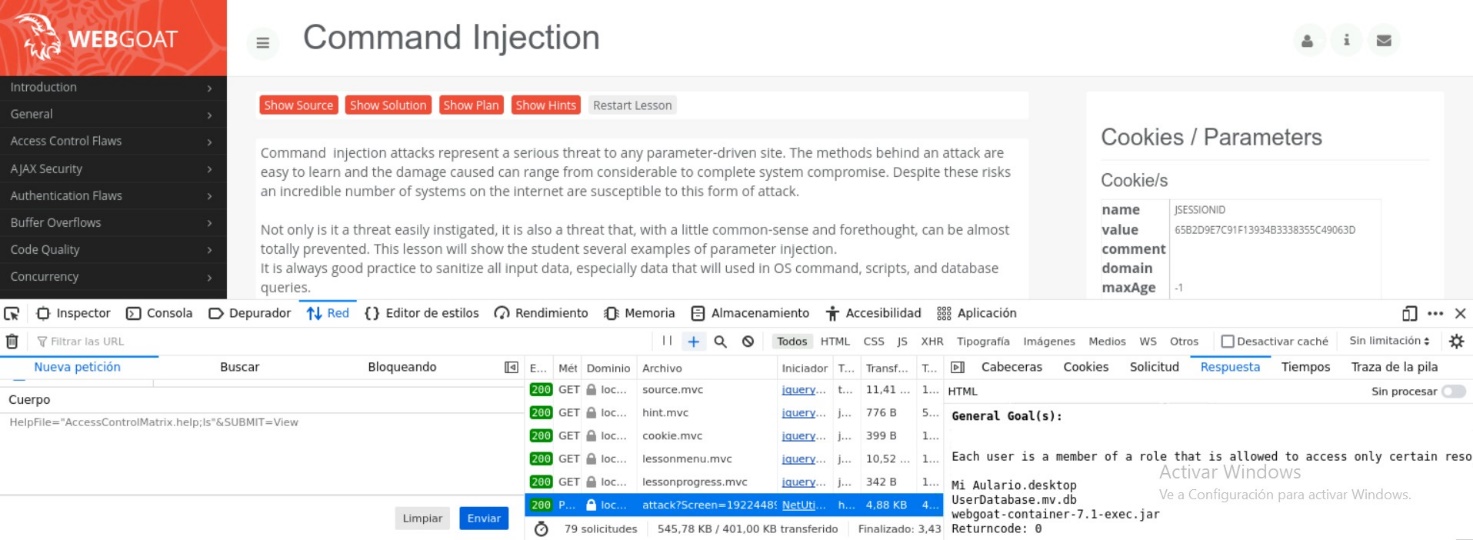
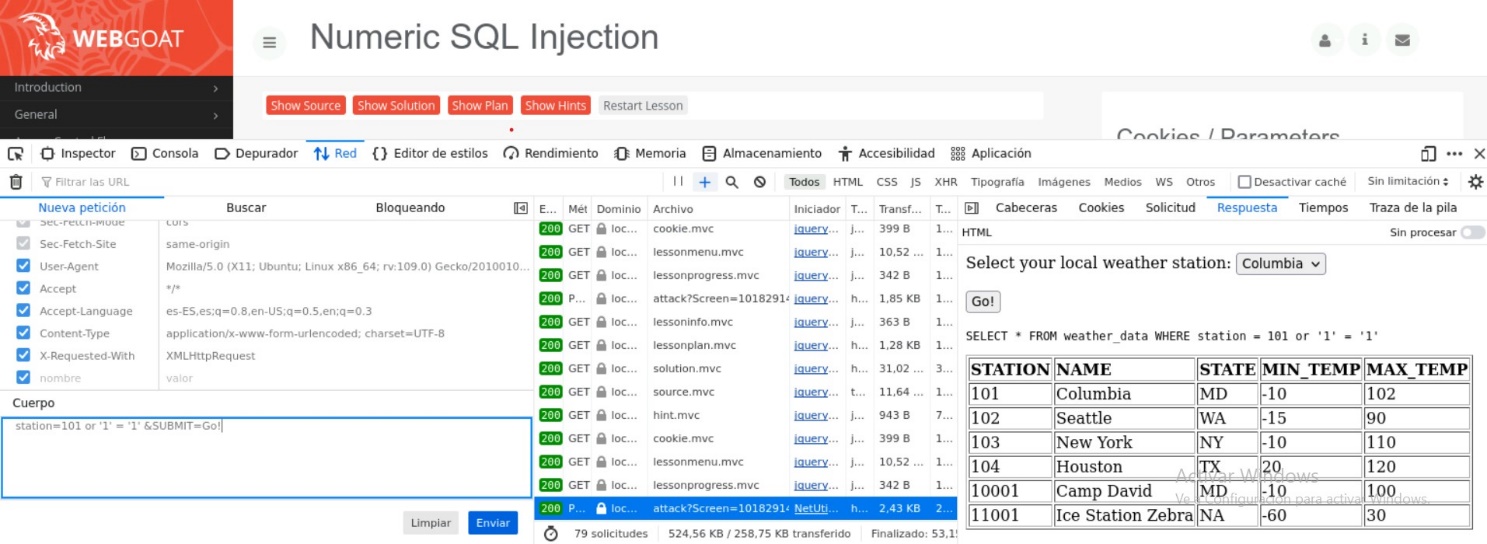
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Nombre Ataque Webgoat | 1.Explicación ataque | 2.Fácil/difícil | 3.Código vulnerable | 4.Consecuencias | 5.Impacto Bajo/Alto |
| Num Sql Inj. | Seleccionas el botón Go con cualquier estación predeterminada. De esta manera, al inspeccionar en la red, se puede visualizar el paquete que vamos a reenviar (último paquete attack). Modificamos el cuerpo del paquete e insertamos “station=101 or ‘1’=’1’&SUBMIT=Go!  “. A través del “or” y del “1=1”, conseguiremos que el “where” se convierta en true al ser el “or” una unión. La sentencia SQL se convertirá en “SELECT \* from weather\_data”, mostrando toda la tabla. Por último, enviamos el paquete y vemos en el servidor la respuesta. | El ataque ha sido fácil debido a que con introducir en station la sentencia y reenviar el paquete, era suficientemente para hacer el injection SQL | En el método injectableQuery, se añade directamente al statement la estación. Por lo tanto, no usa PreparedStatement para comprobar que no se añade una SQL Injection ni se valida la entrada de alguna manera | La consecuencia es que se muestra todos los datos de todas las columnas de la tabla weather\_data | El impacto es muy alto debido que al hacer una injection sql simple, consigues los datos de toda la tabla |
| Comm Inj. | Seleccionas el botón View para que, al inspeccionar la red, se pueda visualizar el paquete attack que se va a reenviar. Modificas el cuerpo del paquete insertando “helpfile=”AccessControlMatrix.help;ls”&SUBMIT=VIEW”. De esta manera, se introduce el comando ls dentro del helpfile para que se ejecute también, y podamos ver los archivos del directorio | El ataque ha sido fácil debido a que solo con introducir el comando, se ejecutará directamente | En el segundo método exec, se pasa como argumento un array de comandos, en el que el tercer argumento es el command injection. Con el método execSimple, se ejecutan todos los comandos, aunque se suponga que no se “puedan” introducir | Las consecuencias son que puedes introducir comandos que muestren información protegida y que no deba conocer el usuario | El impacto es intermedio dependiendo del comando que puedas introducir, y el tipo de información al que puedas acceder |
| Log Spoof | Seleccionas el botón View para que, al inspeccionar la red, se pueda visualizar el paquete attack que se va a reenviar. Modificas el cuerpo del paquete insertando “username=Smith%0d%0aLogin Succeeded for username: admin<script>alert(document.cookie)</script> &password=guest &SUBMIT=Login. A continuación, reenviamos el paquete. De esta manera, insertamos en el log la url y se muestra al hacer el login que se ha introducido correctamente. | Intermedio. Se considera el ataque intermedio ya que tienes que introducir en formato url el username. | if (inputUsername.length()! = 0). En la anterior comprobación, únicamente comprueba que el username sea de tamaño positivo. Si el tamaño es mayor que 0, procede a validar y decodificar la url. | Las consecuencias son que puedas introducir injections sql en el log del programa, y así descubras o introduzcas nuevos usuarios que no deberían estar. | Alto. Tiene un impacto alto debido a que el log tiene que ser una de las partes más seguras del programa. |
| Hidden fields | Seleccionamos el botón UpdateCart. Inspeccionamos para ver el paquete attack correspondiente (el último paquete attack). En el cuerpo del paquete, modificamos el price para ponerlo a una cantidad menor correspondiente a la que se debería de pagar (en mi caso 0). Reenviamos el paquete, y se realizaría la compra con el precio que se haya establecido anteriormente. | El ataque ha sido sencillo porque con modificar el atributo precio en el paquete attack, era suficiente para poder pagar por la compra | La validación del precio se comprueba en el comienzo de la clase a través de la variable pattern y script. En cambio, se debería realizar la validación de la entrada en el método createContent, para que se compruebe si es válida la entrada del usuario en el lado del servidor | Las consecuencias son que podrías realizar una compra que costase una gran cantidad de dinero, y que únicamente tengas que pagar una cantidad de dinero irrisoria por ella. | El impacto es muy alto, porque en este caso, una compra de altas cantidades podría salir gratis. Con lo cual, la empresa acarrearía enormes pérdidas y problemas. |
| Buffer Overflow | Seleccionamos el botón submit para visualizar el modelo del paquete attack que vamos a reenviar. En segundo lugar, copiamos más de 4096 caracteres en el parámetro 3 (utilizamos el carácter A), que en este caso es el Room Number. Modificamos dicho argumento en el cuerpo del paquete “first\_name=&last\_name=&room\_no=A……..A”, y reenviamos el paquete. A continuación, seleccionamos el nuevo paquete attack que se envía y aceptamos los términos en la siguiente pantalla. Seleccionamos el nuevo paquete attack, y en respuesta, seleccionamos sin procesar. Se podrá visualizar todos los datos en “value” ordenados. Basta con volver a la página principal con los datos de uno de los usuarios para poder introducirte en el sistema del hotel | El ataque ha sido difícil porque se requiere primero en analizar si es posible un buffer overflow, y en ese caso, a partir de cuantos valores. Además, se necesita visualizar y modificar varios paquetes | En el método makeThirdStep, se encuentra la siguiente comprobación “if(param3.length() > 4096)”. Si se cumple la condición, se añaden todos los datos de los usuarios. Por lo tanto, se podrá utilizar cada uno de los datos para registrarse | Las consecuencias son que se puede registrar con cada uno de los usuarios, pudiendo hacer cualquier tipo de acción en la página del hotel | El impacto es muy alto ya que con los datos de los usuarios se podrá acceder a cualquier parte del hotel, y se podría cancelar o realizar reservas, gastar dinero… |
| Thread Safety | Seleccionamos la tarea Thread Safety Problems. Abrimos otra misma pestaña copiando la url. Como tenemos dos usuarios (jeff y daves), introducimos un nombre en cada pestaña. Seleccionamos el botón submit a la vez en ambas pestañas, y se mostrará la información del registro de uno de los dos usuarios en las dos pestañas (la información del segundo usuario). | El ataque ha sido muy fácil porque con abrir una nueva pestaña e introducir los dos usuarios, era suficiente para conseguir la información de registro del otro usuario | En el método createContent, se realiza una comprobación para ver si se está con el usuario del momento. El problema está en que antes de seleccionar la información de la base de datos, se realiza un sleep de 1.5 segundos. En ese tiempo, otro usuario podría intentar acceder a su respectiva información de registro. Con lo cual, el nuevo usuario será el usuario del momento, y el antiguo usuario al terminar el sleep obtendrá la información del nuevo usuario. | Las consecuencias son que, entrando con tu registro, consigues la información de registro de otro usuario con el que podrás acceder de ahora en adelante | El impacto es muy alto debido a que con la información de registro se podrá realizar cualquier tipo de acción con riesgo en la aplicación |
| ZipBomb | Creamos un archivo a través de Python que ocupe más de 20 mb. Para ello, simplemente escribimos múltiples espacios o caracteres en el archivo txt. Comprimimos el zip, y lo añadimos a la tarea para subirlo | El ataque ha sido fácil porque con crear un zip de determinado tamaño (máximo 2mb), y con cuyos archivos de su interior ocupen más de 20 mb, era necesario para realizar el ataque | En el método handleRequest, se realizan dos comprobaciones con los tamaños de 20 y 2 mb. En primer lugar, se valida que el archivo ocupe menos de 2mb (“if (item.getSize() < 2000 \* 1024”). A continuación, si los archivos del interior ocupan más de 20 mb (“if (total > 20 \* 1024 \* 1024”), se añaden a la webSesion y se crean mensajes | Las consecuencias son que puedes saturar o hacer que el programa se quede sin memoria porque no está preparado para una entrada de tal tamaño. Esto podría bloquear la aplicación y que no se pudiera utilizar ante tal carga de memoria | El impacto es alto debido a que con un introducir un zip en la aplicación, podrías conseguir que se sature toda la aplicación, y que los usuarios no pudieran utilizarla |
| Malicious Execution | Creamos un archivo jsp e introducimos un código cualquiera para que se ejecute. En mi caso, he creado el archivo jsp en netbeans. He introducido en el interior el código que proporciona webgoat para crear el file en el directorio establecido. A continuación, seleccionamos el botón browse y el archivo jsp. Para finalizar, seleccionamos Start Upload para subir el archivo | El ataque ha sido fácil porque se podía introducir un código cualquiera dentro del archivo jsp, y con subir el archivo era suficiente para realizar el ataque | String uploaded\_file\_path =uploads\_and\_target\_parent\_directory + UPLOADS\_RELATIVE\_PATH  + java.io.File.separator + item.getName();  File uploadedFile = new File(uploaded\_file\_path); En el siguiente código, se puede comprobar cómo no se verifica si la extensión es segura. De esta manera, puedes introducir un archivo malicioso. | Las consecuencias son que puedas ejecutar cualquier tipo de código peligroso dentro del archivo jsp. En el ejemplo, se crea un file llamado guest.txt en el directorio establecido dentro del Escritorio. | El impacto es alto debido a que puedes ejecutar código que tenga mucho riesgo, y el grado de posibilidad de ataques es enorme porque depende en su totalidad del código que introduzcas |

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre Ataque Webgoat | 6.CWE correspondiente | 7.Ataque CWE dificultad contraria | 8.Consecuencia CWE contrarias |
| Num Sql Inj. | - CWE-20: Improper Input Validation (no se valida la entrada) | Un posible ataque complicado sería que en el código hubiera mucha validación de entrada y se tuviera que analizar cada una de las partes para ir poco a poco descubriendo vulnerabilidades | Las posibles consecuencias contrarias serían que la tabla mostrada fuera de nula utilidad al conocer con anterioridad todos esos datos |
| Comm Inj. | - CWE-89: Improper Neutralization of Special Elements used in an-SQL Command ('SQL Injection') | Un posible ataque complicado sería en el caso de ser comprobados la mayoría de los comandos que no se pueden introducir, y haga falta ir comprobando cuáles no están siendo prevenidos | Las posibles consecuencias contrarias serían que se introdujera un comando de baja utilidad que no proporcione ningún riesgo |
| Log Spoof | CWE-117: Improper Output Neutralization for Logs | Un posible ataque más simple sería que no hiciera falta ni codificar el username a url y que únicamente se compruebe que el tamaño sea positivo | Las posibles consecuencias contrarias serían que no se pueda dañar los archivos logs |
| Hidden fields | - CWE-89: Improper Neutralization of Special Elements used in an-SQL Command ('SQL Injection') | Un posible ataque más complicado sería solo se pudiera modificar el precio de algunos productos, y se tuviera que ir analizando en cuales de los productos se puede bajar el precio. | Las posibles consecuencias contrarias serían que no se pudieran prácticamente modificar los precios de los artículos, o solo algunos de precio y cantidad pequeña. |
| Buffer Overflow | - CWE-120: Classic Buffer Overflow | Un posible ataque más simple sería el no tener que estar visualizando varios paquetes attack, debido a que con modificar uno solo de ellos es suficiente. | Las posibles consecuencias contrarias serían que, en vez de conseguir los datos de los usuarios, se pudieran conseguir datos mucho menos irrelevantes para el sistema |
| Thread Safety | - CWE-307: Improper Restriction of Excessive Authentication Attempts | Un posible ataque más complicado sería que hicieran falta crear múltiples hilos y no solo dos como en este caso, y que al hacer ello, se requiriera de mucha memoria | Las posibles consecuencias contrarias serían que se obtuviera un tipo de información carente de valor en vez de la información de registro de un usuario |
| ZipBomb | -CWE-434: Unrestricted Upload of File with Dangerous Type | Un posible ataque más complicado sería que se restringiera el tamaño mucho más, o que se permitiera una entrada de muchísimo tamaño haciendo que sea muy difícil crear un archivo de tanto tamaño | Las posibles consecuencias contrarias serían que el tamaño del zip que se introdujera no consiguiera saturaciones haciendo que tenga un riesgo mucho menor |
| Malicious Execution | -CWE-434: Unrestricted Upload of File with Dangerous Type | Un posible ataque más complicado sería que validara la entrada, pero de forma incorrecta, dejando algún fallo en algún lado. En ese caso, se debería ver qué tipo de ataque no está siendo comprobado | Las posibles consecuencias contrarias serían que el código introducido no tuviera prácticamente ningún tipo de vulnerabilidad y no pusiera en riesgo a la aplicación |



Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Sitio web

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Sitio web

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Teams

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente