edix
Digital Workers

Actividad de desarrollo: Seguridad en redes

DEVOPS - UF4

Seguridad en Redes

Vamos a poner en práctica los conceptos de seguridad que acabamos de aprender, empleando para ello el servicio web que estamos creando durante este curso.

Enunciado

Para proteger el acceso y los datos de nuestro nuevo servicio web, se establecen los siguientes **requisitos**:

1. Se limitará el acceso a usuarios registrados: se crearán nuevos endpoints:
    a. **Creación de usuario**, donde como mínimo se dará de alta un usuario identificado por un ID y una contraseña. Por motivos de seguridad, **la contraseña nunca se almacena en claro.**
    b. **Eliminación de usuario,** servicio complementario al anterior, elimina un usuario del sistema e invalida todos sus tokens.
        i. Evidentemente, es un endpoint al que solo puede acceder el propio usuario.
    c. **Login**, servicio que responderá a un par ID/contraseña válidos con un token de servicio, **válido durante una hora como máximo.**
    d. **Logout,** servicio que invalida automáticamente los tokens de un usuario.
2. Se protegerá el acceso a los endpoints existentes, empleando un token que **se enviará mediante cabeceras HTTP**.
    a. Se debe validar el token para ver que no sea demasiado antiguo, y que no corresponda a un usuario inválido.
3. Para proteger los datos en tránsito, se requiere que el acceso al servicio aproveche las capacidades de cifrado de datos protocolo TLS.
    a. En el caso de Python, podemos conseguir un servidor Flask SSL de forma trivial:

```
$  cat requirements.txt
Flask==2.0.2
pyOpenSSL==21.0.0

 $  cat app.py
from flask import Flask
app = Flask(__name__)

@app.route("/")
```

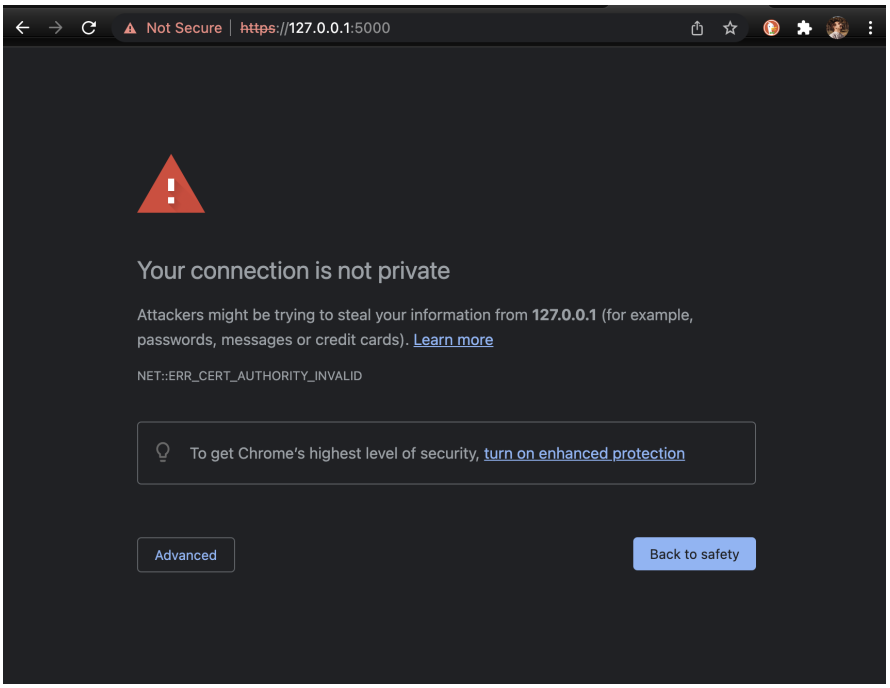
```
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run(ssl_context='adhoc')
```

b. Si estamos empleando Java, el reto será mayor. Se recomienda echar un vistazo a la clase HttpsServer, existen ejemplos en Stack Overflow para empezar a trabajar.

c. Existen más soluciones, mientras consigamos **que los datos sensibles no viajen en claro por la red**, la solución es válida.

Nota

Es posible, dependiendo de la solución elegida, que el navegador no reconozca el certificado que se use.



Esto es **perfectamente válido** durante el desarrollo, en absoluto penalizará la nota final.

Si conseguimos un certificado local válido para su importación y uso, deberá aportarse en la carpeta '**certs/**' de la entrega.

Criterios de evaluación

La actividad será totalmente correcta si y solo si el servicio consigue restringir el acceso a los endpoints de consulta mediante un token válido.

Hito	Puntuación
Endpoint de login + procesamiento de cabeceras	+50%
Endpoints de logout y gestión de usuarios	+15%
Cifrado de datos en tránsito	+15%
Código claro, limpio y mantenible. Verbos HTTP y semántica REST adecuada.	+10%
Tests que validan los requisitos pedidos	+10%

Serán motivo de penalización:

- Empleo de soluciones o algoritmos **ineficientes.**
- Almacenamiento o procesamiento inseguro de las credenciales.
    - Si existe una manera de **forjar un token válido**, la solución será incorrecta.
        - **NOTA:** en el caso de no implementar HTTPS, no se considerarán los datos en tránsito para esto.
- Ausencia **total** de tests unitarios.

Se valorará de forma muy positiva:

- La documentación, tanto en el código como en el entregable, en su justa medida.
- Uso de Docker para levantar una base de datos que almacene datos de usuario.
- Empleo y documentación de scripts de apoyo.
    - Por ejemplo, para la creación de pares de certificados.
- Nos podemos volver locos **perfectamente,** y desplegarlo en un cluster de Kubernetes o en la nube.
    - Mientras cumpla requisitos, la solución será válida **siempre y cuando** se encuentre todo debidamente declarado en ficheros que podamos emplear para automatizar el despliegue.
    - Recordemos que no todos los recursos son gratuitos en los *free tiers* de las Clouds, o dejan de serlo pasado un tiempo.