Desafío Pentest
Prometeo Open Banking

Diego Moraes

Contexto

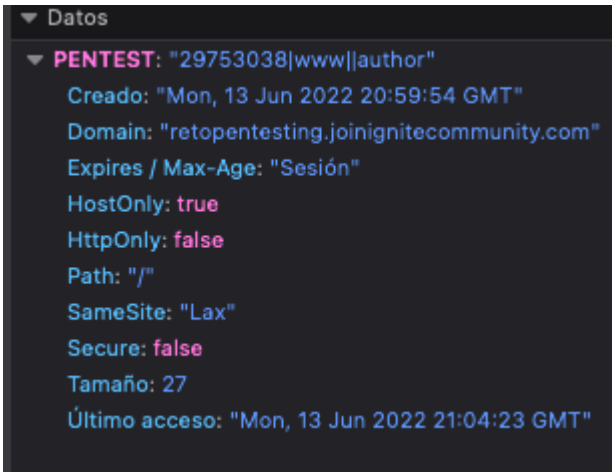
Prometo Open Banking disponibilizo una aplicación web la cual cuenta con vulnerabilidades para que entusiastas puedan encontrar brechas de seguridad y armar un reporte sobre las mismas. En este documento se dejan por escrito cuáles fueron las fallas encontradas.

Análisis

Como primer paso se comenzó utilizando la plataforma para conocer un poco más de su funcionamiento, la misma cuenta con una landing page en la que se muestra snippets que pueden cargar los usuario que se registren a la web. La aplicación cuenta con varios formularios, los cuales son grandes candidatos para probar posibles vulnerabilidades, más todavía en aquellos que aceptan como input HTML.

1 - **Creación de usuario**: El formulario de registro es básico, solamente se pide username y password pero no se valida un formato de contraseña para que sea seguro. Este es un punto de mejora para la aplicación, ya que se pueden crear usuario con los caracteres que sean lo cual facilita a que una atacante pueda iniciar sesión utilizando fuerza bruta, tampoco hay un chequeo con un recaptcha para contrarrestar a bots que están intentando este tipo de ataque.

2- **Manejo de sesiones:** Analizando cómo se maneja la sesion de los usuario se detecta de que se guarda una cookie



Si analizamos el valor de esta cookie, la misma está compuesta por tres strings separados por un pipe y sin ningún tipo de encriptación, donde el primer valor es un identificador y el segundo string es el nombre del usuario logueado. Si nos guardamos el valor de esta cookie podemos acceder sin problemas a la sesión, la cookie puede ser robada fácilmente a través

de una inyección de código Js en la web ya que el primer valor nunca cambia, parece ser un identificador del usuario.
Para probar que el robo de la cookie si funciona podemos crearnos dos usuario diferentes, en mi caso utilice el usuario **aaa** y **www**.

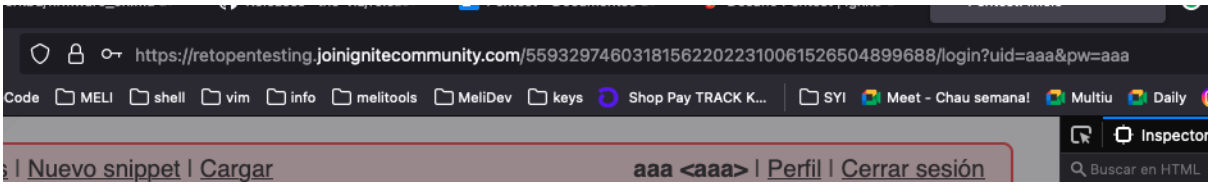
Pasos para reproducirlo:

1. Iniciar sesión con el user aaa
2. Guardar el valor de la cookie (85564940|aaa||author)
3. Salir de la sesión
4. Iniciar sesión con el user www
5. Modificar el valor de la cookie PENTEST por el guardado anteriormente
6. Refrescar la página y ver como ahora estamos en la sesión del usuario aaa.

Esto es posible porque la cookie tiene seteado el valor HttpOnly en false, lo cual permite que sea seteada sin utilizar el protocolo HTTP a través del header Set-Cookie.

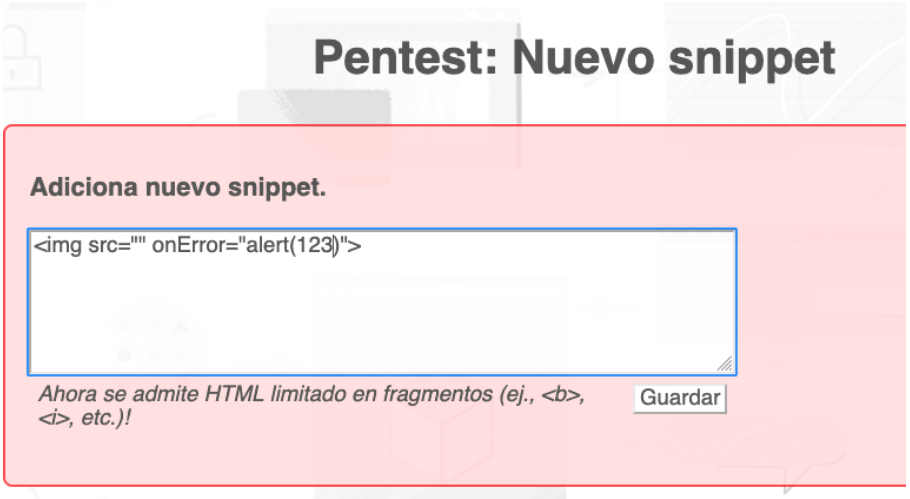
Este tipo de vulnerabilidad puede ser catalogada por el tipo **Session Hijacking**.

Además, luego del inicio de sesión podemos ver el user y la pass del user en la url

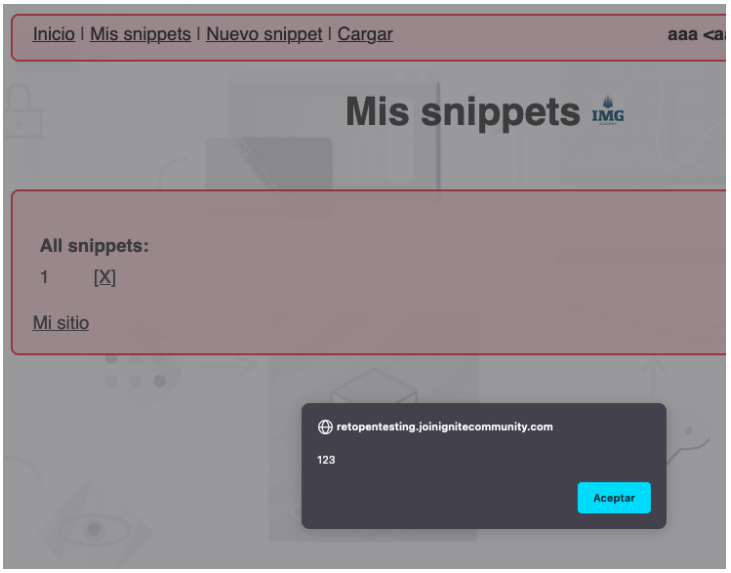


2- **Creación de nuevo snippet**

El formulario de creación de nuevo snippet permite escribir HTML pero no cuenta con ninguna validación o control sobre el código HTML que se escribe, lidiar con HTML escrito por el usuario puede ser peligroso ya que se puede inyectar código Javascript a través de él. Conociendo un poco como funcionan las etiquetas HTML podemos explotar este form inyectando código JS, para este caso utilice la etiqueta img, la cual cuenta con un parámetro que recibe código JS que se ejecuta en caso de que la imagen no cargue correctamente.



Registrando un snippet de esta forma, este quedará guardado y como no tiene ningún source especificado para cargar la imagen se ejecutará el código que va como parámetro en onError. Al cargar el listado de snippets, el alert se ejecuta sin ningún problema.

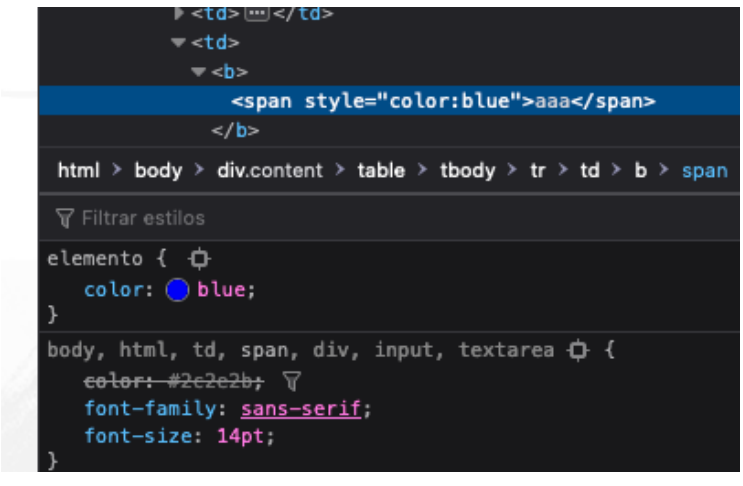


Como todos los usuario de la plataforma están habilitados para ver los snippets de los demás, podemos ejecutar código JS de lado de cualquier usuario que esté utilizando la plataforma. Esta vulnerabilidad es catalogada como **XSS**, lo cual combinado con Session Hijacking puede ser muy peligroso. Como contingencia de esto recomiendo utilizar alguna librería como WYSIWYG la cual tiene controles de estos posibles ataques.

3. **Edición de perfil**

El siguiente formulario permite editar el perfil del usuario, en esta prueba analice el campo de Color de Perfil, el cual hace que el color del usuario se peuda ver de otro color.
Si ponemos blue, el nombre de perfil lo podemos ver en color azul

¿Cómo funciona esto? Viendo los estilos de este elemento se puede ver que blue, se está seteando a través de la propiedad color de CSS



Esto significa que también tenemos control sobre el CSS de la página, a través de css se pueden encontrar alguna vulnerabilidades, también inyectando código o cargado urls de otro sitio usando la función url(). **Esta vulnerabilidad no llegue a explotarla.**

4. **Refresh de la página:** La landing page provee un link para poder refrescar la lista de snippets que están cargados, la cual refresca la pagina actual. Si analizamos el código de cómo se hace este reload, podemos identificar que se esta haciendo uso de la función ***eval*** de javascript la cual sirve para ejecutar código desde cualquier string que se le pase a la función, con un ataque man in the middle se podría interceptar la request y cambiar el texto de la response para ejecutar el código que nosotros queramos. **Esto no llegue a explotarlo.**

```
/**
 * Refreshes the current page by loading the url and then passing the
 * json response object to the callback.
 */
function _refresh(url, callback) {
  var httpRequest = window.XMLHttpRequest ? new XMLHttpRequest()
      : new ActiveXObject("Microsoft.XMLHTTP");
  httpRequest.onreadystatechange = function() {
    if (httpRequest.readyState == 4) {
      _feed = callback;
      eval('(' + httpRequest.responseText + ')');
      httpRequest = null;
    }
  }
  httpRequest.open("GET", url, true);
  httpRequest.send(null);
```

5. **Análisis de posible inyección SQL:** Realice pruebas con la herramienta sqlmap para poder detectar algún tipo de brecha en la base de datos y poder obtener datos de la misma. Se realizaron pruebas heurísticas pero no logre ejecutar codigo sql, el unico detalle es que todas las consultas quedaron registradas como entries en la base de datos y se puede registrar lo que sea con estos caracteres. Esto significa que los strings están llegando a la base de datos, quizá con algún tamper o combinación de escapes específicos se pueda lograr obtener datos de la DB o incluso una reverse shell.

6. **Headers**
Los header de las request muestran cual es la version del servidor web que se esta usando, en este caso es un nginx versión 1.14.1 la cual cuenta con vulnerabilidades conocidas, podemos verlas aqui