línea horizontal

## Baltazar Ruiz

INFORME DE VULNERABILIDADES WEB BASICAS

Este informe se va a centrar en la auditoría de una aplicacion web, que se ha encontrado ser vulnerable. En esta guía encontrarás información específica sobre cada una de las vulnerabilidades que la pagina: WebGoat, en su Version 8.1.0 dispone.

**RESUMEN EJECUTIVO**

1. Resumen de la evaluacion
2. Estado de seguridad
3. Recomendaciones

**METODOLOGIA**

1. Fase de reconocimiento
2. Vulnerabilidades descubiertas

[**PRODUCTO Y FASES**](#_c5rpsdy8g2ak)

[Fases del proyecto](#_xi2np1ocu1h5)

[Reuniones semanales](#_kb4m3fnuara)

[**LISTA DE TAREAS PARA LA INCORPORACIÓN**](#_x5u0l8hx0kbh)

[Semana 1](#_jltys38rhgql)

[Semana 2](#_3vm19g16qm7v)

[**RECURSOS**](#_i1tpq79if8d)

[Listas de distribución2](#_ayk4vet9v6jp)

[Glosario de términos](#_3fg7k4srhys9)

# 

## 

**RESUMEN EJECUTIVO**

Este informe muestra una evaluación de seguridad y penetración en la cual diversas pruebas se llevaron a cabo contra la aplicación web “WebGoat”.

El propósito del informe era utilizar técnicas de explotación para evaluar la seguridad de la aplicación frente a los criterios vistos a lo largo de esta semana, para validar sus mecanismos de seguridad e identificar posibles amenazas y vulnerabilidades.

**La evaluacion proporciona:**

información sobre la resiliencia de la aplicación para soportar ataques de usuarios no autorizados y el potencial de

usuarios que pueden abusar de sus privilegios y acceso.

Este informe actual detalla el alcance de las pruebas realizadas y todos los hallazgos significativos junto con las medidas correctivas detalladas.

**IMPORTANTE**

El resumen a continuación proporciona a la audiencia no técnica un resumen de los hallazgos durante la evaluación, junto con las mejores prácticas para corregir.

| **Resumen de evaluación**  Según la evaluación de seguridad a la aplicacion web **“WebGoat”**, el estado actual de las vulnerabilidades identificadas esta en un **Estado:**  **CRÍTICO,** que si no se aborda a tiempo, estas vulnerabilidades podrían ser un desencadenante de una brecha de seguridad cibernética.  Estas vulnerabilidades, se puede solucionar fácilmente siguiendo las mejores prácticas y recomendaciones proporcionadas a lo largo del informe.  **Vulnerabilidades a tener en cuenta por orden de peligro:**  **A(3) INJECCION**  **A(5) CONFIGURACION INCORRECTA DE SEGURIDAD**  **A(6) COMPONENTES VULNERABLES O OBSOLETOS**  **A(7) FALLO DE IDENTIFICACION Y AUTENTICACION**  **Recomendaciones**  Adoptar las mejores prácticas y las herramientas adecuadas es fundamental para mitigar los riesgos y reforzar la seguridad de las aplicaciones web.   * **Realizar una auditoría de seguridad exhaustiva**   Las auditorías de seguridad periódicas son un método excelente para garantizar que se siguen las mejores prácticas de seguridad en tu aplicación web y encontrar rápidamente cualquier fallo potencial en tus sistemas. Una auditoría de seguridad no sólo puede ayudarte a estar al tanto de las posibles vulnerabilidades, sino también a proteger tu negocio.   * **Encriptar todos los datos**   Cuando alguien utiliza tu aplicación web, puede revelar información sensible. Esta información no debe ser accesible a ninguna parte no autorizada. Por lo tanto, es fundamental garantizar que tu aplicación web proporcione cifrado de datos durante el tránsito y en reposo. |
| --- |

# 

# METODOLOGIA

En esta seccion podran encontrar como la auditoria de la pagina WebGoat se llevo a cabo.

## Fase de reconocimiento

En esta primera fase nuestro objetivo era recolectar la mayor capacidad de información, relacionada a la aplicacion web.

**Herramientas utilizadas:** **Nmap, Wappalyzer.**

Mediante el uso de herramientas de escaneo como, **“Nmap”**, pudimos encontrar que los puertos **8080** y **9090** se encontraban abiertos. Tambien gracias a la misma descubrimos que el sistema operativo que se estaba utilizando se trataba de un **“Linux”**. El lenguaje de programamcion que se estaba utilizando lo descubrimos al realizar pruebas con **“Wappalyzer”**

Lenguaje de Programacion utilizado por WebGoat: **Java**

Fue mediante el uso de estos puertos que pudimos acceder a la aplicacion web.

## 

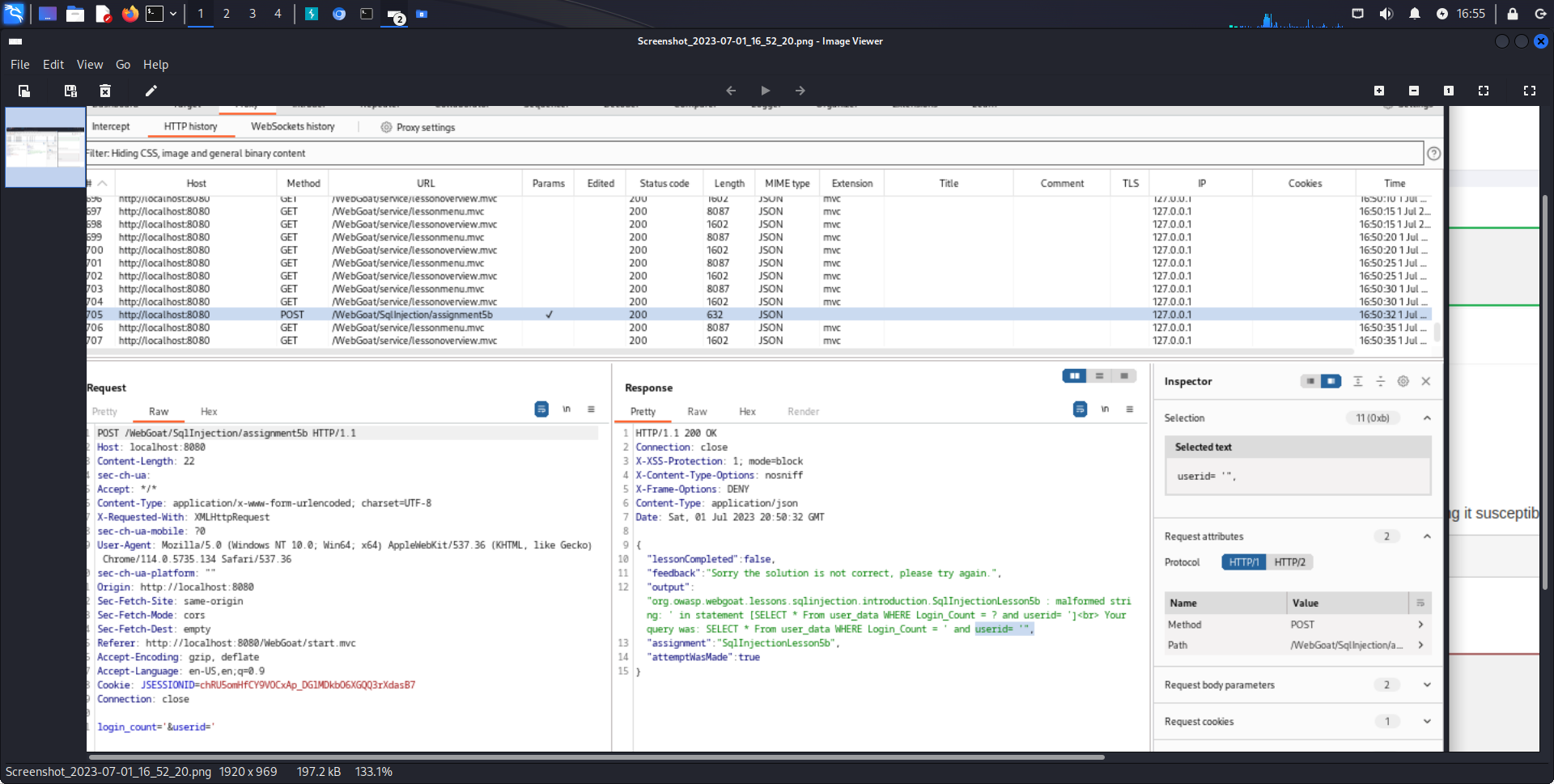
## VULNERABILIDADES DESCUBIERTAS

En este parte del informe se van a citar todas las vulnerabilidades descubiertas a lo largo de esta auditoria.

**HERRAMIENTAS UTILIZADAS A LO LARGO DE LAS EVALUACIONES: BURP**

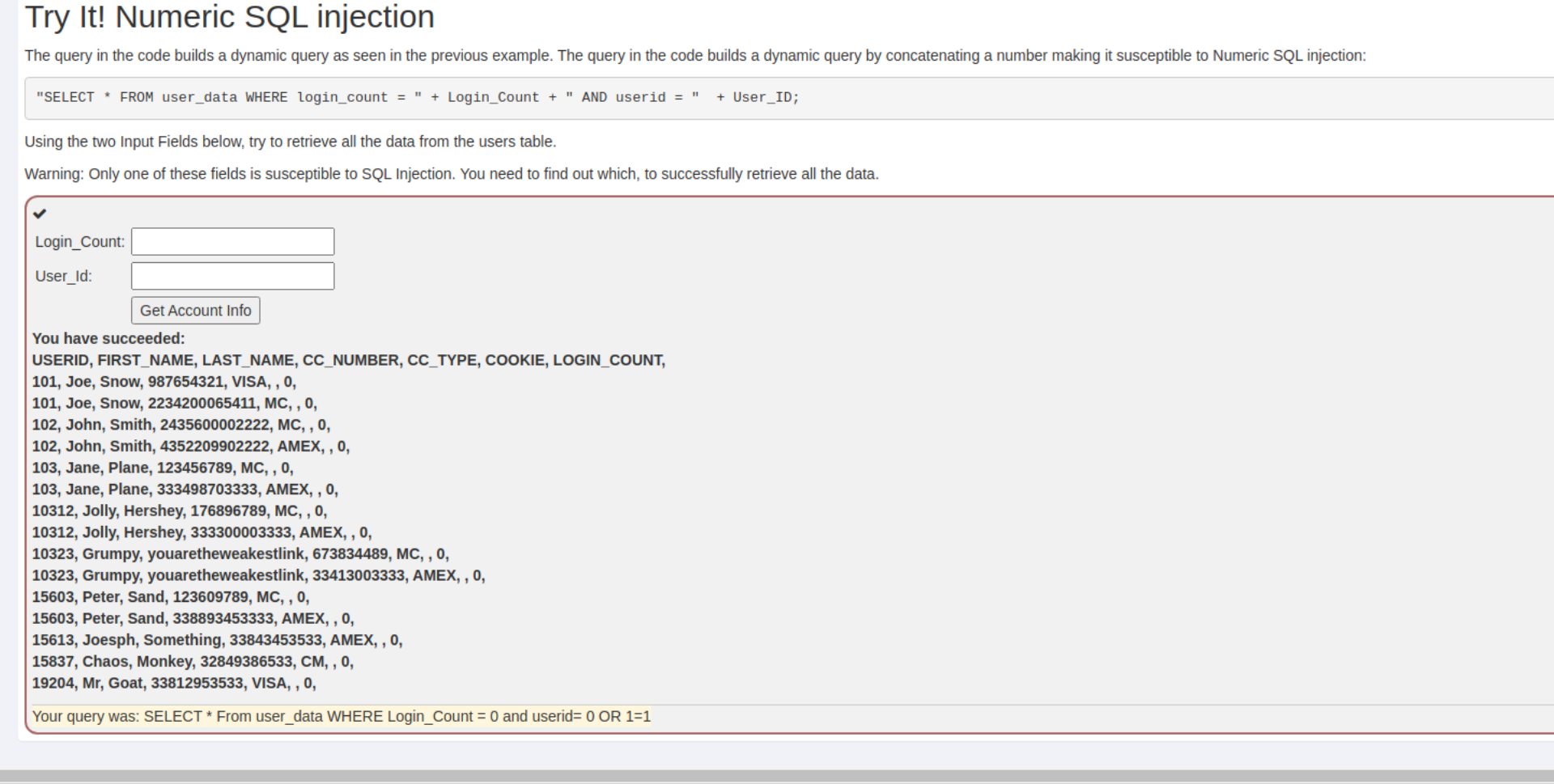
# NUMERIC SQL INJECTION A3 (10)

Los ataques contra parámetros numéricos son la forma más sencilla de lograr una inyección SQL. Este tipo de vulnerabilidad también está muy extendido ya que los desarrolladores suelen considerar que los parámetros numéricos son seguros cuando en la mayoría de los casos no lo son.

lenguajes de tipo débil, como PHP, no obligan a las variables a mantener su tipo de datos inicial. Como resultado, es posible insertar una instrucción SQL manipulada en parámetros que se suponía que contenían valores numéricos. Obviamente, es una situación perfecta para atacar y enviar segmentos SQL.

## userid = ’ “

* Haciendo evaluaciones y Con la ayuda de Burp, descubrimos que habia un fallo de vulnerabilidad en el segundo input que disponia el apartado 10. Este segundo apartado, USERID, era susceptible a ataques de injeccion SQL.
* **Si una aplicación genera consultas SQL simplemente concatenando las cadenas proporcionadas por el usuario a la consulta, es probable que la aplicación sea muy susceptible a ataques de String SQL.**
* Convirtiendo el Valor userid a TRUE logramos obtener acceso a todos los usuarios de la tabla.



# SQL INJECTION A3 (11)

## 

* Aca podemos ver otro ejemplo de lo hablado anteriormente.

Pudimos acceder a la tabla de usuarios facilmente utilizando comandos de consulta basicos, SQL. Disponiendo de un solo numero de autenticacion TAN y modificando la consulta para que sea verdadera siempre, pudimos ver la informacion de salarios de cada uno de los usuarios de la tabla.

**SOLUCIÓN**

* Prevenir la inyección SQL requiere mantener los datos separados de los comandos y consultas. Usar una API segura, que evite usar el intérprete por completo.

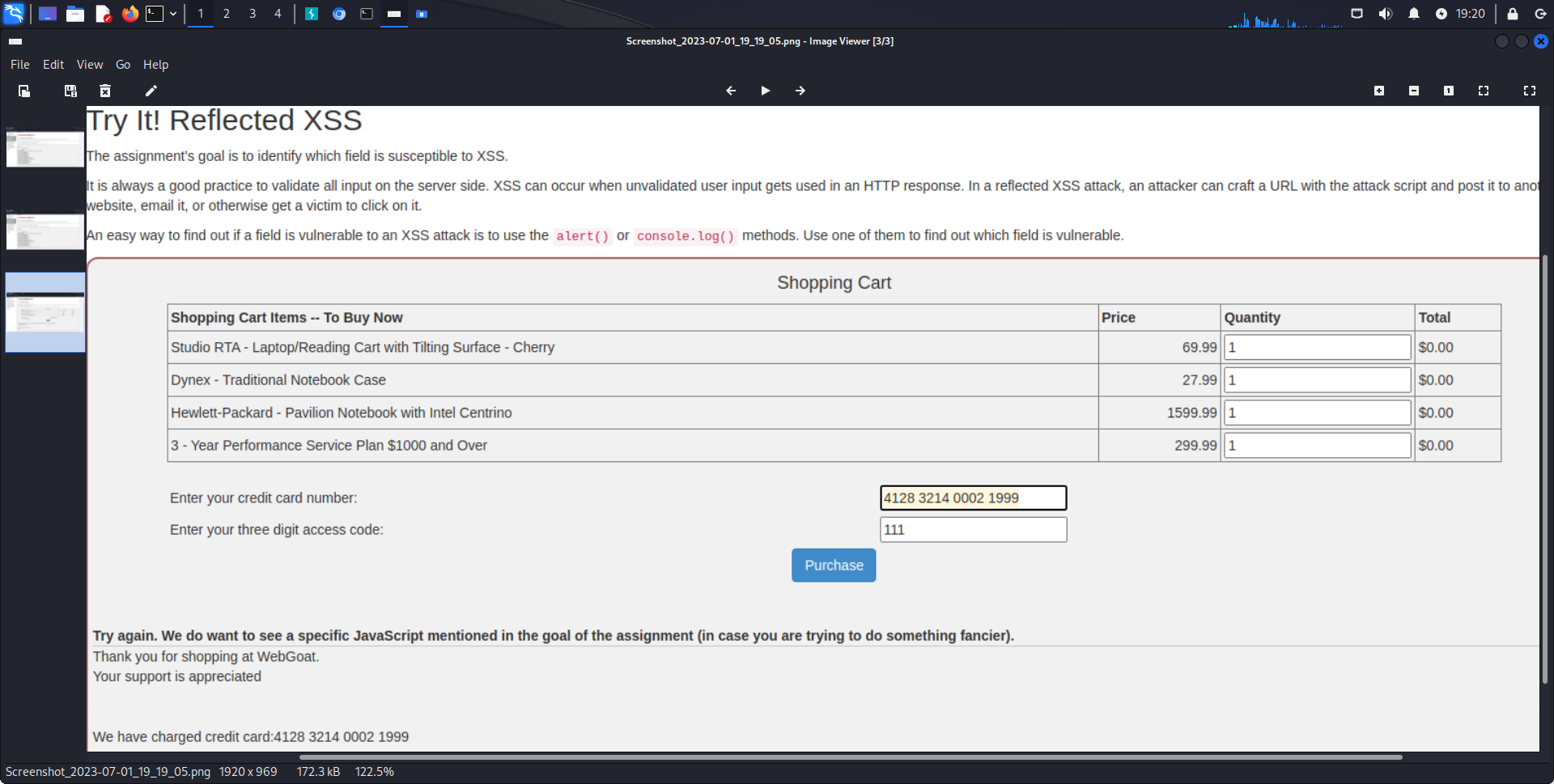
**CROSS SITE SCRIPTING A3 (7)**

Avanzando en la evaluacion encontramos vulnerabilidades a ataques de XSS Reflejado. Utilizando un Script basico: <script>alert("XSS Test")</script> Testeamos si alguno de los campos de input era susceptible a alguno de estos ataques.

**SOLUCIÓN**

* Utiliza antivirus y comprueba que detecta malware correctamente.
* Mantén las aplicaciones y sistemas (navegadores web, antivirus, sistema operativo) actualizados.
* Utiliza frameworks que codifiquen el contenido para prevenir ataques XSS, como Ruby 3.0 o React JS.
* Filtra la entrada de datos del usuario lo más específicamente posible.
* Codifica los datos de salida para los usuarios (HTML, URLs, JavaScript y CSS).
* Aplica políticas de seguridad de contenido (CSP).
* Implementa un WAF (Web Application Firewall). Al igual que con las inyecciones SQL, un firewall de aplicaciones web ayuda a impedir la ejecución de ataques XSS, filtrando y monitoreando el tráfico HTTP entre una aplicación e Internet.

**Ejemplo del campo de Input susceptible: Marcado en color Amarillo**



**security misconfiguration A5 (4) y (7)**

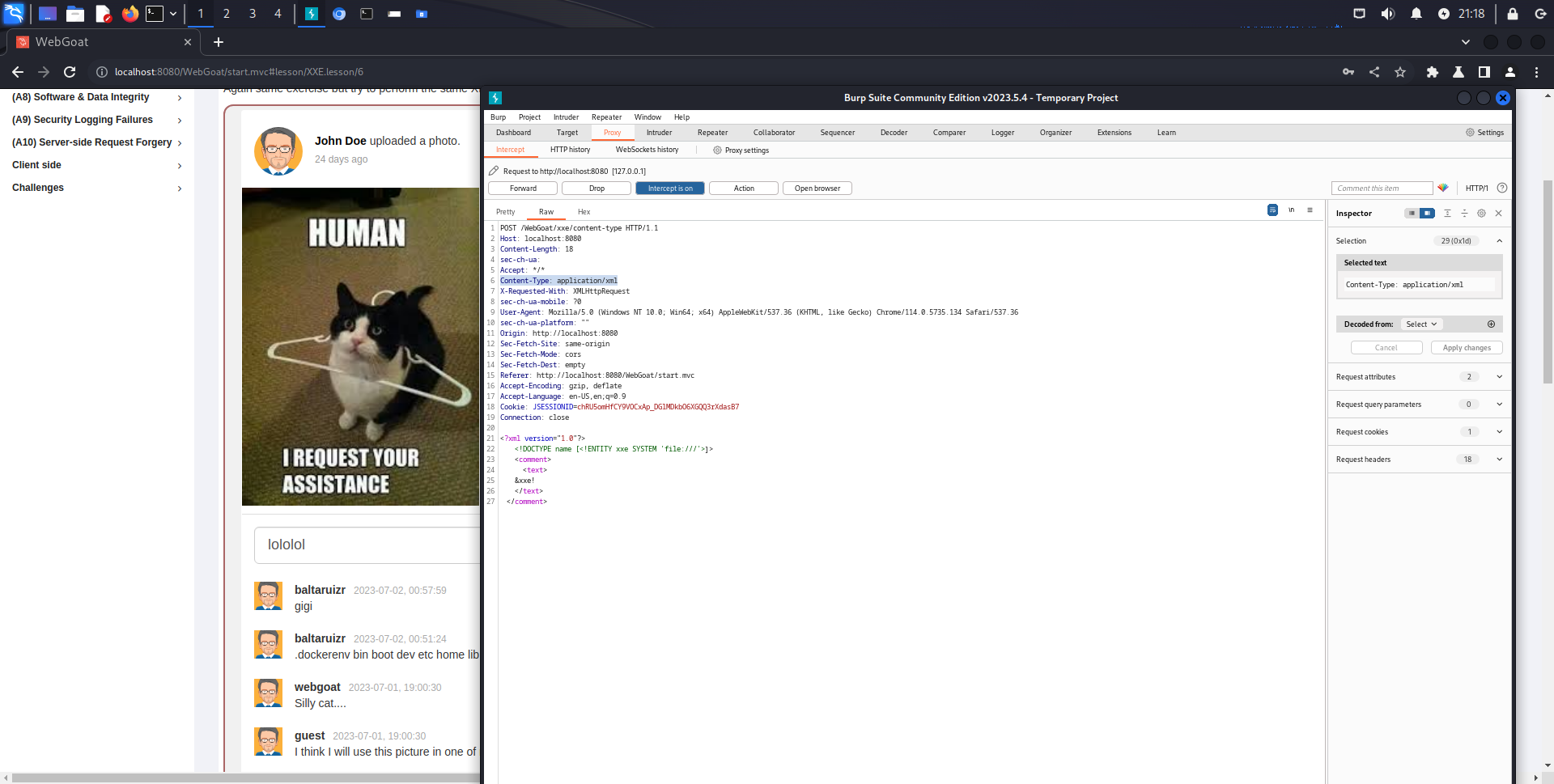
Utilizando el interruptor de Burp, fuimos capaces de explotar una vulnerabilidad de tipo XXE. esta vulnerabilidad de seguridad web permitio interferir con el procesamiento de datos XML de una aplicación. ver archivos en el sistema de archivos del servidor de aplicaciones e interactúar con cualquier sistema externo o de back-end al que pueda acceder la propia aplicación.

**SOLUCION**

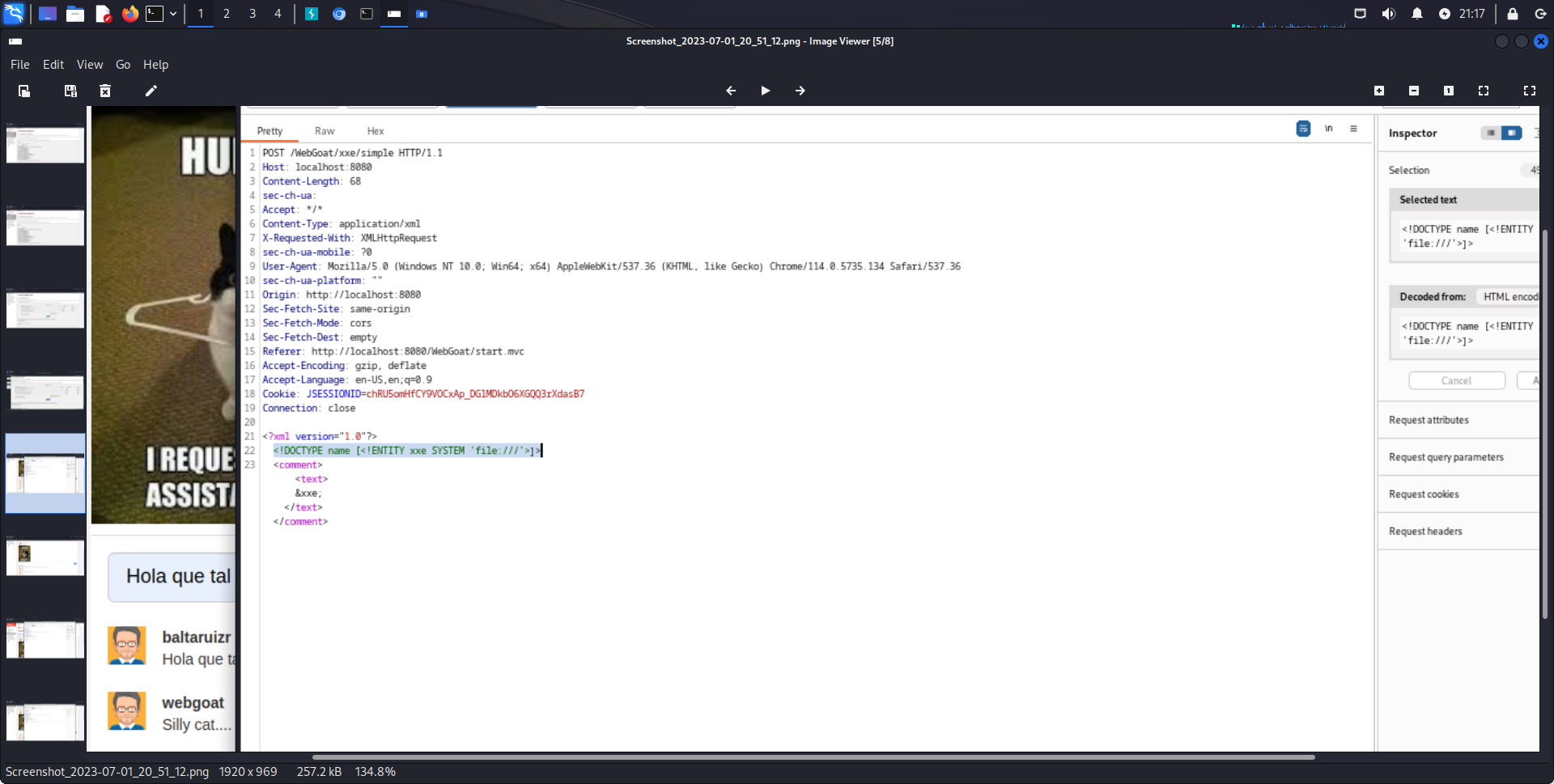
Prácticamente todas las vulnerabilidades XXE surgen porque la biblioteca de análisis XML de la aplicación admite funciones XML potencialmente peligrosas que la aplicación no necesita ni pretende usar. La forma más fácil y efectiva de prevenir los ataques XXE es deshabilitar esas funciones.

Generalmente, es suficiente deshabilitar la resolución de entidades externas y deshabilitar el soporte para XInclude. Por lo general, esto se puede hacer a través de las opciones de configuración o anulando mediante programación el comportamiento predeterminado.

**Ejemplo de procesamiento de datos tipo XML:**



**Ejemplo de Listado del root del Sistema de archivos:**



**VULN & OUTDATED COMPONENTS A (6)**

Los componentes vulnerables o obsoletos son un problema que no deberiamos de pasar por alto. Los componentes vulnerables y obsoletos se refieren a cuando el código de código abierto o propietario contiene vulnerabilidades de software o ya no se mantiene. Este código puede estar en forma de bibliotecas o marcos, desafortunadamente, este código a menudo se implementa con poca o ninguna consideración por la seguridad, lo que genera consecuencias potencialmente graves para los usuarios de la aplicación y pone en riesgo la reputación de las empresas.

Se ha encontrado mediante un ataque XSS que el codigo del que dispone WebGoat esta desactualizado. Es importante tener en cuenta que somos susceptibles a un ataque si:

* Si no conoce las versiones de todos los componentes que utiliza (tanto del lado del cliente como del lado del servidor). Esto incluye los componentes que usa directamente, así como las dependencias anidadas.
* Si el software es vulnerable, no es compatible o está desactualizado. Esto incluye el sistema operativo, el servidor web/de aplicaciones, el sistema de administración de bases de datos (DBMS), las aplicaciones, las API y todos los componentes, los entornos de tiempo de ejecución y las bibliotecas.
* Si no busca vulnerabilidades con regularidad.
* Si no corrige o actualiza la plataforma, diariamente.
* Si los desarrolladores de software no prueban la compatibilidad de las bibliotecas actualizadas, mejoradas o parcheadas.
* Si no asegura las configuraciones de los componentes

**Como prevenir**

Debe existir un proceso de administración de parches para:

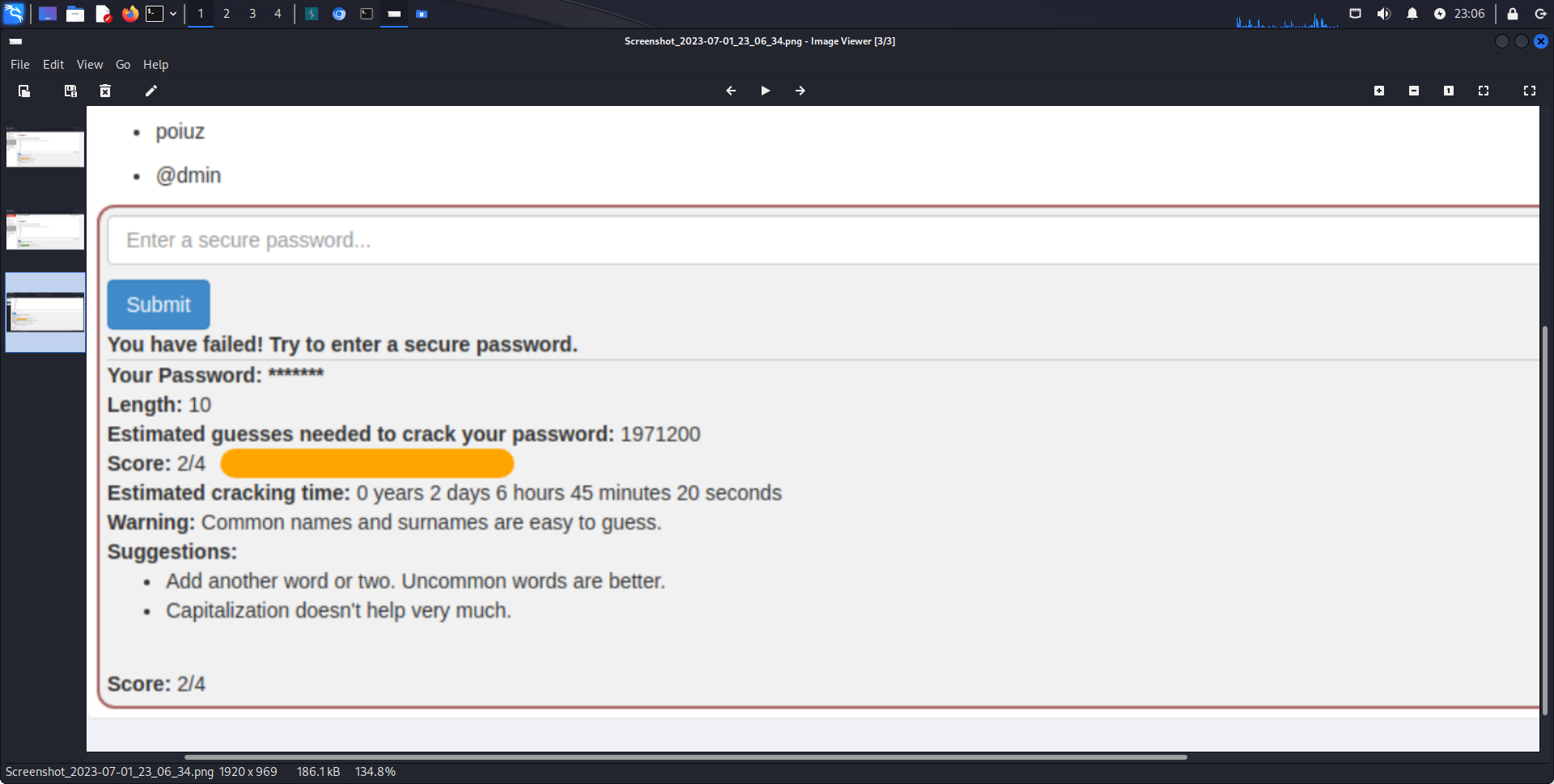
* Eliminar las dependencias no utilizadas, las funciones, los componentes, los archivos y la documentación innecesarios.
* Hacer un inventario continuo de las versiones de los componentes del lado del cliente y del lado del servidor

Además, los componentes solo deben obtenerse de fuentes confiables con preferencia por paquetes firmados para preservar la integridad de los datos. También se recomienda probar todos los parches y actualizaciones dentro de un entorno de ensayo para evitar introducir problemas inesperados en los sistemas de producción. Una prueba de penetración de aplicaciones web puede ayudar a comprender mejor los riesgos y vulnerabilidades presentes en sus aplicaciones, así como ayudar a crear un plan para remediarlo.

**SECURED PASSWORDS A (7)**

Las contraseñas débiles y fáciles de adivinar hacen que incluso la estrategia de ciberseguridad más sólida sea fácil de eludir. Si un atacante informático adivina o descifra una contraseña, el intruso puede acceder a su cuenta o sistema sin dar la alarma y comprometer cualquier activo que haya mantenido seguro detrás de una contraseña.

**Ejemplo de contraseña debil:**



Una contraseña segura es una palabra o frase única que un hacker no puede adivinar o descifrar fácilmente. Estas son las características principales de una contraseña confiable y segura:

* Mínimo 12 caracteres (cuanto más, mejor).
* Tiene una combinación de letras mayúsculas y minúsculas, números, puntuación y símbolos especiales.
* Aleatorio y único.

Si bien la complejidad mejora la seguridad de las contraseñas, la longitud es la característica clave. La mejor manera de hacer que una contraseña sea segura es hacerla larga.