UNIVERSIDAD NACIONAL DE LA MATANZA

Departamento de Ingeniería e Investigaciones Tecnológicas

**Seguridad y Calidad en Aplicaciones Web**

Referente de Cátedra: Walter R. Ureta

Plantel Docente: Pablo Pomar, Walter R. Ureta

Alumnos

* Baldo Ignacio Uriel
* González Diego Gastón
* Peralta Christian
* Núñez Tomas Agustín

# Tecnologías utilizadas

* Eclipse Java EE IDE for Web Developers V. Neon.3 Release (4.6.3)
* Apache Tomcat 7.0.75/9.0
* Java 8
* HSQLB
* Maven
* Hibernate

# Entrega 1 – 13 de Mayo

Historias de usuario

## ● Como usuario quiero registrarme en la aplicación.

## ● Como usuario quiero poder guardar un texto con mi cuenta

## personal.

## ● Como usuario quiero poder cambiar mi contraseña.

## ● Como usuario quiero poder recuperar mi contraseña.

## ● Como administrador quiero poder deshabilitar o habilitar a los

## usuarios.

## ● Como usuario quiero conocer el historial de actividad de mi

## cuenta.

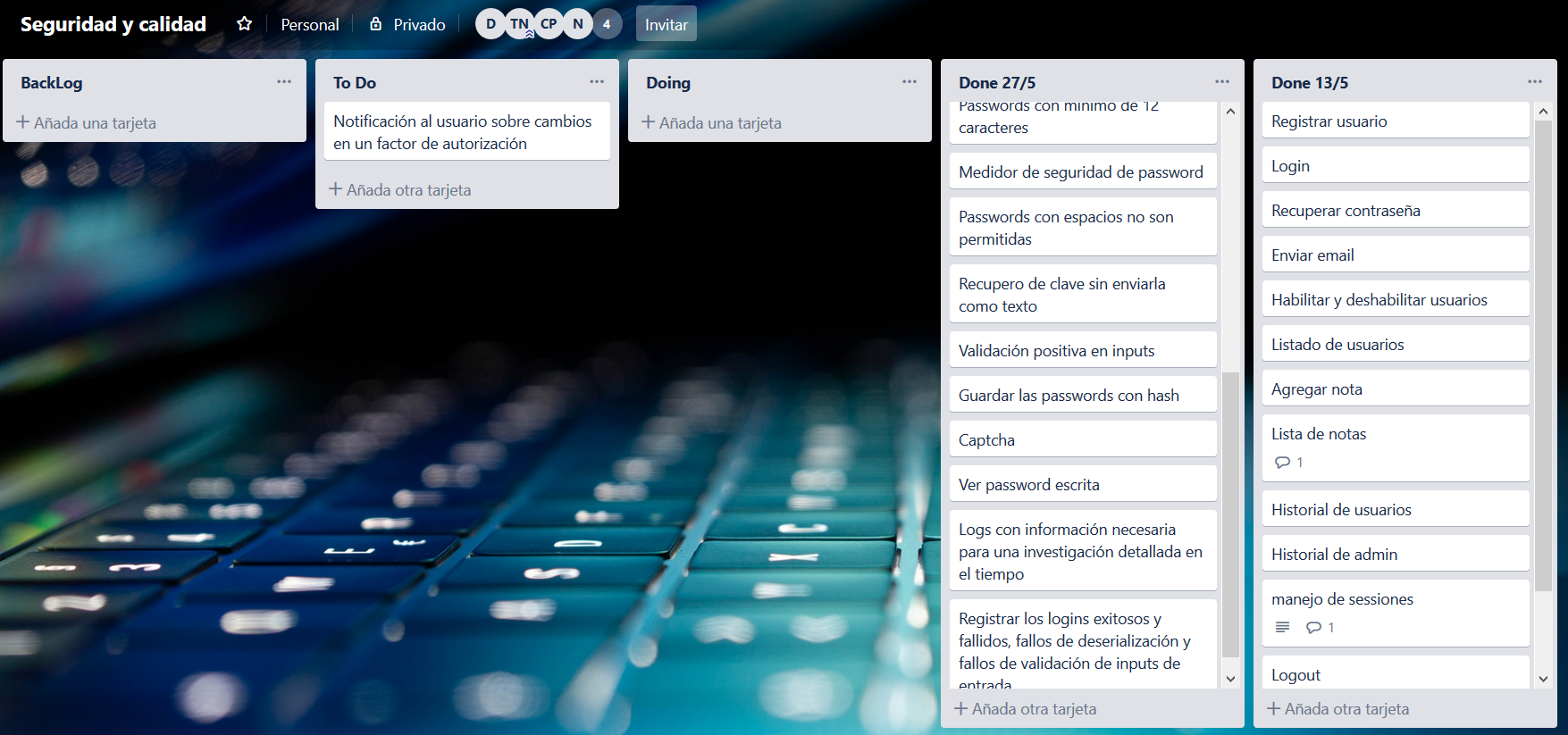
## ● Como administrador quiero conocer el historial de actividad de

## cada usuario.

Paso 1

Se decide crear un sitio partiendo de un maquetado utilizando Bootstrap.

Para dar seguimiento al avance del proyecto, se crea un dashboard en Trello (<https://trello.com/b/9hjqwobv/seguridad-y-calidad>)



Paso 2:

Se modelan las entidades necesarias para persistir en base de datos lo requerido en el enunciado.

Se crean las siguientes entidades:

Usuario: persiste los datos de las cuentas de usuario, tales como Email, Nombre, Password, Rol, Habilitado, FechaAlta, FechaModificacion

Nota: persiste las notas creadas por los usuarios.

Log: lleva un registro de la actividad de los usuarios mientras perduran en el sitio.

Funcionalidad: Tiene las distintas pantallas que pertenecen a la aplicación. De esta manera se podrá asociar cada registro de log a una pantalla y funcionalidad dentro de cada pantalla de forma particular.

Paso 3

Se creó un registro de usuario, un login, y se armaron visualizaciones para los 2 tipos de acceso solicitados. Admin y Usuario, con los accesos individuales para cada uno.

Paso 4

* Como el enunciado no requería encriptación, se decide guardar las contraseñas en texto plano.
* Se creó la funcionalidad de recuperación de claves, enviando por correo las claves en texto plano, dado que el enunciado no especificaba ningún nivel de seguridad como requisito mínimo.

# Entrega 2 – 27 de Mayo

## Implementar en la aplicación asignada los siguientes puntos del

## Application Security Verification Standard 4.0 (ASVS) para el Nivel

## dos (2):

## ● V2: Authentication Verification Requirements (V2.1.1-3, V2.1.5-12,

## V2.2.1, V2.2.5, V2.4.1-5, V2.5.1-5. Total 23 puntos)

## ● V5: Validation, Sanitization and Encoding Verification Requirements

## (V5.1-5.5)

## ● V7: Error Handling and Logging Verification Requirements (V7.1-7.4)

Paso 1

Se decide analizar la documentación, y armar una lista simplificando lo requerido por cada punto, y así evaluar el nivel de impacto para llevar a cabo la implementación de cada uno, y analizar si aplica o no.

**V2**: Authentication Verification Requirements

V2.1.1: Passwords con mínimo de 12 caracteres

Análisis: Aplica

Plan de acción: Se implementa validación.

V2.1.2: Password con 64 o más caracteres son permitidas

Análisis: Aplica

Plan de acción: Se implementa validación.

V2.1.3: Passwords con espacios no son permitidas

Análisis: Aplica

Plan de acción: Se implementa validación con javascript.

V2.1.5: Passwords con caracteres ascii como emoticonos no son permitidas

Análisis: Aplica

Plan de acción: Se implementa validación con javascript.

V2.1.6: La funcionalidad de cambio de password requiere el usuario actual y la nueva password

Análisis: Aplica

Plan de acción: Se implementa validación con javascript.

V2.1.7: Chequear que la password ingresada no sea una de las comunes ya violadas, como 123456 o cosas por el estilo. Podemos chequearlo contra una lista interna nuestra o usar una api externa. Lo mejor y más fácil es que tengamos una lista interna, y ver que la password ingresada en el registro, o en el cambio de password no esté en esa lista.

Análisis: Aplica

Plan de acción: Se implementa una lista de contraseñas conocidas, y una búsqueda entre ellas para aceptar o rechazar la password ingresada.

V2.1.8: Tener un medidor de seguridad para verificar que la password sea fuerte. Esos que te obligan a agregar mayúsculas, minúsculas, caracteres especiales, y van pintando de rojo, amarillo y verde. Ver si hay algo ya hecho para copiar y pegar de algún lado.

Análisis: Aplica

Plan de acción: Se implementa un validador de password, indicando con colores y una leyenda si la contraseña ingresada es débil, media o fuerte.

V2.1.9: No tener reglas de composición de contraseñas con caracteres permitidos. Por ejemplo no limitar la cantidad de números, o de mayúsculas o minúsculas o caracteres especiales.

Análisis: Aplica

Plan de acción: Se implementa validación verificando calidad de passwords a través de cantidades de caracteres de distintos tipos.

V2.1.10: Que no haya requisitos de cambio de password periódico o impedimentos de contraseñas históricas.

Análisis: No aplica

Plan de acción: No requerimos cambios de password periódicos, y no tenemos forma de verificar históricos de passwords dado que usamos un algoritmo de encriptación no desencriptable.

V2.1.11: Verificar que funcione el “pegar”, los ayudantes de contraseña del navegador, y se permitan administradores externos de contraseñas.

Análisis:

Plan de acción:

V2.1.12: Verificar que el usuario pueda ver la contraseña escrita (con el ojito), o poder ver el ultimo carácter escrito antes que pase a ser un punto o asterisco.

Análisis: Aplica

Plan de acción: Agregamos la posibilidad de visualizar la password presionando en una imagen de un ojo, para que el usuario pueda verificar si ingresó la password correctamente.

V2.2.1: Verificar que haya controles de anti automatización. Esto es que intenten con fuerza bruta adivinar una password. Lo que habría que hacer acá es meter un captcha, e ir acumulando cuantos intentos de logueo fue haciendo erróneos, y si le pifió 5 veces seguidas por ejemplo, bloquearle la cuenta haciendo que no pueda loguearse hasta dentro de media hora por ejemplo. Lo otro mas complejo para hacer pero no se si vale la pena es bloquear la IP pero con lo de bloquear la cuenta para que no pueda loguearse en un tiempo alcanzaría.

Análisis: Aplica

Plan de acción: Implementamos un captcha, y bloqueos de cuentas de usuario luego de una cantidad fija de intentos de ingreso fallido.

V2.2.5: Verificar que cuando tengamos separados un proveedor de servicio de credenciales (CSP) y la verificación de la password de la aplicación, haya en el medio de los 2 endpoint un TLS. (Este no aplica)

Análisis: No aplica.

Plan de acción: No tenemos separados el sitio y la validación de passwords, por lo que no tenemos 2 endpoints distintos, como si usaramos un servicio web.

V2.4.1: Guardar las passwords con hash y salt.

Análisis: Aplica

Plan de acción: Implementamos el algoritmo PBKDF2, que requiere la especificación de un salt, para obtener el hash que guardaremos en la base de datos.

V2.4.2: El SALT que usemos, debe ser único por cada password que guardemos, y tiene que medir al menos 4 caracteres. Podemos tomar los primers 4 o últimos 4 caracteres dela password sin hashear y usar eso como salt. Es importante que se haya elegido un algoritmo de encriptación con salt, como por ejemplo AES256.

Análisis: Aplica

Plan de acción: Lo implementamos tomando los últimos 4 caracteres de la password ingresada por el usuario, y generando un hash con md5 para utilizar como salt al momento de invocar a la encriptación con PBKDF2.

V2.4.3: Verificar que si usamos el algoritmo PBKDF2, la cantidad de iteraciones para encriptar sea el máximo que permita el servidor, que suele ser al menos 100.000 iteraciones.

Análisis: Aplica

Plan de acción: Implementamos este algoritmo, por lo que COMPLETAR NACHO

V2.4.4: Verificar que si usamos el algoritmo bcrypt, el factor que pongamos sea lo que el servidor se aguante. Tipicamente es 13.

Análisis: NO aplica

Plan de acción: --

V2.4.5: Verificar que se haga al menos una iteración adicional en el algoritmo de encriptación, que el salt sea secreto y que sea generado por un generador de bits aleatorio aprobado y propocione la fuerza de seguridad minima. El salt se debe almacenar por separado de las contraseñas encriptadas, por ejemplo en un dispositivo tipo un modulo de seguridad (hardware).

Análisis: No Aplica

Plan de acción: Como obtenemos el salt dinámicamente a partir de la password ingresada por el usuario, no necesitamos almacenarlo.

V2.5.1: Verificar que en el recupero de clave, la clave no sea enviada en texto al usuario.

Análisis: Aplica

Plan de acción: Para recuperar la clave, enviamos un link para que modifiquen la clave desde el sitio. La clave vieja pierde validez, y debe definirse una nueva. El link incluye información encriptada y token para verificar desde el controller que la invocación del action es a través de un link que nosotros mismos generamos con anterioridad.

V2.5.2: Verificar que el recupero de claves basado en conocimiento o preguntas secretas no esté presente.

Análisis: No aplica

Plan de acción: La recuperación de passwords se realiza a través de un link que llega por correo electrónico.

V2.5.3: Verificar que el recupero de clave no revele la contraseña actual de ninguna forma.

Análisis: No aplica

Plan de acción: No revelamos la clave de ninguna forma.

V2.5.4: Verificar que las cuentas compartidas o default no estén presentes. Por ejemplo root, admin, etc. Esto lo podemos meter en la lista de uno de los puntos de arriba, tipo claves conocidas no válidas.

Análisis: Aplica

Plan de acción: Utilizamos una validación por lista negra en las cuales tenemos las cuentas de mail que no se pueden utilizar.

V2.5.5: Verificar que si se reemplaza o cambia un factor de autorización, por ejemplo agregando algo más aparte del email y contraseña, el usuario sea notificado.

Análisis: Aplica

Plan de acción: Si se realiza algún agregado al formulario de login, se notificará al usuario a través de correo electrónico y los cambios se detallaran en la home del sitio.

***V5: Validation, Sanitization and Encoding Verification Requirements***

V5.1: Verificar que la aplicación tenga defensas frente a ataques del tipo de contaminación de parámetros HTTP, particularmente si el framework de la aplicación no hace distinción sobre el origen de los parámetros que llegan por get, post, etc.

Análisis: ¿?

Plan de acción: ¿?

V5.2: Verificar que el framework proteja contra ataques de asignación masiva de parámetros, como campos privados o similar.

Análisis:

Plan de acción:

V5.3: Verificar que todos los inputs sean validados usando validación positiva (lista blanca)

Análisis: Aplica

Plan de acción: Se utilizan validaciones de los campos del lado cliente y también de lado servidor.

V5.4: Verificar que todos los datos sean verificados con los formatos correspondientes, por ejemplo caracteres, largo, y patrones. Como por ejemplo, si fuese tarjeta de crédito, los números validos, etc. En nuestro caso aplicaría que sea un email valido el email ingresado.

Análisis: Aplica

Plan de acción: Se verifica que los campos requeridos cumplan con el formato esperado a través de regular expressions.

V5.5: Verificar que la url envíe solo a destinos que tengamos en nuestra lista blanca, y mostrar advertencias cuando se está dirigiendo contenido no autorizado. Por ejemplo si nos cambian la url, o si nos agregan cosas a nuestro post.

Análisis:

Plan de acción:

**V7: Error Handling and Logging Verification Requirements**

V7.1: Verificar que la aplicación no loguee credenciales o detalles de pago. Los tokens de sesión deben ser guardados encriptados.(No usamos tokens )

Análisis: No aplica

Plan de acción: No guardamos datos en sesión.

V7.2: Verificar que la aplicación no registre otros datos sensibles definidas bajo normas de privacidad o políticas de seguridad relevante.

Análisis: No aplica

Plan de acción: No guardamos datos sensibles como datos de tarjeta de crédito.

V7.3: Verificar que los logs incluyan logins satisfactorios y fallidos, fallos de deserialización, o sea cuando nos llegan cosas que no esperamos, y fallos de validación de los inputs de entrada.

Análisis: Aplica

Plan de acción: se incluyeron los logs correspondientes ante intentos de ingreso erróneos a la app.

V7.4: Verificar que los logs incluyan información necesaria que permitan una investigación detallada del lapso de tiempo en que las cosas pasaron.

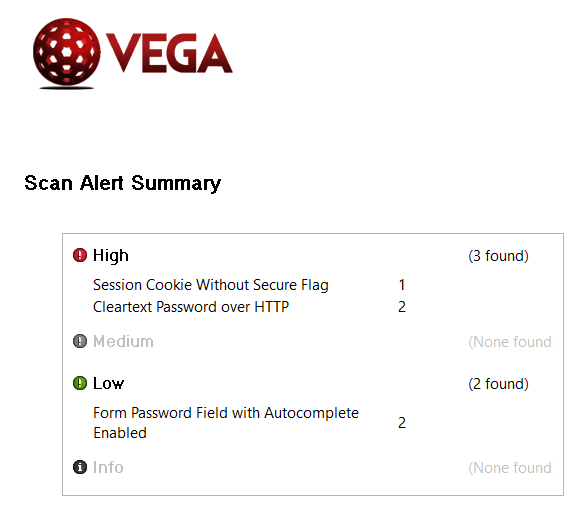
Análisis: Aplica

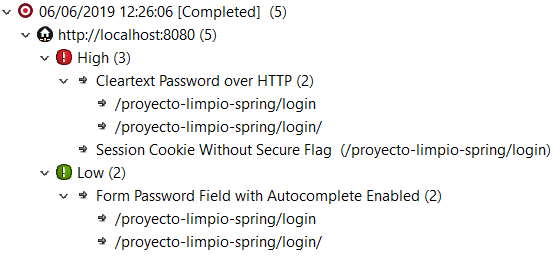
Plan de acción: Tenemos logs de todas las acciones del usuario con timestamps para verificar la forma en que se acontecieron y así dar un seguimiento completo.

# Entrega 3 – 10 de Junio

Ejecutar el scanner Vega y reporte de problemas encontrados. https://subgraph.com/vega/

Primer análisis

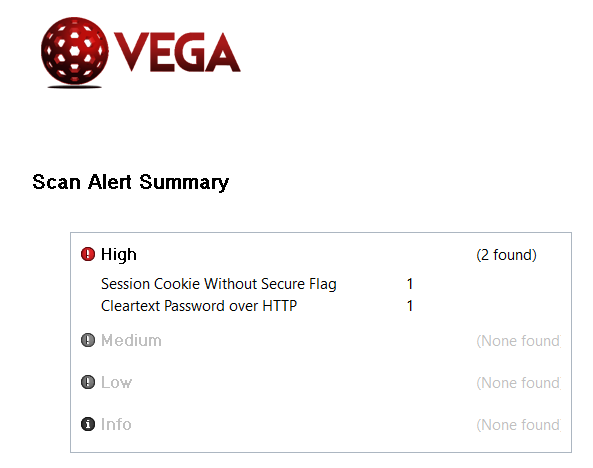


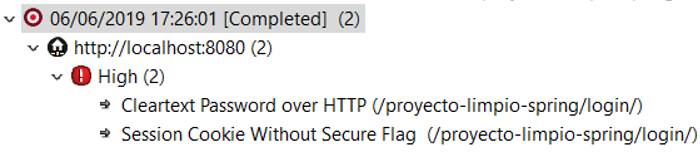


Primero decidimos resolver los de complejidad Low, que se debían a que no habíamos especificado los Autocomplete para los campos del login.

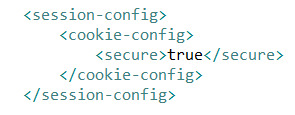
Lo resolvimos agregando el tag autocomplete="off" en el login, cambiar password, recuperar password, y registro de usuario.

Segundo análisis

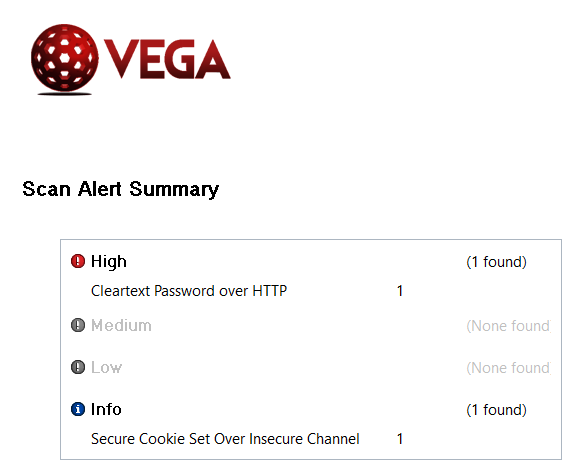


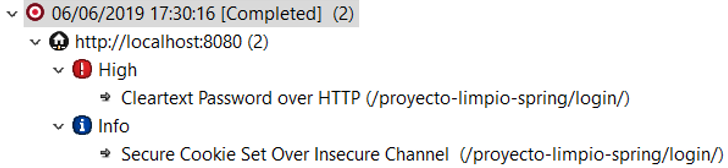


Analizamos el origen del segundo ítem, y lo resolvimos seteando la key “Secure” dentro del archivo web.xml



Tercer análisis

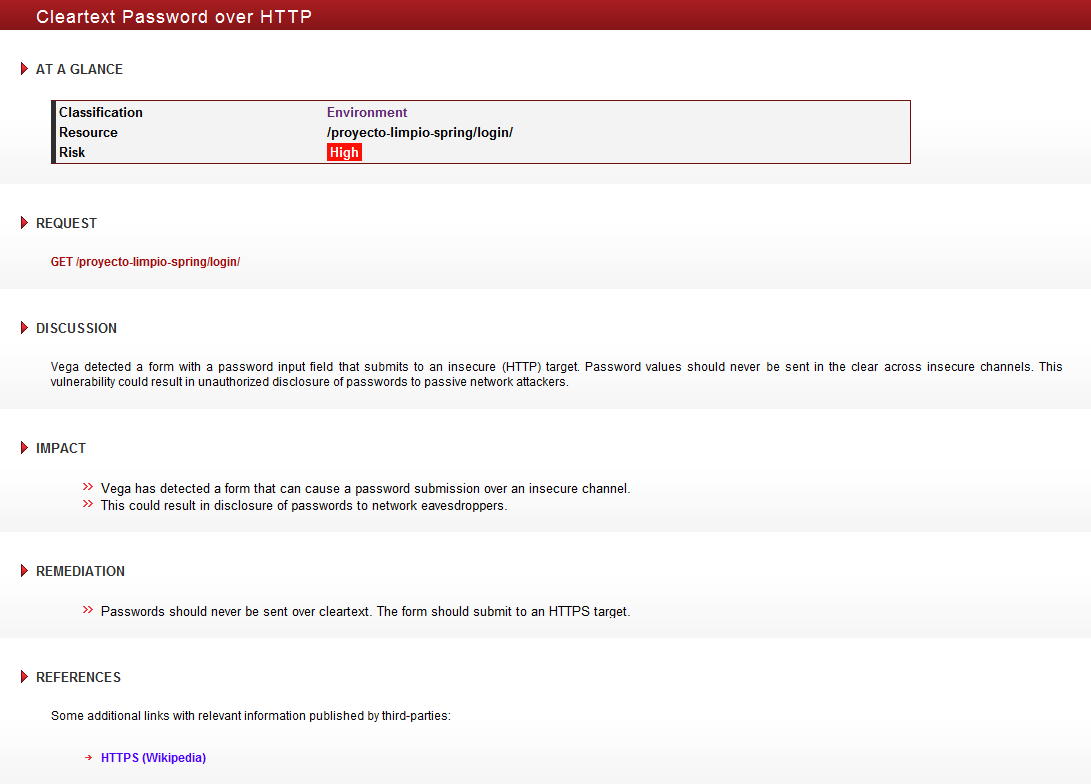




Notamos que el punto de seguridad de cookies pasó a Info, en lugar de High.

Luego, gracias a la herramienta detectamos que ambos puntos que quedan por resolver, desaparecerían estableciendo un canal HTTPS, con un certificado de seguridad.

**Ítem de severidad HIGH**



**Ítem de severidad INFO:**

