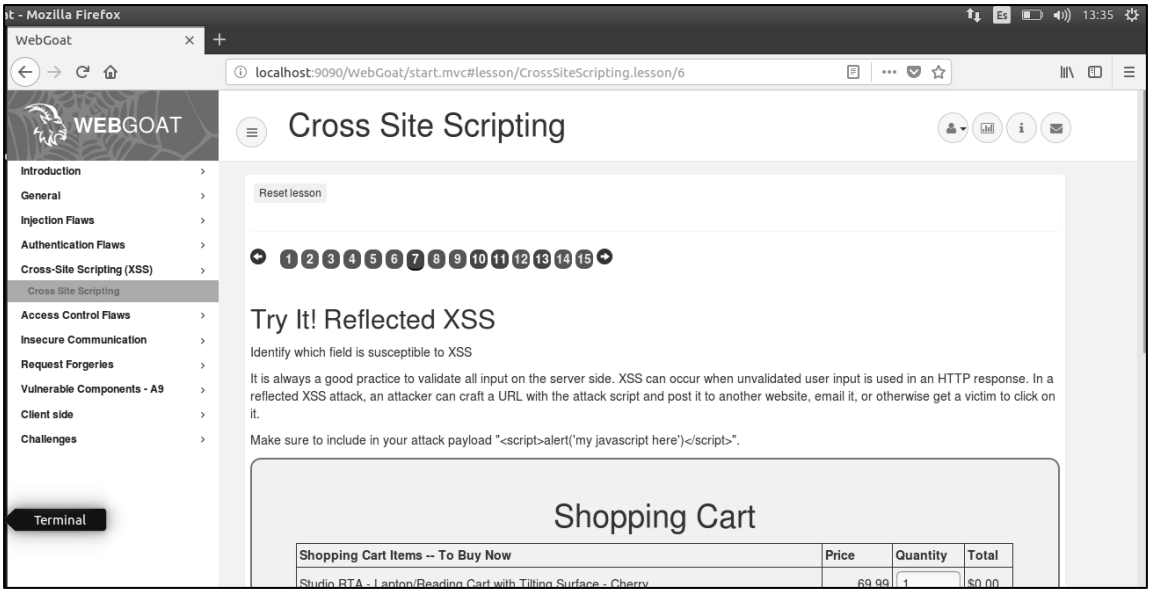
A continuación se listan los riesgos posibles que se generan al momento de una aplicación sufra un ataque XSS:

- Robo de *cookies* de sesión.
- Creación de solicitudes falsas.
- Creación de campos falsos en una página para recolección de credenciales.
- Redirección a una página no confiable.
- Creación de solicitudes enmascaradas como de un usuario válido.
- Robo de información confidencial.
- Ejecución de código malicioso en un sistema de usuario final.
- Inserción de contenido hostil e inapropiado.
- Agregar validez a los ataques de *pishing*, al usar un dominio válido.

3.3.3. Probando inyección XSS reflejada

La lección *Cross Site Scripting* de WebGoat posee un ejercicio para realizar pruebas de inyección XSS reflejada, que se encuentra en el apartado 7. El ejercicio consiste en encontrar un campo de la página Web que sea vulnerable a la inyección XSS, de manera que se agregue un valor ejecutable de JavaScript a la página Web. En la figura 82 se muestra la página del ejercicio de inyección XSS reflejada.

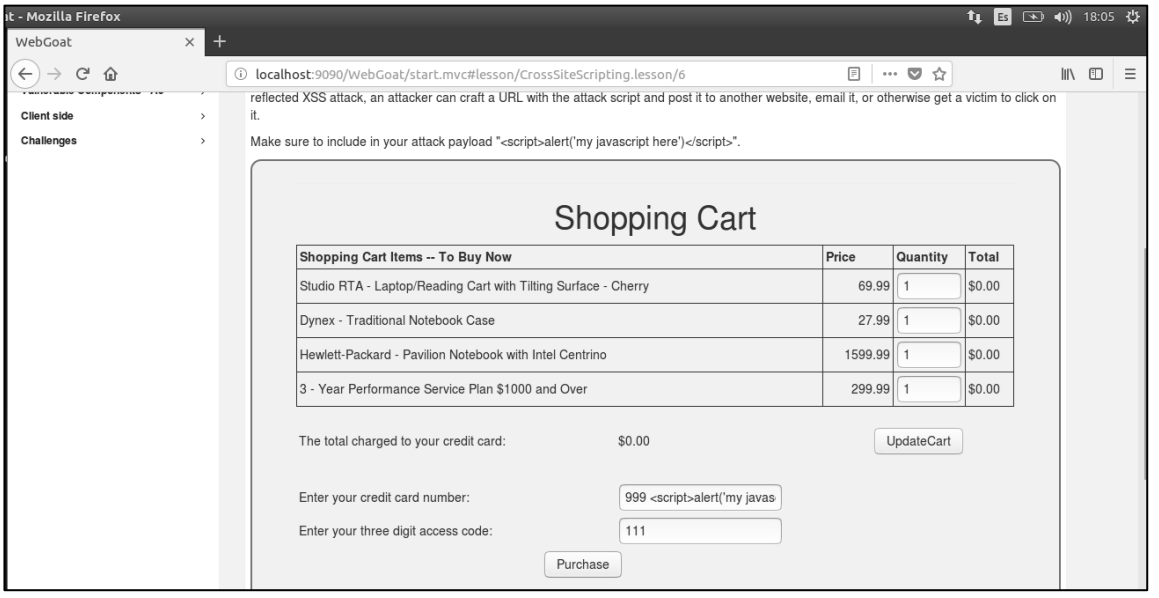
Figura 82. **Ejercicio de inyección XSS reflejada**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 27 de mayo de 2018.

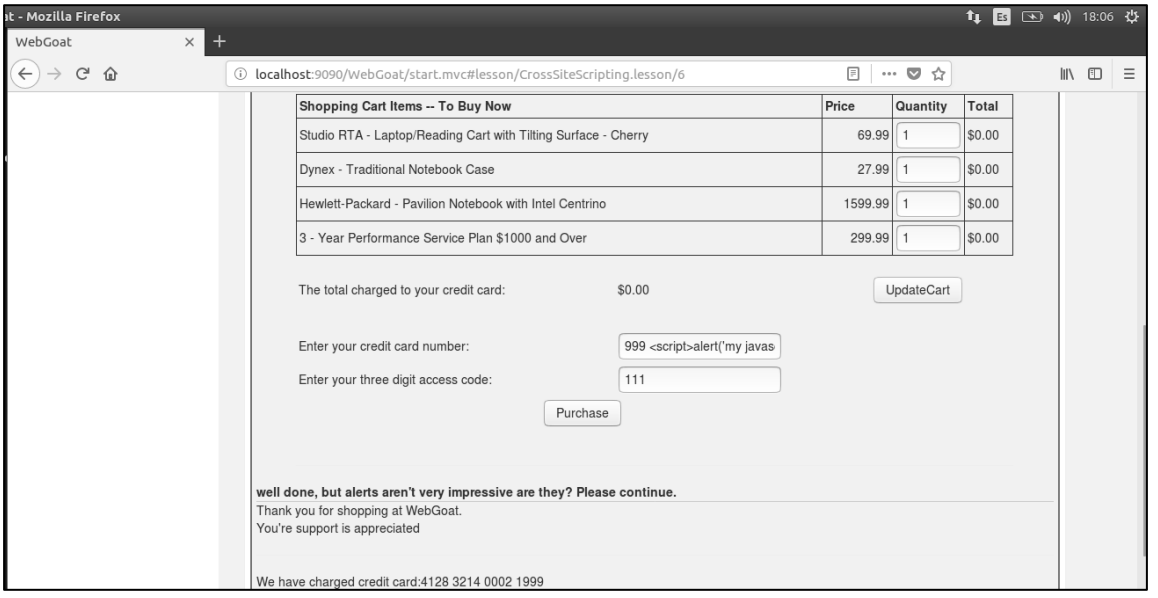
En la figura 83 se muestra la solución del ejercicio de inyección XSS reflejada, en el cual se utiliza el campo proporcionado para la tarjeta que realizará el pago, campo cuyo valor es tomado del formulario que envía el cliente y mostrado posteriormente como respuesta, sin ningún tipo de validación. En la figura 84 se muestra el mensaje de la aplicación indicando que el ejercicio se ha completado satisfactoriamente.

Figura 83. **Solución de ejercicio de inyección XSS reflejada**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 27 de mayo de 2018.

Figura 84. **Respuesta de ejercicio de inyección XSS reflejada**



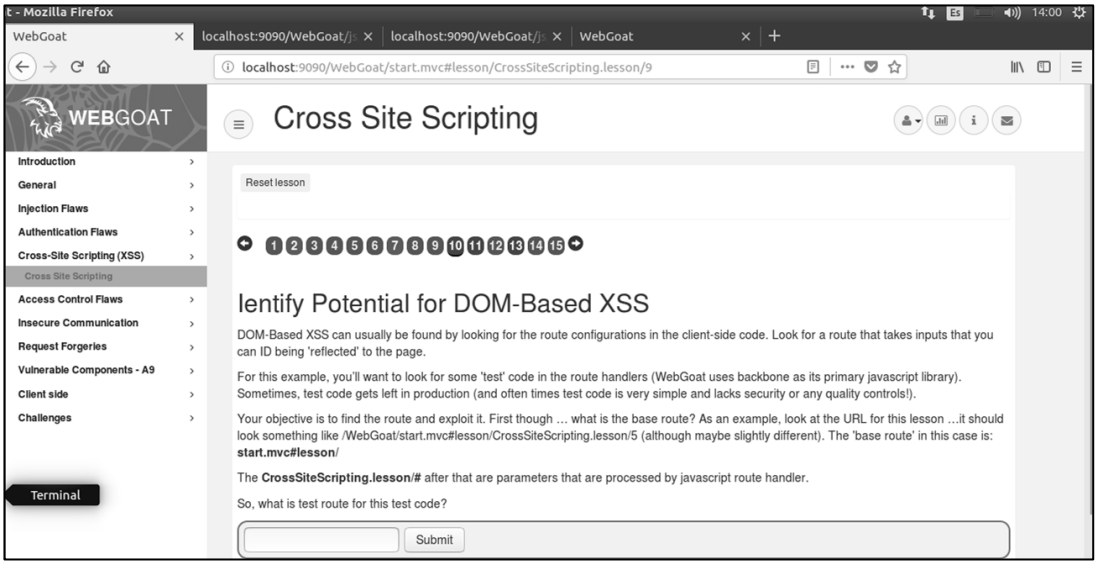
Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 27 de mayo de 2018.

3.3.4. Identificando inyección XSS basa en DOM

En el apartado 10 de la lección *Cross-Site Scripting* de la aplicación WebGoat, se presenta un ejercicio práctico que permite identificar la inyección XSS utilizando el DOM de una página web. En este ejercicio se solicita examinar el contenido de la página web para determinar cómo esta interpreta las distintas funciones que ofrece, mediante el uso de JavaScript. El objetivo a alcanzar es determinar la ruta que la aplicación usa para realizar pruebas de ejecución de funciones JavaScript. Bajo la premisa que en muchas ocasiones se realizan puestas en producción de código que ha servido para realizar pruebas durante el desarrollo, pero que no se tomó la precaución de borrarse.

En la figura 85 se muestra el ejercicio directamente de la aplicación WebGoat. El cual presenta una página web simple con un cuadro de texto que permitirá ingresar la respuesta que solicita la lección, acerca de la ruta que la misma aplicación posee, para realizar pruebas. De esta manera se aprovecha esta ruta para realizar un ataque XSS en la aplicación en el siguiente ejercicio.

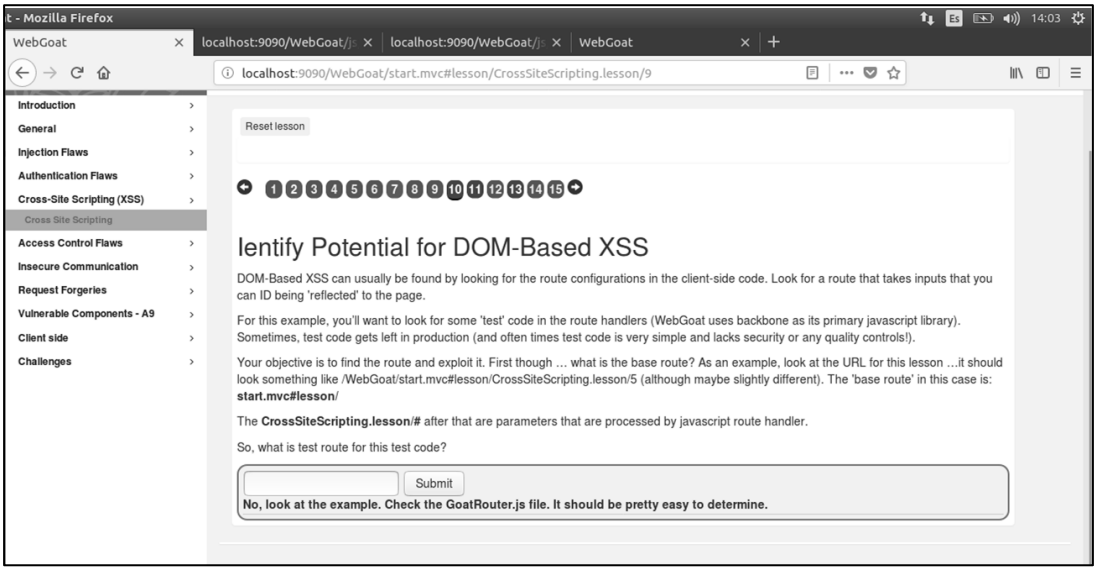
Figura 85. **Identificando inyección XSS basada en DOM**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 28 de mayo de 2018.

Para llevar a cabo el ejercicio se debe examinar cómo está construida la página de la lección actual. En este caso existe la posibilidad de ver el código fuente de la página o inspeccionar elementos. Para efectos del ejercicio se realiza *click* en el botón *submit* que aparece en la página para realizar un estudio de cómo se comporta. En la figura 86 se muestra el resultado obtenido al realizar esta acción y la respuesta brindada por la aplicación.

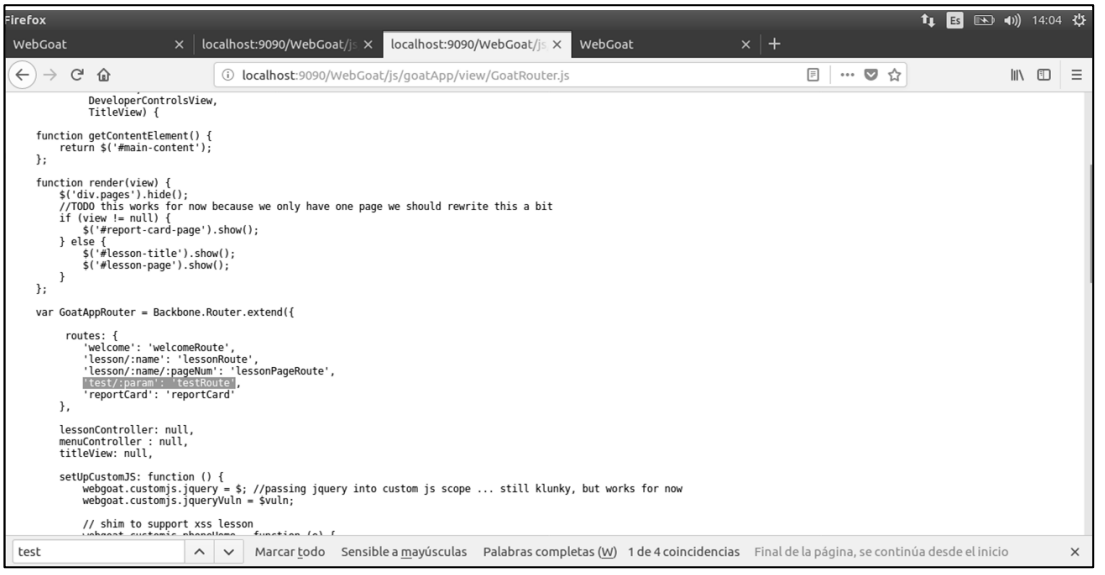
Figura 86. **Probando formulario, inyección XSS basada en DOM**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 28 de mayo de 2018.

Al observar la repuesta obtenida en la figura 86 indica que visualizando el archivo GoatRouter.js se encontrará la solución del ejercicio. De manera que se inspecciona la página para determinar la ruta de este archivo y así posteriormente acceder a este para visualizar el contenido, tal y como se observa en la figura 87.

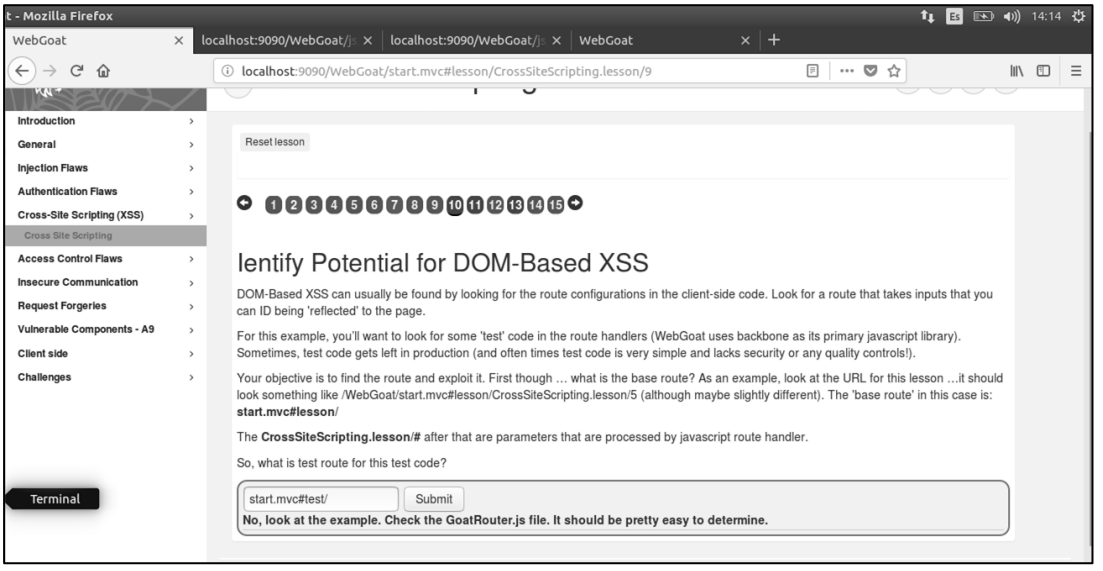
Figura 87. **Archivo GoatRouter.js**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 28 de mayo de 2018.

En la figura 87 se marca el valor buscado del archivo GoatRouter.js, que es una dirección que la aplicación utiliza para realizar pruebas del enrutamiento mediante el uso de funciones JavaScript. De manera que con este valor obtenido se da solución al ejercicio presentado. Tomando en cuenta la dirección base de la aplicación. La solución a este ejercicio se muestra en la figura 88.

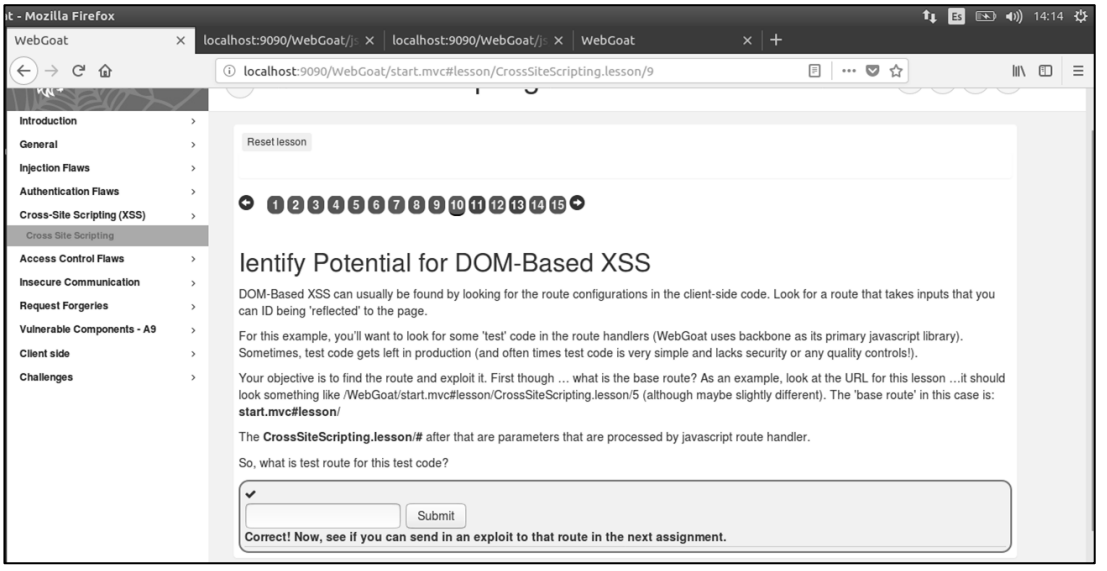
Figura 88. **Solución identificación de inyección XSS basado en DOM**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 28 de mayo de 2018.

Una vez ingresada la repuesta esperada, la aplicación indica que se ha resuelto el ejercicio de manera correcta y la respuesta obtenida se utilizará en un ejercicio posterior. Como se muestra en la figura 89.

Figura 89. **Identificación finalizada inyección XSS basada en DOM**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 28 de mayo de 2018.

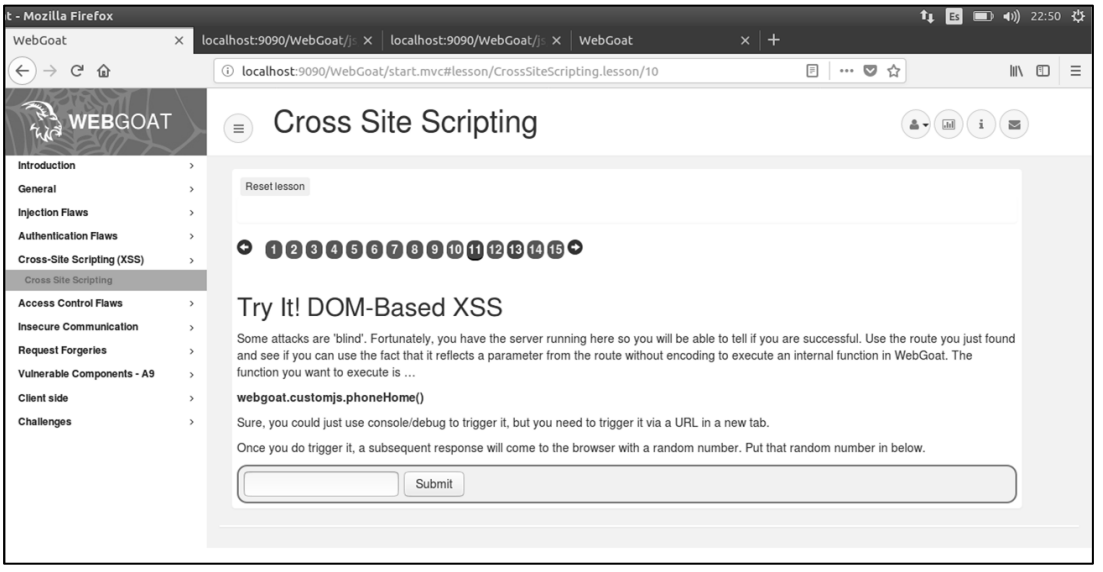
3.3.5. Probando inyección XSS basada en DOM

En el apartado 11 de la lección *Cross-Site Scripting* de la aplicación WebGoat, se presenta un ejercicio práctico que permite realizar pruebas de inyección XSS utilizando el DOM de una página web. En este ejercicio se solicita el uso de la dirección encontrada en el ejercicio anterior de identificación de inyección XSS, aprovechando que esta envía un parámetro sin codificación, de forma que se ejecute una función interna de la aplicación WebGoat. La función a ejecutar es la siguiente:

webgoat.customjs.phoneHome ()

La función *phoneHome*() debe ser consumida usando una nueva ventana, sin usar la consola para ejecutar la función, de manera que cuando se consuma ejecute una petición post en la cual la aplicación responderá un número aleatorio, que posteriormente debe ingresarse en el campo proporcionado en la aplicación con este fin, para finalizarlo exitosamente. En la figura 90 se muestra el ejercicio dentro de la aplicación WebGoat.

Figura 90. **Ejercicio de inyección XSS basada en DOM**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 29 de mayo de 2018.

Para solucionar el ejercicio se utiliza la dirección de prueba, descubierta en el ejercicio anterior de identificación de inyección XSS basado en DOM. El valor a utilizar como parámetro en la función es el siguiente:

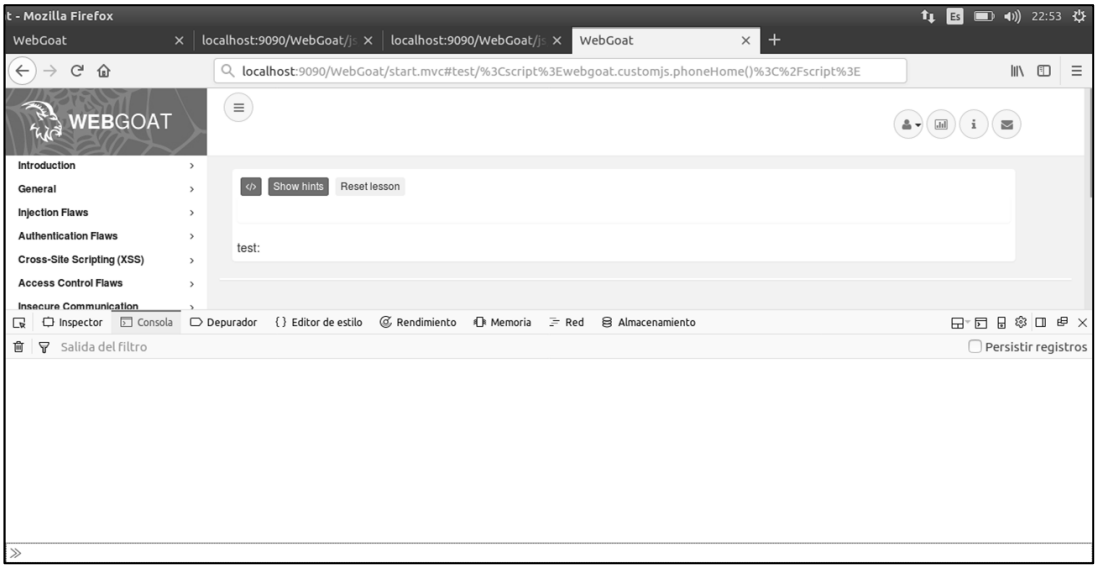
```
<script>webgoat.customjs.phoneHome()</script>
```

En este caso se debe convertir el valor presentado a código por ciento, de manera que sea interpretado como parte de los valores enviados por medio de la URL y luego transcrito en el archivo de enrutamiento js. El valor queda de la siguiente forma:

%3Cscript%3Ewebgoat.customjs.phoneHome()%3C%2Fscript%3E

En la figura 91 se encuentra la solución final usando una nueva pestaña en el navegador y la ruta base definida anteriormente.

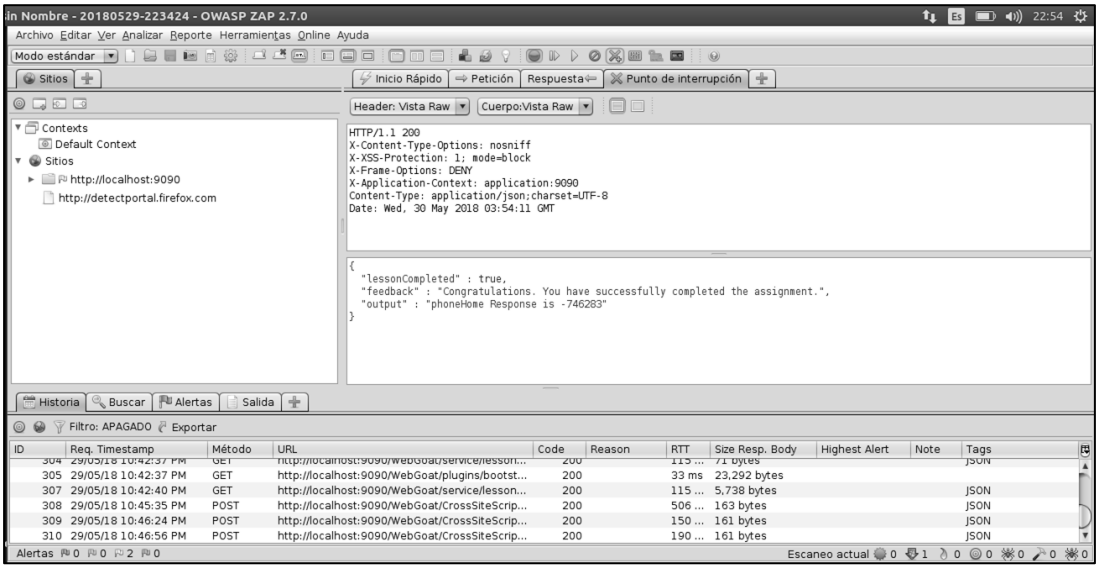
Figura 91. **Solución ejercicio de inyección XSS basado en DOM**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 29 de mayo de 2018.

Posterior al ejecutarse la función, se debe capturar la respuesta obtenida del servidor, en este caso se usa la aplicación OWASP ZAP, con este fin. En la figura 92 se muestra la respuesta obtenida.

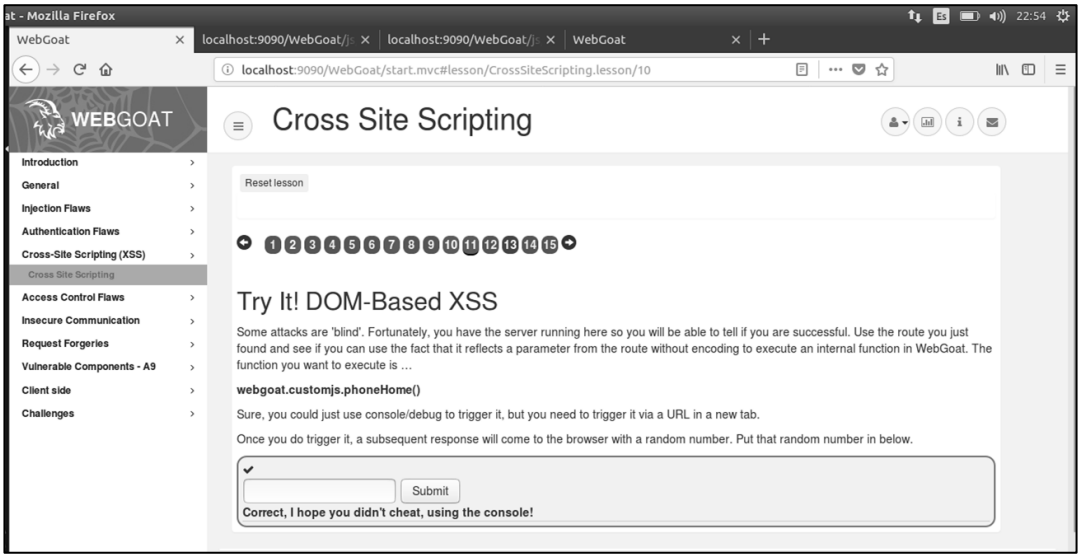
Figura 92. **Respuesta de función *phoneHome***



Fuente: elaboración propia con base en aplicación OWASP ZAP. Consulta: 29 de mayo de 2018.

Una vez ingresado el valor obtenido de la ejecución de la función *phoneHome*, en el campo de texto proporcionado por la aplicación se completa el ejercicio de manera exitosa, como se muestra en la figura 93.

Figura 93. **Ejercicio finalizado de inyección XSS basada en DOM**



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 29 de mayo de 2018.

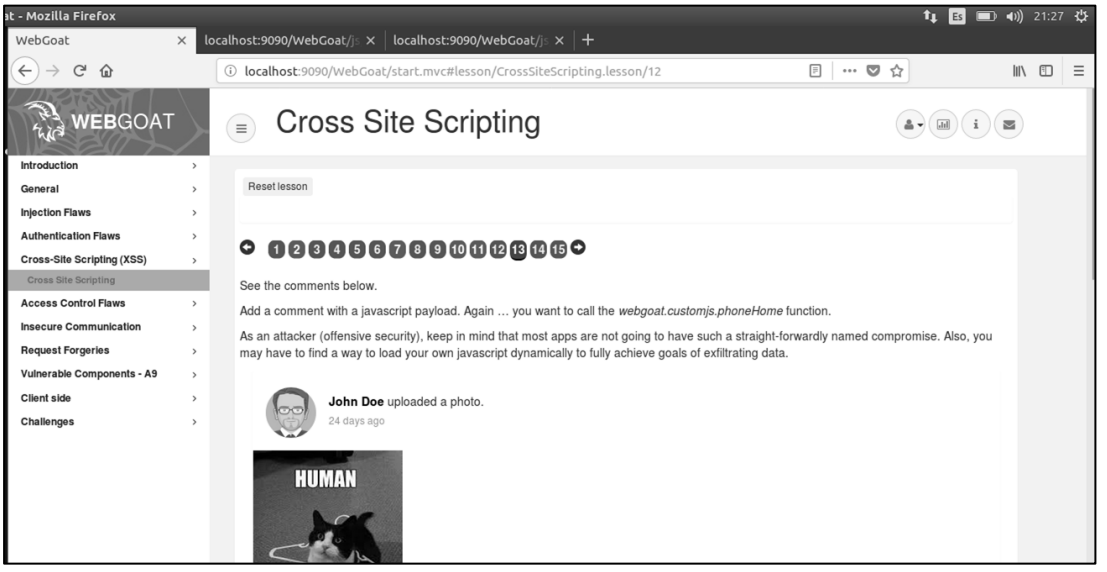
3.3.6. Probando *Cross Site Scripting*

En el apartado 14 de la lección *Cross-Site Scripting* de la aplicación WebGoat, se presenta un ejercicio práctico que permite realizar pruebas de inyección XSS (*cross site scripting*). En este ejercicio se solicita que mediante realizar un comentario en la página web presentada se ingrese código JavaScript. De forma que se ejecute una función interna de la aplicación WebGoat, al momento de cargar la información guardada en los comentarios. La función a ejecutar es la siguiente:

webgoat.customjs.phoneHome ()

La función *phoneHome*() al momento de ser consumida, ejecuta una petición post en la cual la aplicación responderá un número aleatorio, que posteriormente debe ingresarse en el campo proporcionado dentro del ejercicio, para finalizarlo exitosamente. Para obtener este valor se debe capturar la respuesta devuelta de la petición post. En la figura 94 se muestra el ejercicio dentro de la aplicación WebGoat.

Figura 94. **Ejercicio *Cross Site Scripting***



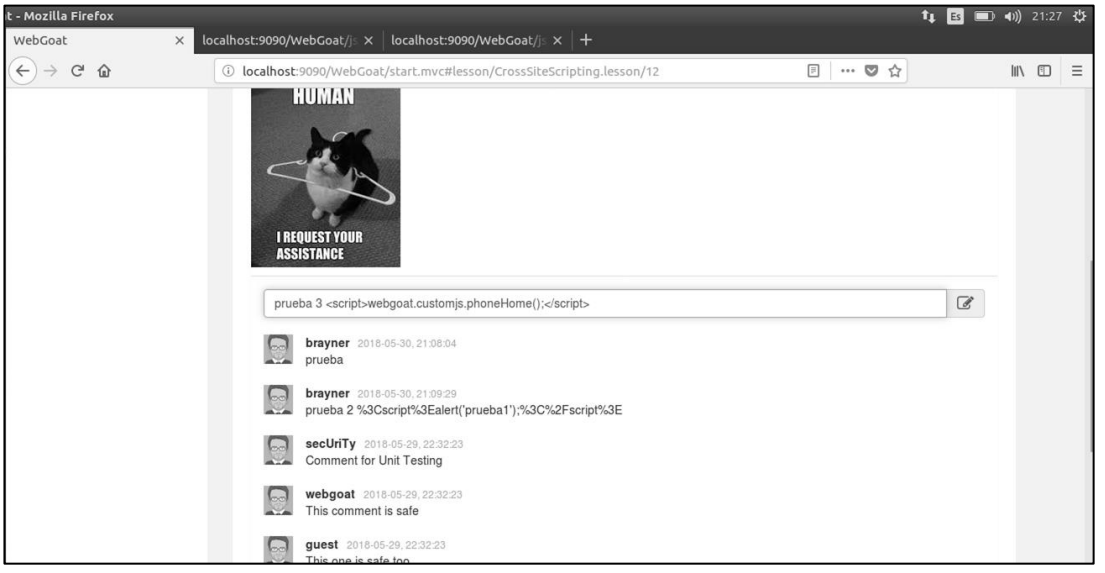
Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 30 de mayo de 2018.

Para completar el ejercicio de *Cross Site Scripting*, se debe ingresar un valor que permita ejecutar la función JavaScript que se encuentra en el archivo de ruteo de la aplicación. De manera que al ejecutarse se obtenga el valor aleatorio que la aplicación brinda y así finalizar el ejercicio exitosamente. El valor a ingresar es el siguiente:

```
<script>webgoat.customjs.phoneHome();</script>
```

En la figura 95 se muestra la solución del ejercicio.

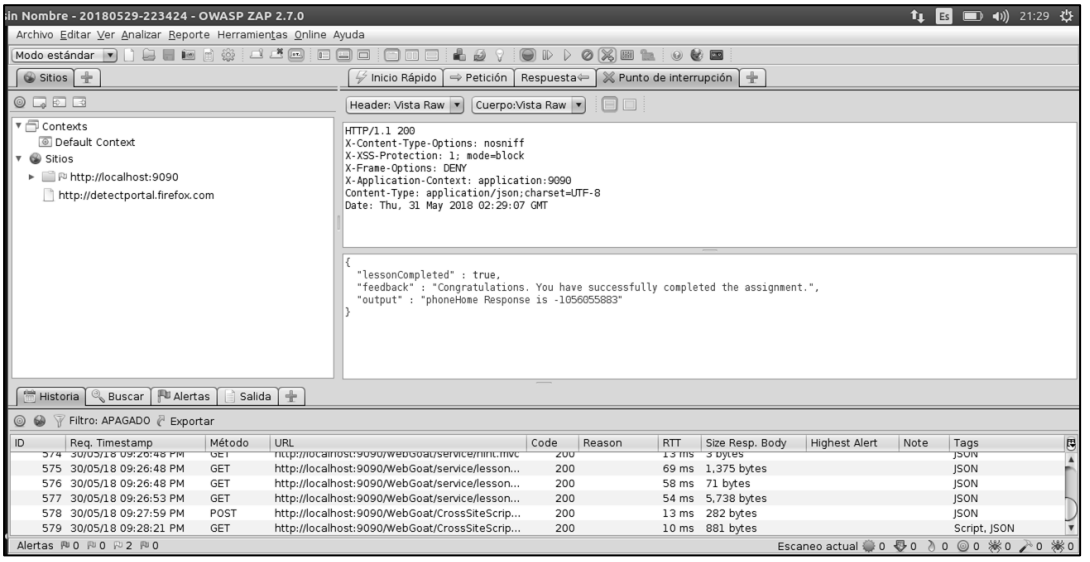
Figura 95. **Solución ejercicio *Cross Site Scripting***



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 30 de mayo de 2018.

Posterior al ejecutarse la función, se debe capturar la respuesta obtenida del servidor, en este caso se usa la aplicación OWASP ZAP, con este fin. En la figura 96 se muestra la respuesta capturada.

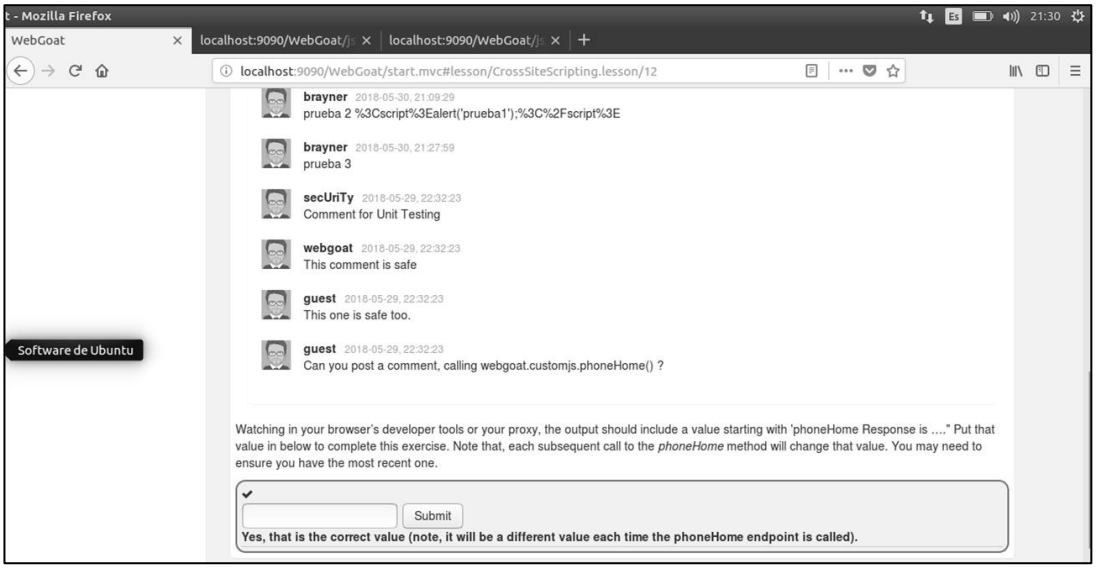
Figura 96. **Respuesta de función *phoneHome* XSS**



Fuente: elaboración propia con base en aplicación OWASP ZAP. Consulta: 30 de mayo de 2018.

Una vez ingresado el valor obtenido de la ejecución de la función *phoneHome*, en el campo de texto correspondiente de la aplicación, se completa el ejercicio de manera exitosa, como se muestra en la figura 97.

Figura 97. **Ejercicio finalizado de *Cross Site Scripting***



Fuente: elaboración propia con base en aplicación WebGoat. Consulta: 29 de mayo de 2018.

3.3.7. Mitigación inyección XSS

Existe una gran cantidad de vectores para realizar ataques usando inyección XSS, los cuales no se abarcarán en este documento. Sin embargo, existen reglas básicas que al cumplirlas conllevan a defenderse de cualquier variante de la inyección XSS, de manera que los activos e información de una empresa o entidad se encuentren seguros. Una de las formas elementales para defenderse de este ataque consiste en realizar las validaciones y escapes necesarios en las entradas de información del servidor.

Debido a las diferentes variaciones que existen del ataque de inyección XSS, la manera más efectiva para protegerse es usar un modelo positivo de prevención. Este modelo se basa en negar todo lo que no está permitido y aceptar solo elementos predefinidos, conocido comúnmente como modelo de lista blanca. De manera que el desarrollador tendrá a disposición espacios en la página HTML (que en este caso será tratado como una plantilla) donde se permite colocar datos que no son confiables. Es importante recalcar que colocar datos no confiables en un lugar no establecido para ello, está prohibido.

Debido a la manera en que los navegadores analizan la información, cada uno de los espacios usados para colocar datos no confiables tendrá reglas de seguridad ligeramente diferentes. De manera que se deben seguir ciertos pasos para asegurar que los datos ingresados en los espacios destinados a datos no confiables y así asegurar que no salgan del contexto y se cree la posibilidad de ejecución de código. Este enfoque trata el documento HTML como una consulta de base de datos parametrizada, donde los datos se guardan en lugares específicos aislados de la ejecución de código.

En algunas ocasiones se considera que solo utilizando una codificación HTML, se mitigará el ataque XSS. Esta premisa es verdadera cuando se usa solamente para los elementos que componen un documento HTML, por ejemplo en etiquetas *div*. Sin embargo deja de ser funcional cuando los elementos maliciosos se incluyen en etiquetas del tipo *script*, en atributos de los diferentes controladores de eventos, dentro de CSS o en una URL. Por lo tanto se necesita establecer reglas que ayuden a proteger los recursos de este ataque, de las cuales se hablarán en el siguiente apartado.

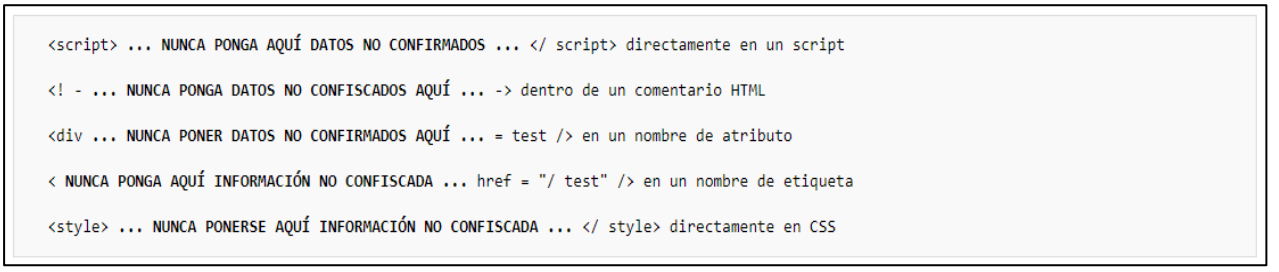
3.3.7.1. Reglas de prevención XSS

Las reglas que se mencionan a continuación tienen como finalidad la prevención de inyección XSS, buscan permitir libertad para el ingreso de datos no confiables en un documento HTML, no son absolutas. Dependiendo del modelo de negocio que la aplicación a implementar maneje, se debe tomar la decisión de cuales reglas serían apropiadas para aplicar. Es importante recalcar que estas reglas no deben usarse pensando en reglas de lista negra, de manera que solo se obvien los elementos mencionados en ellas, si no en una lista blanca que permitirá defenderse, inclusive de futuros métodos de ataque XSS.

3.3.7.1.1. Regla #0. Denegar todo

La primera regla implica que no se inserten datos no confiables a excepción de en los lugares que se definan en las reglas #1 a #5. El motivo de la existencia de la regla #0 es que existe una gran cantidad de diferentes contextos HTML, lo que complica la creación de reglas de excepción a evitar en una aplicación. Entre los contextos se mencionan los contextos anidados, que implica el uso de URLs dentro de métodos o funciones JavaScript. En la figura 98 se muestran ejemplos de contextos donde se debe evitar a toda costa el ingreso de datos sin confirmar.

Figura 98. **Contextos inseguros para datos no confirmados**



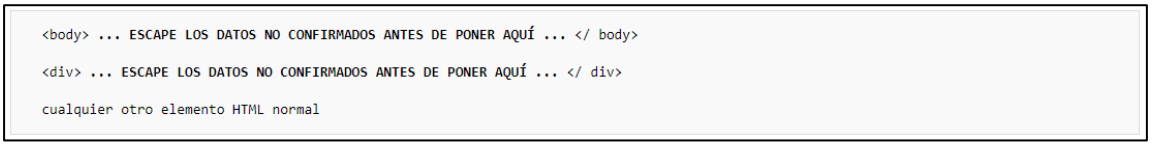
Fuente: OWASP.
https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet.
Consulta: 09 de junio de 2018.

Es importante recalcar que nunca se debe aceptar código JavaScript real de una fuente que no sea confiable para ejecutarse posteriormente.

3.3.7.1.2. Regla #1. Escape HTML

Esta regla es utilizada cuando se desea insertar datos no confiables directamente en el cuerpo del documento HTML, dentro de etiquetas como: *div*, p, b, td, th, entre otras. La mayoría de los *frameworks* usados en aplicaciones Web tienen métodos de escape para los elementos que se muestran en la imagen 99. Es importante hacer hincapié que no es suficiente usar solo estos métodos ya que existen contextos que no son abarcados.

Figura 99. **Contextos que necesitan escape de datos ingresados**



Fuente: OWASP.
https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet.
Consulta: 09 de junio de 2018.

Debido a que existe la posibilidad de que haya contextos que no se tomen en cuenta, es necesario realizar el escape de ciertos caracteres que tienen un significado en los elementos HTML, así se evitará el cambio a cualquier contexto de ejecución como: *script*, estilo o controladores de eventos. En la tabla X se muestra la conversión a entidades hexadecimales de los caracteres útiles en HTML.

Tabla X. **Caracteres con codificación de entidades HTML**

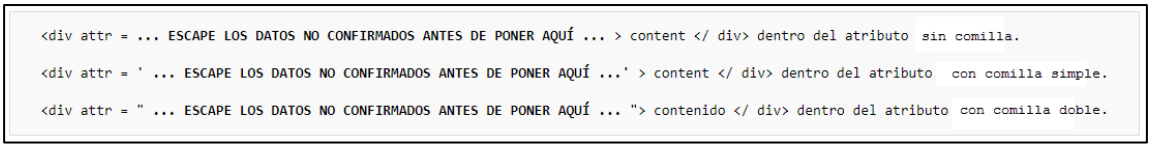
Carácter	Codificación
&	&amp;
<	<
>	>
"	"
'	'
/	/

Fuente: OWASP.
https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet.
Consulta: 09 de junio de 2018.

3.3.7.1.3. Regla #2. Escape de atributos

Esta regla es usada para colocar datos no confiables dentro de los valores de atributos en un documento HTML, tales como: ancho, nombre, valor, entre otros. Es importante recalcar que esto no debe utilizarse para atributos complejos como *href, src, style* o cualquier controlador de eventos adicional. En la figura 100 se muestran ejemplos de los contextos donde se deben escapar los datos no confiables de manera que se cumpla esta regla.

Figura 100. **Contextos de escape atributos HTML**



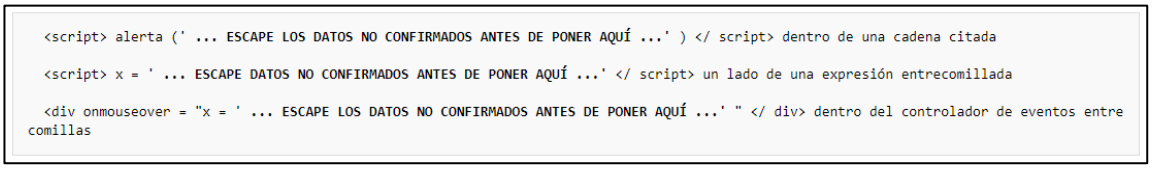
Fuente: OWASP.
https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet.
Consulta: 09 de junio de 2018.

Entre el contenido no confiable, se debe convertir todos los caracteres con valores ASCII menores a 256, a excepción de los caracteres alfanuméricos, con el siguiente formato: &#xHH. También existe la posibilidad de usar una entidad con nombre, si esta definición se encuentra habilitada. La razón de esta regla, se debe a que en ocasiones los programadores usan atributos sin usar comillas para encerrarlos. Debido a que los atributos que se encuentran encerrados entre comillas se escapan con estas. Sin embargo los atributos que no se encuentran encerrados entre comillas son quebrados utilizando una sin fin cantidad de caracteres.

3.3.7.1.4. Regla #3. Escape de JavaScript

La tercera regla aplica para el código JavaScript que una aplicación genera de manera dinámica, refiriéndose a bloques de scripts como a los atributos de controladores de eventos. El único lugar seguro para colocar datos no confiables en este contexto, es dentro de un valor de datos que se encuentre entre comillas. Utilizar datos no confirmados en otro contexto es sumamente peligroso, ya que sin comillas es bastante sencillo el cambio de contexto a ejecutar y esto vuelve vulnerable a un ataque a la aplicación. En la figura 101 se muestran ejemplos de contexto donde se incluyen datos no confiables.

Figura 101. **Contextos de escape funciones JavaScript**



Fuente: OWASP.
https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet.
Consulta: 09 de junio de 2018.

Existen funciones en JavaScript que no existe manera segura de ingresar datos no confiables, por lo tanto, no se deben usar datos ingresados por el cliente en estas. A continuación se listan algunos ejemplos de funciones y en la figura 102 se muestra un ejemplo gráfico del uso de las mismas:

- .eval().
- .innerHTML().
- .html().

- .append().
- .setInterval().

Figura 102. **Ejemplo de función insegura JavaScript**

```
<script>
window.setInterval (' ... AUN CUANDO ESCAPES LOS DATOS QUE NO SE ENCUENTRAN, USTED ESCRIBE AQUÍ ...' );
</ script>
```

Fuente: OWASP.
https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet.
Consulta: 09 de junio de 2018.

En estos casos se debe escanear todos los caracteres de 256, a excepción de los caracteres alfanuméricos, con el siguiente formato: \xHH. Con esto se evita el cambio de valor en los datos dentro del contexto o atributo usado. No deben utilizarse caracteres de escape como '\', debido a que crea la posibilidad de coincidir con analizador de atributos que se ejecuta primero y haga que la acción malintencionada se realice. Además son susceptibles a ataques de escape, donde el atacante envía el mismo valor para que el código enviado posteriormente sea ejecutado.

3.3.7.1.5. Regla #3.1. Escape de valores JSON

En la web 2.0 que se usa actualmente, la necesidad de tener datos generados de manera dinámica usando un contexto de JavaScript, crea un riesgo adicional. Por lo regular se utilizan métodos Ajax buscando mantener la seguridad, lo que no es siempre efectivo. Las aplicaciones crean un bloque inicial de JSON, que se carga en la página, de manera que sea el único lugar usado para el almacenamiento de datos. De tal forma que la información que se guarda se vuelva difícil de obtener sin romper el formato y el contenido de los valores.

La primera defensa a implementar en estos casos, es la validación del tipo de documento que está enviando una petición al servidor. En este caso se debe asegurar que el *Content-type* sea del tipo JSON y no de otro tipo que sea vulnerable a que el navegador mal interprete el contexto y realice la ejecución de *scripts* inyectados, con fines maliciosos. Además de esto es necesario realizar una codificación de los datos enviados usando alguna de las siguientes opciones:

- Serializar JSON: existen herramientas que permiten serializar una cadena a código JavaScript, de manera que este sea agregado de forma segura en etiquetas *script*. Los caracteres HTML y los finalizadores de JavaScript deben escaparse.[14]
- Codificación de entidades HTML: esta técnica ayuda a separar los datos del código del lado del servidor sin cruzar los límites de contexto donde deben ejecutarse. La forma en que se usa es colocar el documento JSON, en la página como un elemento normal y posteriormente analizar el código HTML, para analizar los elementos.

[14] Un ejemplo de serializador es el proporcionado por Yahoo, que se puede encontrar en el siguiente enlace: https://github.com/yahoo/serialize-javascript.

En la figura 103 se muestra un ejemplo de uso de estas técnicas.

Figura 103. **Ejemplo de escape de JSON**

```
<div id = "init_data" style = "display: none">
   <% = html_escape (data.to_json)%>
</ div>

// archivo js externo
var dataElement = document.getElementById ('init_data');
// decodificar y analizar el contenido del div
var initData = JSON.parse (dataElement.textContent);
```

Fuente: OWASP.
https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
Consulta: 10 de junio de 2018.

3.3.7.1.6. Regla #4. Escape de CSS

Esta regla es usada cuando se desea colocar información no confiable en una hoja o etiqueta de estilo, que compone a un documento HTML. Es imprescindible reconocer que el uso de CSS, crea una posibilidad de un ataque inminente debido a ser una herramienta muy poderosa. Por lo tanto es importante solo usar datos no confiables en un valor de propiedad y no en otros lugares. Además de evitar el uso de datos no confiables en lugares en propiedades complejas como: *url*, comportamiento y personalizado. En la figura 104 se muestran contextos CSS donde se incluyen datos no confiables.

Figura 104. **Contextos de escape CSS**

```
<style> selector {propiedad: ... ESCAPE LOS DATOS NO CONFIRMADOS ANTES DE PONER AQUÍ ... ;  } </ style> valor de la propiedad

<style> selector {propiedad: " ... ESCAPE LOS DATOS NO CONFIRMADOS ANTES DE PONER AQUÍ ... ";  } </ style> valor de la propiedad

<span style = "property: ... ESCAPE LOS DATOS NO CONFIRMADOS ANTES DE PONER AQUÍ ... "> text </ span> valor de la propiedad
```

Fuente: OWASP.
https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet.
Consulta: 10 de junio de 2018.

Es importante recalcar que existen contextos CSS en los que no existe forma segura de colocar datos no confiables como datos de entrada. Además debe asegurar que las URL comiencen de manera correcta con “http” y no usando “javascript”, ni expresiones, como se muestra en la figura 105.

Figura 105. **Valores a evitar en contextos CSS**

```
{background-url: "javascript: alerta (1)";  } // y todas las demás URL
{text-size: "expresión (alerta ('XSS'))";  } // solo en IE
```

Fuente: OWASP.
https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet.
Consulta: 10 de junio de 2018.

En estos contextos los caracteres con valores ASCII menores a 256, deben escaparse a excepción de los caracteres alfanuméricos, el formato para el escape es el siguiente: \HH. No debe utilizarse ningún atajo para el escape, debido a que existe la posibilidad que el carácter usado coincida con el analizador de atributos HTML, que se ejecuta primero.

3.3.7.1.7. Regla #5. Escape de URL

Esta regla es utilizada cuando se ingresan valores o datos no confiables en los parámetros de URL en el método *get* de HTTP. En la figura 106 se muestra un ejemplo de este contexto.

Figura 106. **Ejemplo de contextos URL**

```
<a href="http://www.somesite.com?test= ...ESCAPE DATOS NO CONFIRMADOS ANTES DE PONER AQUÍ ..."> enlace </ a>
```

Fuente: OWASP. https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet. Consulta: 10 de junio de 2018.

Los caracteres con valores ASCII menores a 256, deben escaparse a excepción de los caracteres alfanuméricos, el formato para el escape es el siguiente: %HH. No deben permitirse las URL, debido a que no hay forma de validar que se realice algún cambio de contexto. Estos atributos tienen que estar encerrados entre comillas, debido a que los atributos sin estas, son vulnerables a modificarse con una gran cantidad de caracteres.

Es importante recalcar que no se debe codificar ninguna dirección URL completa o parcial con codificación URL. Si los datos no confiables se ingresan en atributos basados en URL, como: *href, src*, entre otros. Los valores recibidos deben validarse, de manera que se asegure que no se utilice un protocolo inesperado, especialmente enlaces JavaScript. El proceso de codificación dependerá del contexto de visualización, como cualquier otro dato. En la figura 107 se muestra un ejemplo de validación de datos no confiables para URL.

Figura 107. **Validación de datos no confiables URL**

```
String userURL = request.getParameter ("userURL")
boolean isValidURL = Validator.IsValidURL (userURL, 255);
if (isValidURL) {
    <a href="<%=encoder.encodeForHTMLAttribute(userURL)%> "> enlace </a>
}
```

Fuente: OWASP.
https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet.
Consulta: 10 de junio de 2018.

3.3.7.1.8. Regla #6. Sanear marcado HTML

Si la aplicación web utiliza secuencia de caracteres o símbolos que se insertan en ciertos lugares con el fin de dar formato a texto, conocido como marcado HTML, una entrada que posee código HTML es muy difícil de validar. La codificación también se complica debido a que se corre el riesgo de romper las reglas que la entrada contiene. Por lo tanto, se necesita una biblioteca que tenga la capacidad de analizar y limpiar texto con formato HTML. El proyecto OWASP, recomienda las siguientes:

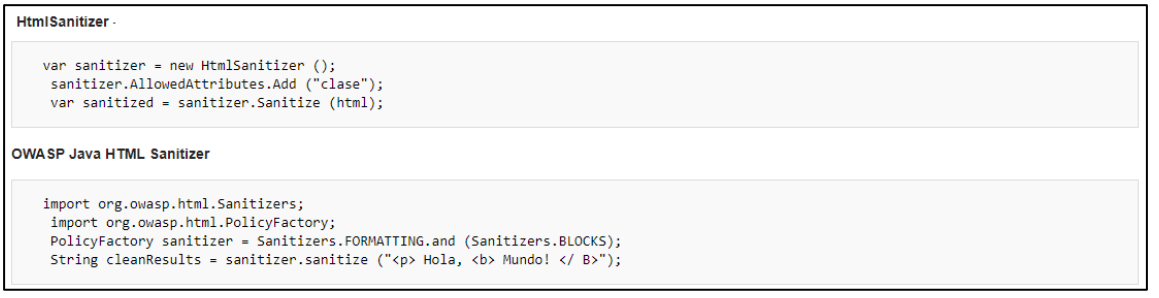
- HtmlSanitizer: biblioteca .Net de fuente abierta, que se base en la limpieza de HTML basado en un modelo de lista blanca, en la que se configuran todos los atributos y métodos permitidos.[15]

[15] Github: https://github.com/mganss/HtmlSanitizer. Consulta: 10 de junio de 2018.

- OWASP Java HTML Sanitizer: es una aplicación escrita en java utilizada para sanear HTML de manera fácil y rápida, por medio de configuraciones. Ha sido escrito utilizando las mejores prácticas de seguridad en mente, posee un gran conjunto de pruebas y es sometido a revisiones constantes.[16]

En la figura 108 se muestran ejemplos de las aplicaciones usadas para sanear datos HTML.

Figura 108. **Ejemplos de bibliotecas para sanear HTML**



Fuente: OWASP.
https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet.
Consulta: 10 de junio de 2018.

[16] OWASP: https://www.owasp.org/index.php/OWASP_Java_HTML_Sanitizer_Project. Consulta: 10 de junio de 2018.

4. PRUEBAS DE INYECCIÓN EN SISTEMA EXISTENTE

En este capítulo se analizan las pruebas realizadas a una aplicación web obtenida con este fin, basándose en los conocimientos adquiridos en el capítulo tres sobre las vulnerabilidades que crean la posibilidad de sufrir un ataque de inyección a un sistema web. Para estas pruebas se utiliza la aplicación SUN RISE[17], sistema web de código abierto que se encuentra en internet, creado para realizar el manejo de reservaciones en un hotel. El propósito es determinar si la aplicación posee vulnerabilidades que comprometan la información que contiene y la seguridad al momento de sufrir un ataque por una entidad externa o inclusive por algún usuario de la misma.

Se explican los diferentes escenarios a analizar y los resultados que se esperan obtener al efectuar las diferentes pruebas. De manera que se tenga conocimiento de los objetivos a lograr, el comportamiento esperado y cuál es el resultado obtenido.

4.1. Inyección SQL

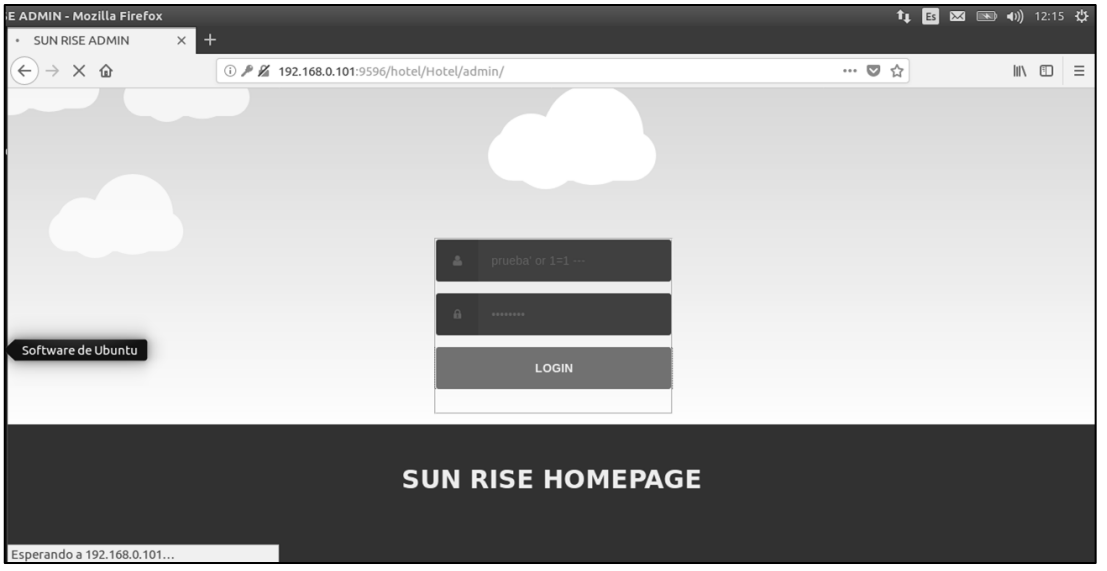
Esta vulnerabilidad se presenta cuando dentro del diseño y desarrollo de una aplicación, conlleva la creación de SQL dinámicos, que crean la posibilidad de exponer los datos e información del aplicativo. A continuación se mencionan y detallan las pruebas realizadas sobre el aplicativo y el resultado obtenido.

[17] *Hotel management project in PHP with MySQL source code free download*: https://wikiphp1.blogspot.com/2017/09/hotel-management-project-in-php-with.html. Consulta: 12 de junio de 2018

4.1.1. Validación de ingreso sin credenciales válidas

En este caso de prueba se busca ingresar a la aplicación sin hacer uso de credenciales que estén registradas. El proceso consiste en validar como la aplicación verifica las credenciales para el acceso de los diferentes usuarios, determinando si es posible realizar inyección SQL, para evadir la comprobación de credenciales. En la figura 109 se muestra el valor ingresado para realizar el primer ataque a la aplicación.

Figura 109. **Ataque de inyección SQL 1, SUN RISE**



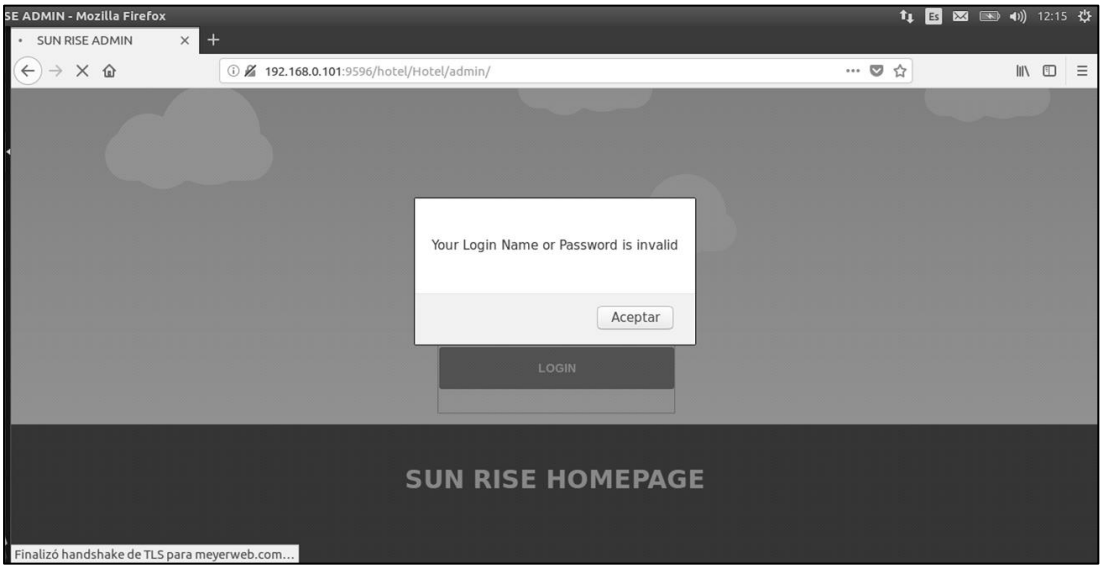
Fuente: elaboración propia con base en aplicación SUN RISE. Consulta: 30 de junio de 2018.

Como se observa en la figura 109 se busca que la aplicación acepte valores para saltar la verificación de ingreso. La forma de realizar la prueba es mediante el ingreso de valores que harían que una consulta directa a la base de datos devuelva un valor verdadero y permita el ingreso a la aplicación. Esta prueba se hace bajo la premisa que la aplicación concatena los valores enviados desde la aplicación y crea un *query* dinámico para la validación de ingreso.

4.1.1.1. Resultado obtenido

En la figura 110 se muestra el resultado obtenido en la aplicación, posterior a realizar la prueba.

Figura 110. **Resultado de ataque de inyección SQL 1, SUN RISE**



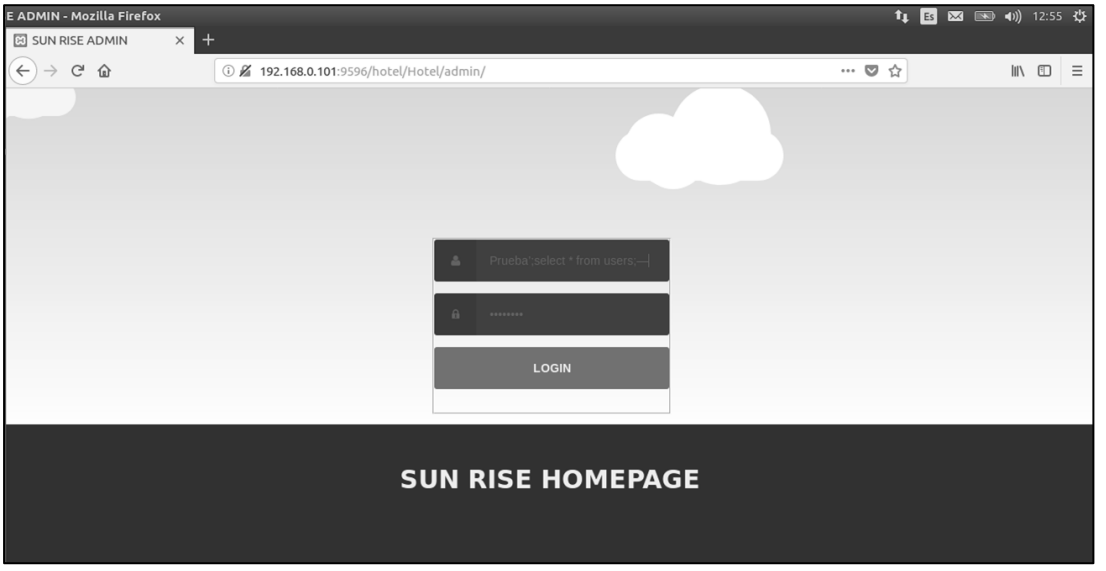
Fuente: elaboración propia con base en aplicación SUN RISE. Consulta: 30 de junio de 2018.

La premisa mencionada anteriormente se comprueba al observar como la aplicación se comporta después de haber enviado los valores. Al analizar el resultado en la figura 110 se muestra que la aplicación no se comporta de la manera esperada, debido a que no se tuvo éxito al ingresar a la aplicación sin contar con un usuario y una contraseña valida.

4.1.2. Obtención de información usando errores de la aplicación

En este caso de prueba, se busca la obtención de información que se encuentra almacenada dentro de la base de datos del aplicativo, haciendo uso de los errores que la aplicación presenta. Este objetivo se logra mediante el ingreso de valores que la aplicación no espera, en forma que al presentar un error, se obtenga información de la misma. En la figura 111 se muestra el valor ingresado para realizar el segundo ataque a la aplicación.

Figura 111. **Ataque de inyección SQL 2, SUN RISE**



Fuente: elaboración propia con base en aplicación SUN RISE. Consulta: 30 de junio de 2018.

En este escenario de prueba se realiza el ingreso de un usuario inexistente acompañado de una consulta directa a la base de datos, en el campo para ingreso a la aplicación. El valor usado para la prueba, como se observa en la figura 111, es el siguiente:

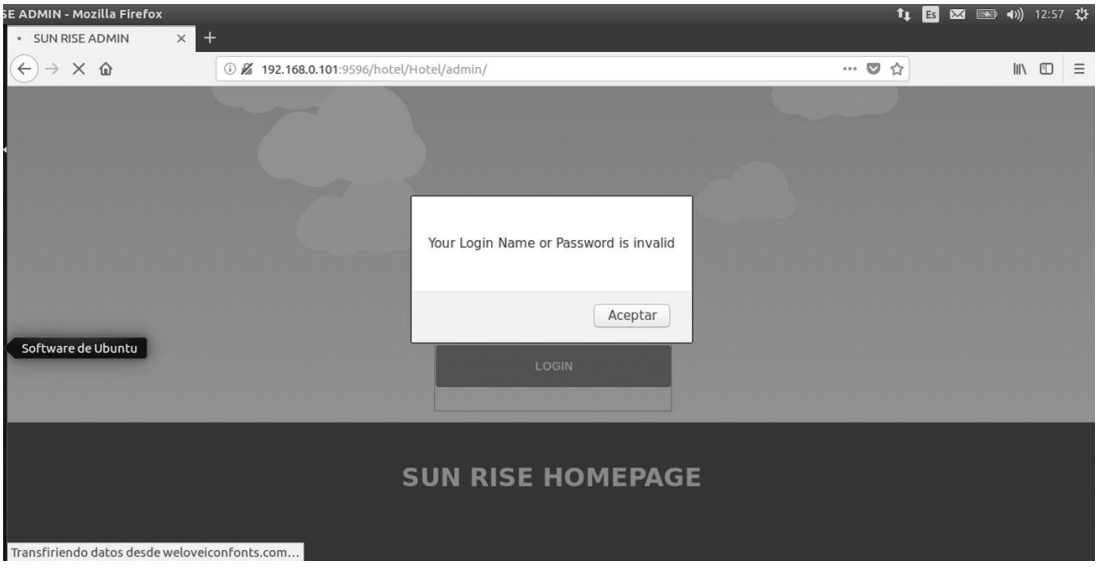
Prueba'; select * from users;—.

Con el valor se busca obtener información de la base de datos, mediante la premisa que la aplicación crea un *query* dinámico, cuando valida los datos de usuario y devolverá un error con detalles de lo sucedido al no obtener los datos esperados.

4.1.2.1. Resultado obtenido

En la figura 112 se muestra el resultado que se obtiene en la aplicación, posterior a realizar la prueba.

Figura 112. **Resultado de ataque de inyección SQL 2, SUN RISE**



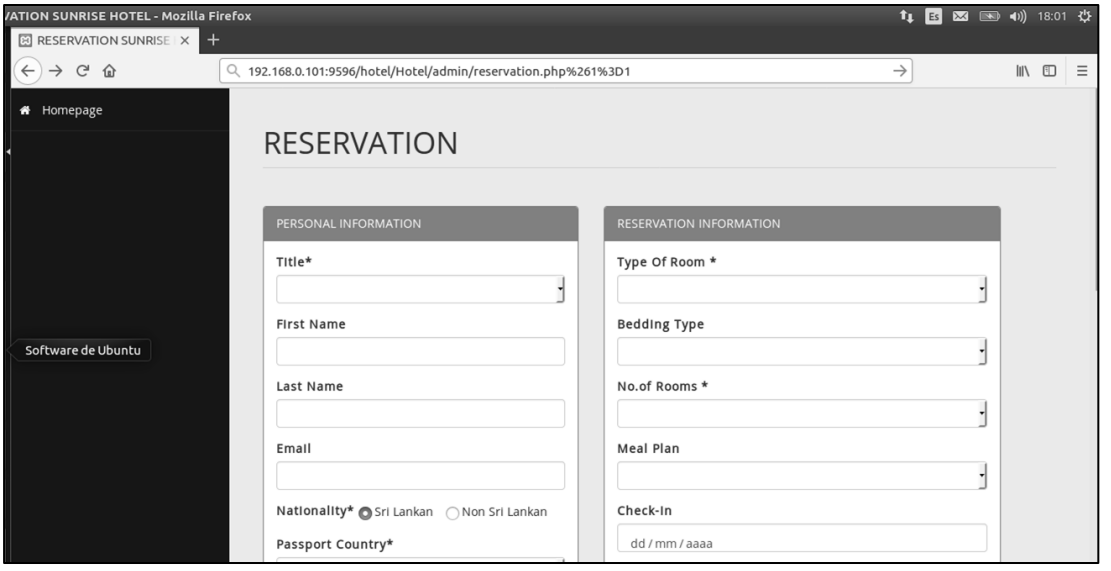
Fuente: elaboración propia con base en aplicación SUN RISE. Consulta: 30 de junio de 2018.

La suposición mencionada en la prueba se aclara al obtener una respuesta de la aplicación, como se muestra en la figura 112 claramente la aplicación está diseñada de manera que no se muestren errores de base de datos al usuario. Por lo tanto la suposición no se presenta como se esperaba.

4.1.3. Validación de inyección ciega de SQL

En este caso de prueba, se busca verificar que la aplicación acepte ingreso de valores SQL sin observar resultados obvios, como un error en pantalla. La forma en validar el funcionamiento de la prueba es observar el comportamiento que la aplicación demostrará, al ingresar valores booleanos dentro los datos enviados en la aplicación para el ruteo de las páginas a mostrar. En la figura 113 se muestra el valor ingresado para realizar el tercer ataque a la aplicación.

Figura 113. **Ataque de inyección SQL 3, SUN RISE**



Fuente: elaboración propia con base en aplicación SUN RISE. Consulta: 30 de junio de 2018.

En este escenario de prueba se realiza el ingreso de un valor en la dirección URL, utilizada por la aplicación para acceder a los diferentes recursos que esta ofrece. De manera que el valor ingresado es del tipo booleano (verdadero o falso), para este caso se utilizan datos con valor verdadero como se muestra a continuación:

&1=1.

Los datos usados para la prueba se codifican en código por ciento (caracteres utilizados en las direcciones URL, para representar valores alfanuméricos). Al realizar la conversión de los datos usados en la figura 113, los valores quedan de la manera siguiente:

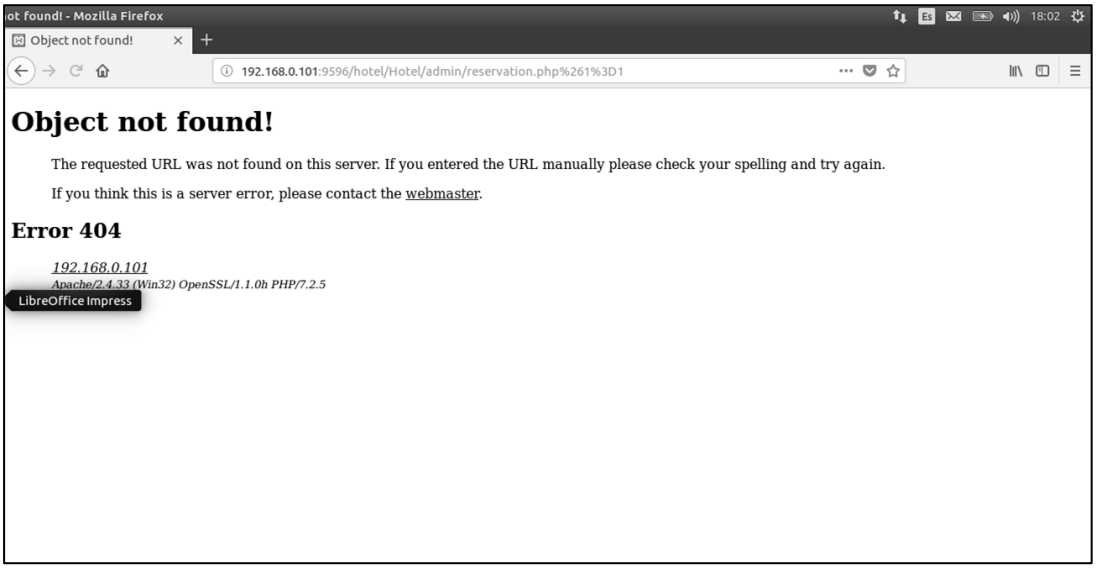
%261%3D1.

El comportamiento esperado en la prueba, es que la aplicación al recibir los datos enviados, devuelva la misma información mostrada antes de ingresarlos. De esta manera se comprueba si la aplicación es vulnerable al ataque de inyección SQL, aunque este diseñada para mostrar errores genéricos que no brinden mayor información al usuario.

4.1.3.1. Resultado obtenido

En la figura 114 se muestra el resultado obtenido en la aplicación, posterior a la realización de la prueba.

Figura 114. **Resultado de ataque de inyección SQL 3, SUN RISE**



Fuente: elaboración propia con base en aplicación SUN RISE. Consulta: 30 de junio de 2018.

En la figura 114 se muestra que el resultado obtenido de la aplicación es un error que no brinda ninguna información adicional de la página, lo que corrobora que el funcionamiento previsto como objetivo de la prueba no se cumple. Por lo tanto la aplicación no es vulnerable a una inyección ciega de SQL.

4.2. Inyección XSS

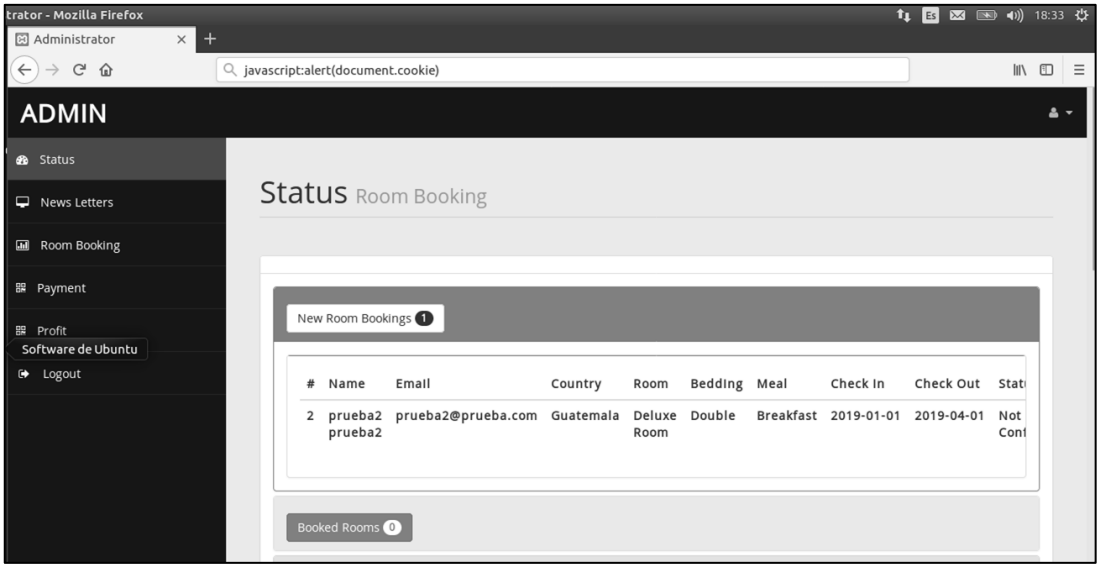
Esta vulnerabilidad se presenta en aplicaciones que usan el lenguaje JavaScript, obteniendo valores enviados por el usuario para realizar alguna acción. De manera que se aproveche esta función para el envío de código malicioso que afecta posteriormente a la aplicación. A continuación se mencionan las pruebas realizadas sobre el aplicativo y el resultado obtenido.

4.2.1. Obtención de *cookies* mediante la URL de la página

El fin de esta prueba es determinar si la aplicación permite el uso de código JavaScript dentro de la URL, con la que se accede desde el navegador. De manera que se permita la ejecución de una alerta para obtener información. El modo de prueba es utilizar una función de alerta en JavaScript, solicitando la información de la *cookie* que usa el cliente de la aplicación, de manera que se determine si existe información que sea accedida mediante este método y si esto representaría una vulnerabilidad en el aplicativo.

En la figura 115 se muestra el valor ingresado para realizar el primer ataque a la aplicación usando XSS.

Figura 115. **Ataque de inyección XSS 1, SUN RISE**



Fuente: elaboración propia con base en aplicación SUN RISE. Consulta: 30 de junio de 2018.

En este escenario de prueba se realiza el ingreso de un valor en la dirección URL, utilizada por la aplicación para acceder a los diferentes recursos que esta ofrece. De manera que el valor ingresado es la llamada a una función de alerta, que devuelve el valor de la *cookie* usada en el aplicativo, como se observa en la figura 115 el valor usado es el siguiente:

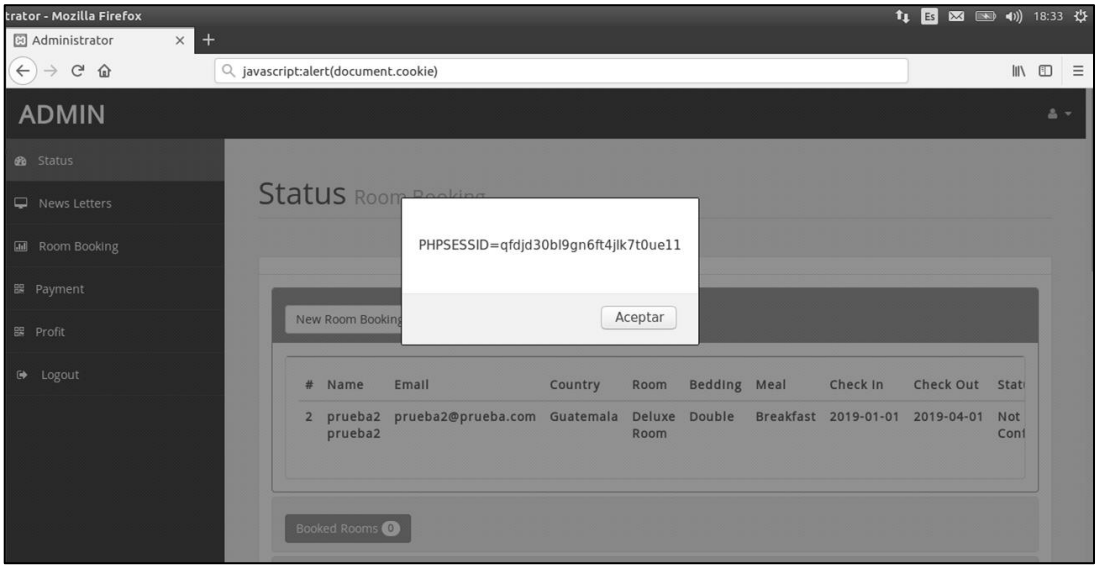
Javascript:Alert(document.cookie);.

El comportamiento esperado en la prueba, es que la aplicación al recibir los datos enviados, devuelva la información de los datos guardados del usuario en la *cookie* que tiene el documento. De esta manera se comprueba si la aplicación es vulnerable al ataque de inyección XSS, al permitir la ejecución de funciones ajenas al desarrollo original.

4.2.1.1. Resultado obtenido

En la figura 116 se muestra el resultado obtenido en la aplicación, posterior a la ejecución de la prueba.

Figura 116. **Resultado de ataque de inyección XSS 1, SUN RISE**



Fuente: elaboración propia con base en aplicación SUN RISE. Consulta: 30 de junio de 2018.

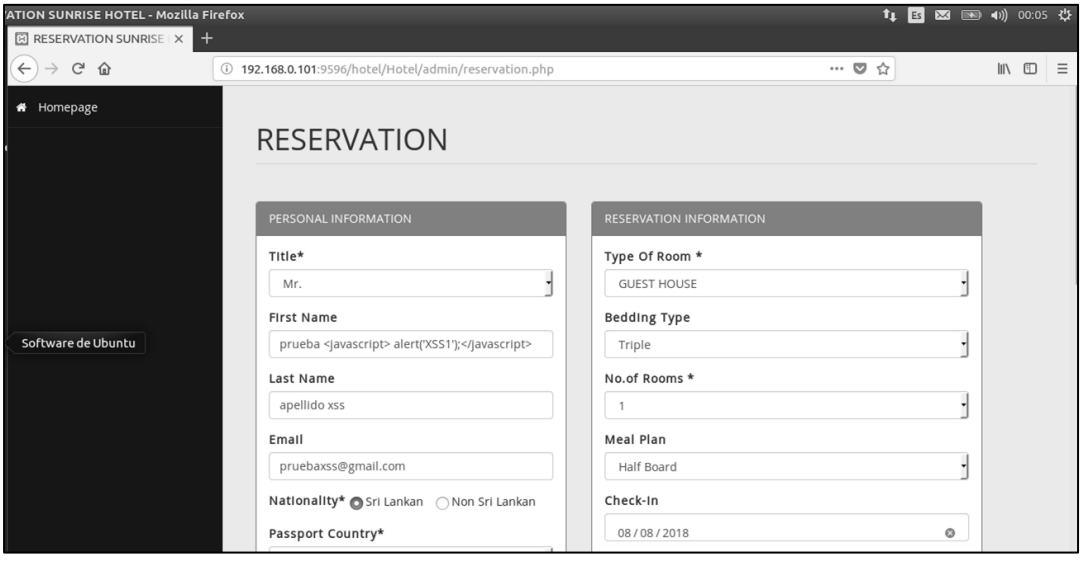
En la figura 116 se muestra que el resultado obtenido de la aplicación es un mensaje de alerta como se esperaba, por lo tanto la aplicación es vulnerable a inyección XSS, mediante el uso de la barra de direcciones. Es importante mencionar que el resultado obtenido no es legible como para usarse en un ataque, lo que logra que sea una vulnerabilidad con poco riesgo.

4.2.2. Almacenaje de código JavaScript

El objetivo de esta prueba, es determinar si existe forma de almacenar código JavaScript dentro de la aplicación, que posteriormente se ejecutará al obtener la información del servidor y mostrarla al cliente. El modo de prueba es el ingreso de funciones JavaScript dentro de los diferentes valores que la aplicación almacena, de manera que se guarde dentro de la base de datos, para comprobar si al mostrar información de los valores ingresados las funciones se ejecutan, de manera que la aplicación sea vulnerable ante ataques de JavaScript.

En la figura 117 se muestra el valor ingresado para realizar el segundo ataque a la aplicación usando XSS.

Figura 117. **Segundo ataque de inyección XSS, SUN RISE**



Fuente: elaboración propia con base en aplicación SUN RISE. Consulta: 30 de junio de 2018.

En este escenario de prueba se realiza el ingreso de una función JavaScript en el campo destinado para el nombre de la persona que realiza la reservación, posterior se llenan los campos adicionales. De manera que el valor ingresado es la llamada a una función de alerta, que se ejecutará al momento de obtener los valores de la base de datos y mostrarlos en el cliente. Como se observa en la figura 117 el valor usado es el siguiente:

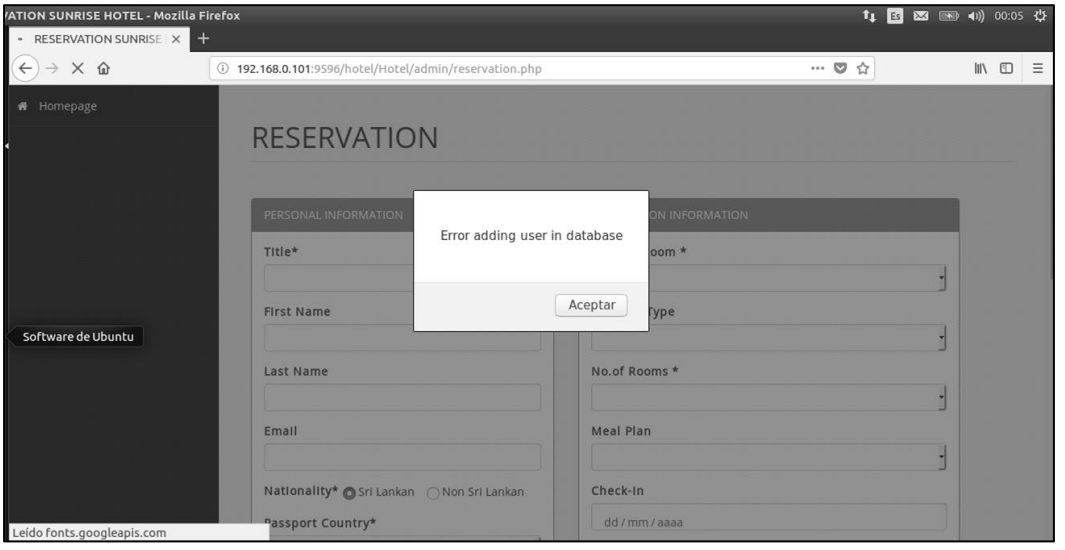
Prueba<javascript> alert(‘XSS1’); </javascript>.

El comportamiento esperado en la prueba, es que la aplicación permita el almacenamiento de la información que incluye la función de JavaScript, de manera que se ejecute posteriormente al mostrar dicha información. De esta manera se compruebe si la aplicación es vulnerable al ataque de inyección XSS, al permitir la ejecución de funciones ajenas al desarrollo original, que se guardan en la base de datos.

4.2.2.1. Resultado obtenido

En la figura 118 se muestra el resultado obtenido en la aplicación, posterior a la ejecución de la prueba en la aplicación.

Figura 118. **Resultado de segundo ataque de inyección XSS, SUN RISE**



Fuente: elaboración propia con base en aplicación SUN RISE. Consulta: 30 de junio de 2018.

En la figura 118 se muestra que el resultado obtenido de la aplicación es un mensaje de error, indicando que la información ingresada es incorrecta. Esto muestra que la aplicación no permite el almacenaje de funciones JavaScript, enviadas entre los valores que el cliente usa para guardar información en el servidor. Por lo que no existe la vulnerabilidad de ejecución de funciones JavaScript desde la base de datos.

4.3. Resumen de pruebas

En la tabla XI se resume el resultado obtenido en las distintas pruebas realizadas a la aplicación SUN RISE, de manera que se establezcan los hallazgos obtenidos.

Tabla XI. **Resumen de pruebas SUN RISE**

Tipo	Prueba	Estado	Existe vulnerabilidad
Inyección SQL	Validación de ingreso sin credenciales válidas.	Completado	No
Inyección SQL	Obtención de información usando error de la aplicación.	Completado	No
Inyección SQL	Validación de inyección ciega SQL.	Completado	No
Inyección XSS	Obtención de *cookies* mediante la información de la página.	Completado	Si
Inyección XSS	Almacenaje de código JavaScript	Completado	No

Fuente: elaboración propia.

CONCLUSIONES

1. Es posible el aprendizaje de vulnerabilidades en aplicaciones web, usando los ejercicios y contenido que tiene la aplicación WebGoat de manera interactiva.

2. Aprender de forma práctica, facilita el entendimiento de los elementos usados para realizar un ataque a una aplicación web, de manera que se mitiguen las posibles debilidades que se explotan.

3. Los ataques más comunes a aplicaciones web, son los ataques de inyección. Entre los cuales destacan los siguientes: ataques de inyección SQL, ataques de inyección XML y ataques de inyección de JavaScript.

4. Al momento de una aplicación web sufrir un ataque expone la información que posee, lo que llega a causar un gran daño tanto a la entidad que es responsable de la misma, como a las entidades dueñas de la información. Esto genera pérdidas desde el tipo monetarias, de reputación, inclusive causar inconvenientes en el marco legal.

5. Una parte esencial al realizar un desarrollo, es realizar validaciones respecto a los valores que el cliente envía al servidor. De manera que se verifique que sean los datos esperados y no datos que creen algún percance o problema tanto en el servidor como en el cliente.

6. Al momento de recibir datos no confiables del cliente al servidor que sean esenciales para el funcionamiento, se debe utilizar una política de lista blanca, de manera que se acepten solo los valores permitidos y todo lo demás sea ignorado o tomado como un error.

RECOMENDACIONES

1. Al realizar una aplicación web, se debe considerar dentro del plan de trabajo el análisis de vulnerabilidades que el código desarrollado presentará, de manera que entre los planes a realizar se encuentre una tarea destinada a establecer seguridad propia de la aplicación y así evitar correr riesgos innecesarios posterior a la implementación.

2. Los datos que un cliente envía al servidor siempre deben tomarse como una fuente de un ataque, es de suma importancia que se validen datos antes de realizar algún proceso con estos, de manera que no exista la posibilidad de un cambio de contexto, que obligue a realizar acciones perjudiciales, sin conocimiento.

3. La manera más confiable de asegurar los datos que son enviados de un cliente a un servidor es usando un modelo de prevención. El modelo recomendado es el modelo de lista blanca, que se enfoca en establecer reglas para los elementos que están permitidos, de manera que se ignoren los que no cumplan con las reglas. De esta forma se asegura prevención incluso ante nuevas amenazas.

.

BIBLIOGRAFÍA

1. ALLEN, Scott. *Http Conciso: Mensajes HTTP.* [en línea]. <https://code.tutsplus.com/es/tutorials/http-succinctly-http-messages--net-33699>. [Consulta 06 de mayo de 2018].

2. ANGULO, Jesus. *¿QUÉ ES HTTP/2 Y QUE VENTAJAS TIENE SOBRE HTTP 1.1?* [en línea]. <https://somostechies.com/que-es-http2/#.Wzl2p6jibIX>. [Consulta: 08 de mayo de 2018].

3. BAARS, Nanne. *How to Perform XXE Injection Attacks*. [en línea]. <https://discotek.ca/downloads/deepdive-reports/webgoat-7.1/archives/webgoat-container-7.1.war/plugin_lessons/xxe-1.0.jar/plugin/XXE/lessonSolutions/en/XXE.html>. [Consulta: 21 de mayo de 2018].

4. BHATI, Narendra. *Attacking JSON Application: Pentesting JSON Application*. [en línea]. <http://www.websecgeeks.com/2015/10/attacking-json-application-pentesting.html>. [Consulta: 21 de mayo de 2018].

5. BOWNE, Sam. *CNIT 123 Project 14: WebGoat Intro*. [en línea]. <https://samsclass.info/124/proj11/proj14-123-WebGoatIntro.html>. [Consulta: 18 de abril de 2018].

6. CABACAS GARCÍA, Tomás. *A FONDO ¿Qué es un servidor proxy y por qué debería implementarlo en mi empresa?* [en línea]. <https://www.muycomputerpro.com/2014/01/07/que-es-un-servidor-proxy>. [Consulta 14 de mayo de 2018].

7. CANTO, Amir. *Códigos de estado de respuesta HTTP*. [en línea]. <https://developer.mozilla.org/es/docs/Web/HTTP/Status>. [Consulta: 7 de mayo de 2018].

8. CODIGO FACILITO. *Que es Git.* [en línea]. <https://codigofacilito.com/articulos/que-es-git>. [Consulta: 19 de abril de 2018].

9. DE LUZ, Sergio. *Caché web (servidor proxy): Qué es y cómo funciona*. [en línea]. <https://www.redeszone.net/2011/01/16/cache-web-servidor-proxy-que-es-y-como-funciona/>. [Consulta: 14 de mayo de 2018].

10. DOCKER, INC. *About Docker CE*. [en línea]. <https://docs.docker.com/install/>. [Consulta: 8 de mayo de 2018].

11. GONZALEZ, Sergio. *HTTP Messages*. [en línea]. <https://developer.mozilla.org/es/docs/Web/HTTP/Messages>. [Consulta: 14 de mayo de 2018].

12. HTTPbis Working Group. *Hypertext Transfer Protocol Version 2 (HTTP/2)*. [en línea]. <https://http2.github.io/http2-spec/#TLSUsage>. [Consulta: 06 de mayo de 2018].

13. Internet Engineering Task Force (IETF). *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. [en línea]. <https://tools.ietf.org/html/rfc7231#section-4.3>. [Consulta: 7 de mayo de 2018].

14. JOSHI, Sunil. *3 Ways That An Xxe Injection Attack Could Hit You Hard!* [en línea]. <https://www.we45.com/blog/3-ways-an-xxe-vulnerability-could-hit-you-hard>. [Consulta: 22 de mayo de 2018].

15. LÁZARO, Diego. *Petición HTTP.* [en línea]. <https://diego.com.es/peticion-http>. [Consulta: 14 de mayo de 2018].

16. MIRÓ, Albert. *GitHub*. [en línea]. <https://www.deustoformacion.com/blog/programacion-diseno-web/que-es-para-que-sirve-github>. [Consulta: 20 de abril de 2018]

17. MITCHELL, Bradley. *Introduction to Proxy Servers in Computer Networking*. [en línea]. <https://www.lifewire.com/introduction-to-proxy-servers-computer-networking-816448>. [Consulta: 16 de mayo de 2018].

18. MUÑOZ, Alvaro. *ATAQUES BASADOS EN ENTIDADES XML*. [en línea]. <http://www.securitybydefault.com/2013/09/ataques-basados-en-entidades-xml.html>. [Consulta: 21 de mayo de 2018].

19. MUSCAT, Ian. *What is XML External Entity (XXE)? Part 1*. [en línea]. <https://www.acunetix.com/blog/articles/xml-external-entity-xxe-vulnerabilities/>. [Consulta: 21 de mayo de 2018].

20. NARVAEZ, Daniela. *Generalidades del protocolo HTTP*. [en línea]. <https://developer.mozilla.org/es/docs/Web/HTTP/Overview>. [Consulta: 20 de abril de 2018].

21. NETWORK WORKING GROUP. *Hypertext Transfer Protocol -- HTTP/1.1*. [en línea]. <https://tools.ietf.org/html/rfc2616>. [Consulta: 07 de mayo de 2018].

22. OLMEDO, Javier. *XXE (Xml eXternal Entity) – "El Nuevo SQLi"*. [en línea]. <https://hackpuntes.com/xxe-xml-external-entity-el-nuevo-sqli/>. [Consulta: 20 de mayo de 2018].

23. OWASP Foundation. *WebGoat Wiki.* [en línea]. <https://github.com/WebGoat/WebGoat/wiki.>. [Consulta: 18 de abril de 2018].

24. OWASP. *Proyecto WebScarab OWASP*. [en línea]. <https://www.owasp.org/index.php/Proyecto_WebScarab_OWASP>. [Consulta: 20 de abril de 2018].

25. OWASP. *SQL Injection Prevention Cheat Sheet*. [en línea]. <https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet>. [Consulta: 20 de mayo de 2018].

26. PÉREZ, Ignacio. *Comprendiendo la vulnerabilidad XSS (Cross-site Scripting) en sitios web*. [en línea]. <https://www.welivesecurity.com/la-es/2015/04/29/vulnerabilidad-xss-cross-site-scripting-sitios-web/>. [Consulta: 28 de mayo de 2018].

27. PINTOR, Mikel. *Vagrant, la herramienta para crear entornos de desarrollo reproducibles*. [en línea]. <http://www.conasa.es/blog/vagrant-la-herramienta-para-crear-entornos-de-desarrollo-reproducibles/>. [Consulta: 20 de abril de 2018].

28. PORTELLA, Jorge. *Videos y guía para WebGoat y WebScarab*. [en línea]. <https://jorgeportella.wordpress.com/ingenieria-de-software/proyecto-de-seguridad-de-aplicaciones-web-owasp/videos-guia-para-webgoat-y-webscarab/>. [Consulta: 19 de abril de 2018].

29. PORTILLA, Christian. *Crear comentarios XML*. [en línea]. <http://lineadecodigo.com/xml/crear-comentarios-xml/>. [Consulta: 22 de mayo de 2018].

30. PUENTES, Andres. *Métodos de petición HTTP*. [en línea]. <https://developer.mozilla.org/es/docs/Web/HTTP/Methods>. [Consulta 05 de mayo de 2018].

31. RAINBOWCRACK. *RainbowCrack*. [en línea]. <http://project-rainbowcrack.com/>. [Consulta: 18 de mayo de 2018].

32. REINO ROMERO, Alfredo. *Declaración de entidades*. [en línea]. <https://desarrolloweb.com/articulos/declaraciones-entidades-xml.html>. [Consulta: 26 de mayo de 2018].

33. ROUSE, Margaret. *Servidor Proxy*. [en línea]. <https://searchdatacenter.techtarget.com/es/definicion/Servidor-Proxy>. [Consulta: 16 de mayo de 2018].

34. SILVA, Fernando. *Buenas prácticas y características clave de HTTP/2.* [en línea]. <https://blog.ida.cl/desarrollo/buenas-practicas-caracteristicas-http2/>. [Consulta: 08 de mayo de 2018].

35. TAMEESH, Faisal. *Exploitation: XML External Entity (XXE) Injection*. [en línea]. <https://depthsecurity.com/blog/exploitation-xml-external-entity-xxe-injection>. [Consulta: 21 de mayo de 2018].

36. UNDERC0DER. *Las 5 Mejores herramientas Analizadoras de red y Sniffers!* [en línea]. <https://underc0de.org/foro/pentest/las-5-mejores-herramientas-analizadoras-de-red-y-sniffers!/>. [Consulta: 17 de mayo de 2018].

37. VAN KESTEREN, Anne. *List of HTTP methods (verbs).* [en línea]. <https://annevankesteren.nl/2007/10/http-methods>. [Consulta: 05 de mayo de 2018].

38. VELASCO, Ruben. *Captura paquetes de red sin instalar ninguna aplicación como Wireshark*. [en línea]. <https://www.redeszone.net/2017/04/29/captura-paquetes-red-windows/>. [Consulta: 20 de abril de 2018].

39. VIALFA, Carlos. *El protocolo HTTP.* [en línea]. <https://es.ccm.net/contents/264-el-protocolo-http>. [Consulta: 18 de mayo de 2018].

ANEXOS

Anexo 1**. Instalación de Webgoat**

Existen varias maneras de instalar WebGoat, las cuales se listan a continuación:

- Instalación independiente.
- Instalación usando Docker.
- Instalación usando las fuentes.
- Instalación usando Vagrant.

- Instalación independiente

La instalación independiente implica en montar la aplicación virtual en la máquina de java, de manera que sea accedida por cualquier navegador.

Prerrequisitos:

- Máquina virtual de Java

Es importante recalcar que se debe contar con una versión de máquina virtual de Java compatible con la versión de WebGoat que se desea montar. Al momento de realizar este trabajo de investigación se encontraba la versión 8 de WebGoat, que es compatible con la versión 8 de Java.

Continuación anexo 1

Se descarga el archivo .jar de la última versión de WebGoat desde los repositorios de Github[18].

Una vez descargado el archivo se debe acceder a la carpeta por medio de la línea de comandos, donde está contenido el archivo .jar descargado, y posteriormente se ejecutara la siguiente instrucción.

- java -jar webgoat-server- << version >>. Jar

Esto iniciará la aplicación Web de WebGoat, en el puerto 8080 por defecto, si se desea iniciar la aplicación en un puerto en específico y una dirección específica se puede usar la siguiente instrucción:

- java -jar webgoat-server- << version >>. jar --server.port = xxxx --server.address = xxxx

- Instalación usando Docker

Docker es una aplicación que permite crear contenedores ligeros y portátiles que permitan ejecutar una aplicación en cualquier dispositivo que cuente con Docker instalado, independiente del sistema operativo instalado y la versión.

Al momento de crear un contenedor para Docker, se configura de manera que tenga todos los complementos necesarios para ejecutar la aplicación. En este caso se configura para contar con la máquina virtual de Java utilizada en la ejecución de la aplicación WebGoat.

Prerrequisitos:

[18] Github. https://github.com/WebGoat/WebGoat/releases. Consulta: 28 de abril de 2018.

Continuación anexo 1

- Docker (https://docs.docker.com/install/#server)

Los desarrolladores de WebGoat, colocan, cada cierto tiempo (aleatorio) una vista previa de la aplicación en Docker Hub[19].

Una vez instalado Docker, abrir una ventana de comando y ejecutar las siguientes instrucciones:

- Docker pull webgoat / webgoat-8.0.
- docker run -p 8080: 8080 -it webgoat / webgoat-8.0 /home/webgoat/start.sh.

Posterior a que se inicie el contendor de Docker, se debe ejecutar el siguiente comando, que mostrara si realmente se está corriendo la aplicación:

- docker ps.

Luego de asegurar que aplicación esté funcionando, se debe verificar la dirección sobre la cual está funcionando, a continuación se brindan las posibles formas de lograrlo:

- Si se usa un docker- machine, se utiliza el comando: docker-machine env.
- Si se usa un boot2docker, se utiliza el comando: docker network inspect bridge.
- De lo contrario, se debe utilizar localhost.

Con la información de la ip y el puerto sobre los cuales se montó la aplicación, se puede acceder agregándole a la ruta /WebGoat[20].

[19] Docker. https://hub.docker.com/r/webgoat/webgoat-8.0/. Consulta: 29 de abril de 2018.
[20] Ejemplo de uso: http://localhost:8080/WebGoat.

Continuación anexo 1

- Instalación usando las fuentes

Git es un software de control de versiones diseñado por Linus Torvalds. Este software permite gestionar los cambios que se realizan sobre algún elemento de software que se encuentra en proceso de desarrollo. Una de las facilidades que proporciona Git es el manejo de versiones del código que posee un proyecto.

GitHub es una plataforma de desarrollo colaborativo de software, que permite alojar el código fuente de un proyecto utilizando Git. De manera que el proyecto puede ser accedido y modificado por cualquier persona que tenga interés en el mismo y luego proponer la mejora para el dueño del proyecto.

Los desarrolladores del proyecto WebGoat, alojan el código fuente de la aplicación en GitHub, de manera que permite ser descargado por cualquier persona para el uso y modificación del mismo.

Prerrequisitos:

- Git (https://help.github.com/articles/set-up-git/).
- Java versión 8 instalado.
- Maven superior a la versión 3.2.1.
- IDE para Java (por ejemplo: Netbeans, Eclipse).

Luego de tener los prerrequisitos se abre una ventana de comando y se ejecuta la siguiente instrucción, para obtener todas las fuentes del proyecto:

- git clone git@github.com: WebGoat / WebGoat.git.

Continuación anexo 1

Luego de obtener las fuentes se debe compilar el proyecto, usando las siguientes instrucciones:

- cd WebGoat.
- git checkout << branch_name >>.
- mvn instalación limpia.

Seguido a la compilación de las fuentes, ya es posible ejecutar el proyecto de WebGoat. La configuración predeterminada es que se ejecute sobre el puerto 8080, para cambiar el puerto en el que se ejecutara la aplicación se busca la siguiente ruta:

- / webgoat-container / src / main / resources / application.properties

En el archivo se puede realizar el cambio para la dirección ip y el puerto en el que se desea montar la aplicación.

- Instalación usando Vagrant

Vagrant es una herramienta de línea de comandos que permite generar entornos de desarrollo reproducibles, para compartir de manera sencilla. Crea y configura máquinas virtuales mediante ficheros de configuración. Esta herramienta está disponible en los siguientes sistemas operativos: Windows, MacOs y GNU/ Linux.

Para compartir el entorno basta con proporcionar el archivo de configuración de Vagrant, de manera que otra persona reproduzca el entorno de desarrollo utilizando un comando.

Continuación anexo 1

Los creadores del proyecto WebGoat proporcionan un entorno de desarrollo completo utilizando Vagrant.

Prerrequisitos:

- Vagrant.
- Virtualbox.

Abrir el programa Vagrant y colocar las siguientes instrucciones:

- cd WebGoat/webgoat-images/vagrant-training
- vagrant up

Una vez terminado el proceso, se debe iniciar sesión en la máquina virtual con el usario vagrant y contraseña vagrant. El código fuente estará disponible en el directorio principal.

Fuente: https://github.com/WebGoat/WebGoat. Consulta: mayo de 2018.