INFORME DE AUDITORÍA

ÍNDICE

1. ÁMBITO Y ALCANCE DE LA AUDITORÍA.

El objetivo principal de esta auditoría fue realizar una evaluación de seguridad básica (pentest) sobre la aplicación web vulnerable WebGoat 8.1.0, con el fin de aplicar y demostrar la metodología de pruebas de penetración en un entorno controlado. La prueba se centró en las fases de reconocimiento (Information Gathering), explotación de vulnerabilidades comunes del OWASP Top 10 2021, y la posterior documentación de hallazgos en un informe formal.

El alcance de las pruebas incluyó:

- **Reconocimiento Pasivo y Activo**: Identificación de tecnologías, puertos, servicios y estructura de la aplicación.
- **Pruebas de Vulnerabilidades Específicas**: Explotación manual de lecciones guiadas dentro de WebGoat.
- **Análisis de Impacto**: Evaluación de las consecuencias teóricas de cada vulnerabilidad explotada en un entorno real.

2. INFORME EJECUTIVO

2.1. Resumen ejecutivo.

Se ha realizado una auditoría de seguridad básica sobre la aplicación web vulnerable **WebGoat**, desplegada en un entorno local controlado. El objetivo principal fue aplicar la metodología de pruebas de penetración, centrándose en las fases de **reconocimiento (Information Gathering)** y **explotación de vulnerabilidades**. La prueba se ha desarrollado de forma manual, utilizando herramientas de seguridad estándar para identificar y validar fallos de seguridad en una aplicación diseñada para fines educativos.

2.2. Vulnerabilidades principales.

CATEGORÍA	VULNERABILIDAD	RIESGO
Injection	A3 Intro: Compromising Confidentiality (String)	CRÍTICO

Injection	A3 Intro: Compromising Integrity (Query Chaining)	CRÍTICO
Injection	A3 Intro: Numeric SQL Injection	ALTO
Injection	A3 Advanced: Pulling Data from Other Tables	ALTO
Identification & Auth Failure	A7: Password Reset - Security Questions	ALTO
Injection	A3 Data Definition Language (DDL)	MEDIO
Injection	A3 Data Manipulation Language (DML)	MEDIO
Injection	A3 CSS: Reflected XSS	MEDIO
Security Misconfiguration	A5 CSRF: Post a Review on Someone Else's Behalf	MEDIO
Vuln. & Outdated Components	A6: The exploit is not always in "your" code	BAJO

Leyenda de Riesgo:

- Crítico: Compromete directamente la confidencialidad o integridad de todos los datos.
- Alto: Permite eludir autenticación, inyectar código o comandos, o acceder a datos restringidos.
- Medio: Afecta a un usuario o función específica, requiere interacción o tiene impacto limitado.
- Bajo: Impacto mínimo, requiere condiciones muy específicas o solo revela información no sensible.

2.3. Conclusiones generales.

La aplicación WebGoat, en su configuración por defecto, presenta múltiples vulnerabilidades deliberadas que reflejan fallos de seguridad comunes en el desarrollo web real. Se ha confirmado la efectividad de las técnicas de reconocimiento pasivo y la explotación manual de vulnerabilidades críticas como la Inyección SQL y el Control de Acceso Roto. Los hallazgos demuestran la importancia crítica de implementar controles de seguridad robustos en todas las capas de una aplicación, especialmente en mecanismos de autenticación y manejo de datos externos.

2.4. Recomendaciones prioritarias.

- **Validación de Entrada y Consultas Parametrizadas**: Implementar estrictos controles de validación y usar consultas preparadas para mitigar todas las formas de inyección SQL.
- **Gestión Robusta de Sesiones**: Utilizar identificadores de sesión largos, criptográficamente aleatorios y regenerarlos tras el login.
- **Cifrado y Hashing Apropiado**: Almacenar todas las credenciales usando funciones de *hash* fuertes y saladas (como bcrypt), y nunca enviar contraseñas en texto claro.
- **Revisión de Flujos de Lógica de Negocio:** Auditar especialmente las funcionalidades secundarias (como el restablecimiento de contraseña) para evitar fallos de lógica que deriven en ataques.

3. DESCRIPCIÓN DE LA AUDITORÍA

3.1. Reconocimiento.

Se realizó una evaluación exhaustiva del objetivo para entender su infraestructura, tecnologías y superficie de ataque expuesta.

Hallazgos Principales:

1. **Entorno y Accesibilidad**:
    - El objetivo se encuentra en un entorno de pruebas local (127.0.0.1), confirmado por la falta de registros públicos y una distancia de red de 0 saltos.
    - La aplicación es accesible a través de HTTP, sin capa HTTPS habilitada en este entorno.
2. **Infraestructura de Red y Servicios**:
    - Se identificaron **dos servicios HTTP activos** ejecutándose en el host:
        1. Puerto **8080/TCP**: Servicio principal de la aplicación web.
        2. Puerto **9090/TCP**: Servicio auxiliar utilizado para funcionalidades específicas de WebWolf (como la gestión de correos electrónicos simulados).
    - Ambos servicios están alojados en **servidores Apache Tomcat** y el sistema operativo subyacente es una distribución **Linux**.

3. **Arquitectura y Tecnologías de la Aplicación**:

    - La aplicación está construida con **Java** en el backend, típico de aplicaciones empaquetadas como archivos WAR para Tomcat.
    - El frontend utiliza un conjunto de **librerías JavaScript antiguas**, destacando versiones específicas como **jQuery 2.1.4** y **jQuery UI 1.10.4**, lo que señala un riesgo potencial por el uso de componentes de terceros desactualizados (Categoría A06 de OWASP).
    - La estructura de directorios descubierta confirma la naturaleza de aplicación Java, revelando rutas reservadas como /WEB-INF.

4. **Superficie de Ataque Mapeada**:

    - No se descubrieron archivos de configuración, backups o directorios administrativos expuestos de manera obvia mediante rutas comunes.
    - Se localizaron los endpoints principales de autenticación (/login, /logout) y un formulario de registro (/registration), definiendo los vectores de entrada iniciales para la aplicación.

Conclusión del Reconocimiento:

La aplicación objetivo, **WebGoat 8.1.0**, se presenta como una aplicación Java estándar desplegada en Tomcat, con un frontend que depende de componentes potencialmente vulnerables por su antigüedad. El entorno controlado y la arquitectura identificada proporcionan el contexto necesario para realizar pruebas de seguridad enfocadas en vulnerabilidades comunes del stack tecnológico detectado, como Inyección SQL, Secuestro de Sesión y Explotación de Componentes Desactualizados. La superficie de ataque inicial no presenta configuraciones groseramente inseguras a nivel de archivos expuestos.

3.2. Explotación de vulnerabilidades detectadas.

1. **Compromising Confidentiality with String SQL Injection**

Nivel de Riesgo Crítico: Permite la extracción masiva de información confidencial).

Descripción de la Vulnerabilidad

La aplicación contiene un formulario de búsqueda de empleados que toma la entrada del usuario y la concatena directamente dentro de una consulta SQL sin ningún tipo de sanitización o uso de consultas parametrizadas. Esto permite a un atacante manipular la estructura lógica de la consulta para recuperar datos no autorizados.

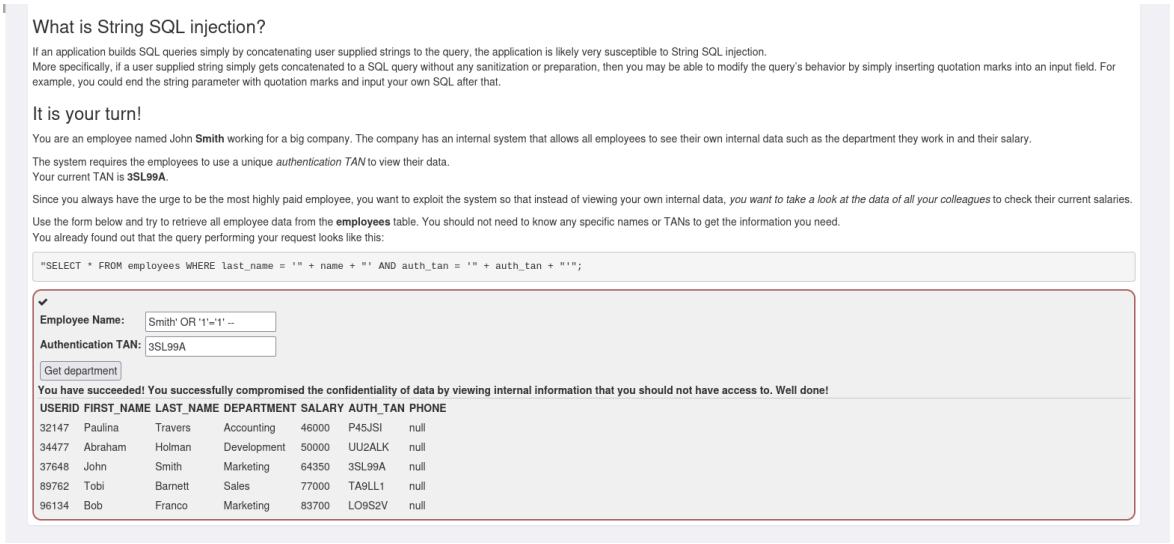
Explotación

1. **Vector de Ataque:** Campo de entrada "Employee Name".
2. **Payload Inyectado:** Smith' OR '1'='1' --
    - Smith': Cierra la cadena esperada por la aplicación.
    - OR '1'='1': Añade una condición que siempre es verdadera, forzando a que la cláusula WHERE de la consulta seleccione todos los registros.
    - -- (doble guión con espacio): Comenta el resto de la consulta SQL original, eliminando cualquier filtro adicional.

3. **Técnica:** Inyección SQL clásica de tipo *Always True* con comentario de línea.
4. **Herramienta Utilizada:** Explotación manual a través de la interfaz web de WebGoat.

Evidencia del Impacto

Tras inyectar el payload, la aplicación devolvió **la lista completa de registros de empleados** almacenados en la base de datos, incluyendo información confidencial como nombres, números de identificación (TAN), salarios y departamentos. Este resultado demuestra una **violación total de la confidencialidad** de los datos, cumpliendo el objetivo de la lección.



2. Compromising Integrity with Query chaining.

Nivel de Riesgo Crítico: Permite la modificación no autorizada de datos sensibles en la base de datos.

Descripción de la Vulnerabilidad

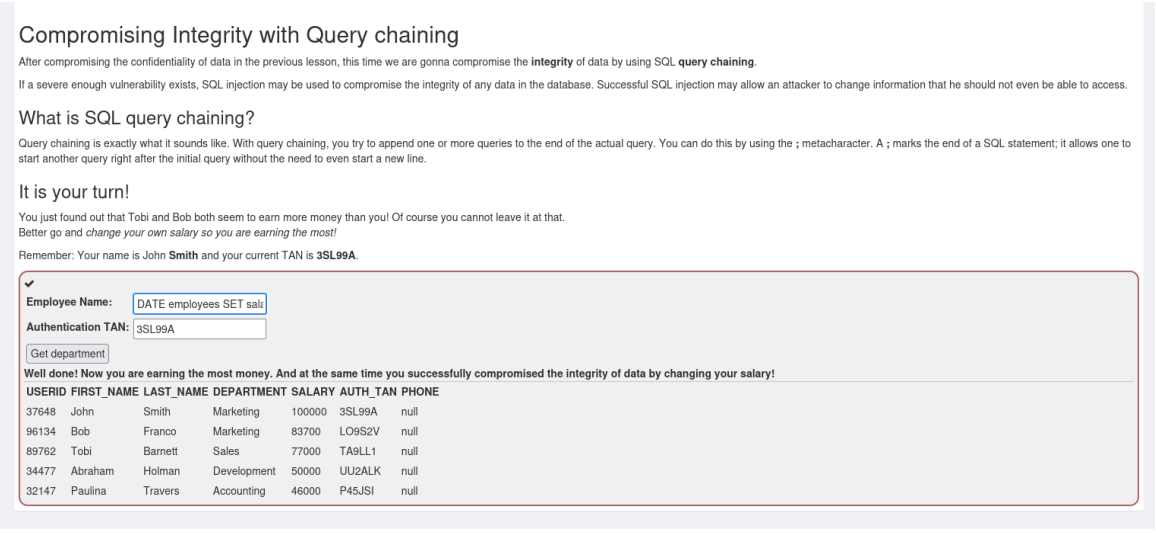
La aplicación es vulnerable a Inyección SQL de segundo orden o *query chaining*. Al no sanitizar la entrada, un atacante puede utilizar el carácter punto y coma (;) para finalizar la consulta original e inyectar una o más consultas SQL adicionales que se ejecutarán de forma secuencial. Esto permite alterar directamente la información almacenada en la base de datos.

Explotación

1. **Vector de Ataque:** Campo de entrada del formulario de búsqueda de empleados.
2. **Payload Inyectado:** '; UPDATE employees SET salary = 1600000 WHERE FIRST_NAME = 'John' AND LAST_NAME = 'Smith' --
    - ': Cierra la cadena de la consulta original.
    - ;: Termina la primera consulta y permite encadenar una nueva.
    - UPDATE employees SET salary = 1600000 ...: Consulta maliciosa que modifica el salario.
    - --: Comenta el resto de la consulta original para evitar errores de sintaxis.
3. **Técnica:** Inyección SQL con *query chaining* utilizando el terminador ; para ejecutar una sentencia UPDATE no autorizada.
4. **Herramienta Utilizada:** Explotación manual a través de la interfaz web de WebGoat.

Evidencia del Impacto

La ejecución exitosa de la consulta encadenada resultó en la **modificación no autorizada de datos financieros sensibles** en la base de datos. Se demostró la capacidad de un atacante para alterar información crítica (salarios) violando gravemente la integridad de los datos, lo que podría derivar en fraudes, pérdidas económicas o manipulación de negocio.



3. Numeric SQL Injection.

Nivel de Riesgo Alto: Permite el bypass de autenticación y la extracción de datos sensibles.

Descripción de la Vulnerabilidad

La aplicación contiene un formulario de login que utiliza un campo numérico (userid) para autenticar al usuario. Este campo es vulnerable a inyección SQL porque su valor se concatena directamente en la consulta sin ser sanitizado o tratado como un parámetro. Al ser numérico, no requiere comillas para manipular la lógica de la consulta.

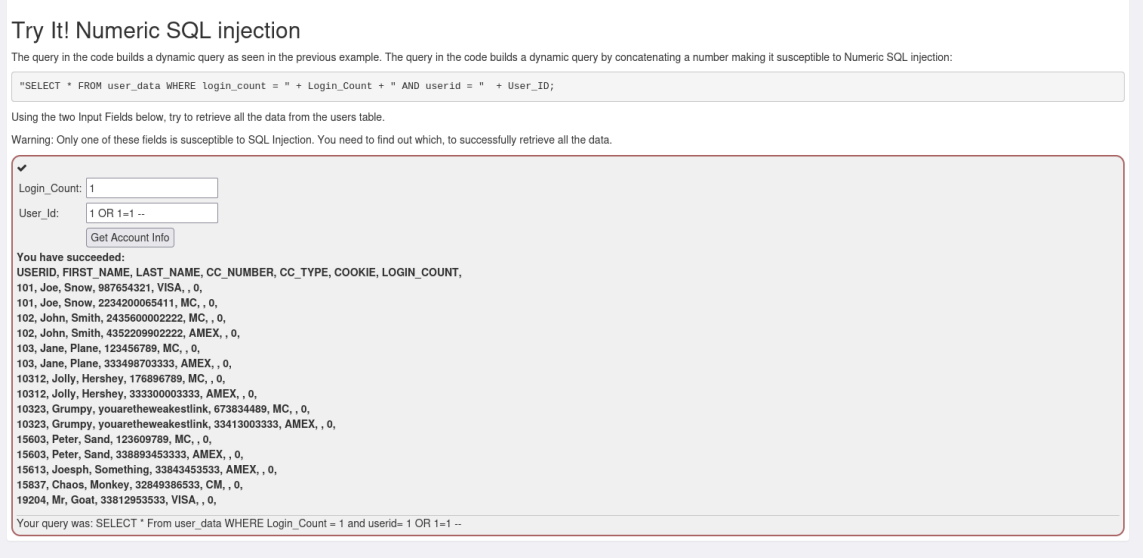
Explotación

1. **Vector de Ataque:** Campo de entrada numérico "userid".
2. **Payload Inyectado:** 1 OR 1=1 --
    - 1: Valor numérico esperado por la aplicación.
    - OR 1=1: Añade una condición que siempre es verdadera (TRUE), haciendo que la cláusula WHERE de la consulta se cumpla para cualquier registro.
    - -- (doble guión con espacio): Comenta el resto de la consulta SQL original, eliminando cualquier verificación adicional (como la contraseña o el "login count").
3. **Técnica:** Inyección SQL numérica clásica de tipo *Always True* con comentario de línea.
4. **Herramienta Utilizada:** Explotación manual a través de la interfaz web de WebGoat.

Evidencia del Impacto

Tras inyectar el payload, la aplicación **bypasseó la autenticación** y devolvió información confidencial que presumiblemente correspondía al primer usuario de la tabla o a múltiples usuarios. Los datos expuestos incluían userid, nombres, apellidos y **números de tarjeta de crédito con su tipo**, lo que representa una grave violación

de confidencialidad y demuestra cómo un fallo de autenticación puede escalar a una fuga masiva de datos sensibles.



4. Pulling Data from Other Tables.

Nivel de Riesgo Alto: Permite la extracción no autorizada de credenciales almacenadas y datos sensibles de otras tablas.

Descripción de la Vulnerabilidad

La aplicación es vulnerable a Inyección SQL de tipo **UNION**. Un campo de entrada utilizado para filtrar datos en la tabla user_data no sanitiza correctamente la entrada del usuario. Esto permite a un atacante extender la consulta original con una sentencia UNION SELECT para recuperar datos de **tablas completamente diferentes** a las previstas por la aplicación, violando la segregación de datos.

Explotación

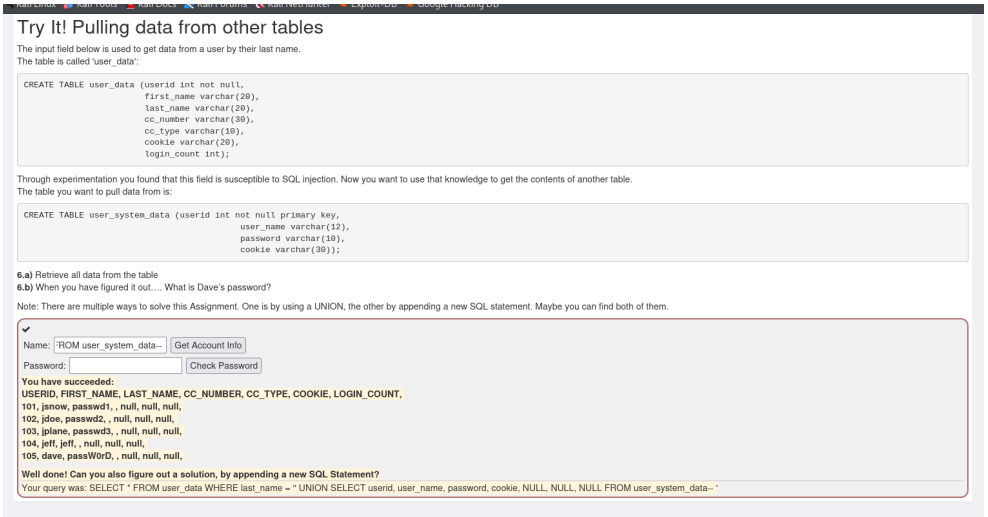
1. **Vector de Ataque:** Campo de entrada utilizado para buscar por apellido (last_name).
2. **Payload Inyectado:** ' UNION SELECT userid, user_name, password, cookie, NULL, NULL, NULL FROM user_system_data --
    - ': Cierra la cadena de la consulta original.

- UNION SELECT: Combina los resultados de la consulta original con los de una nueva consulta que selecciona datos de user_system_data.
- userid, user_name, password, cookie, NULL, NULL, NULL: Proyección de columnas que coincide en número (7) y tipo de datos aproximado con las columnas de user_data (userid, first_name, last_name, cc_number, cc_type, cookie, login_count).
- FROM user_system_data: Especifica la tabla objetivo de la que se roban datos.
- --: Comenta el resto de la consulta original.

3. **Técnica:** Inyección SQL de tipo **UNION-based**, explotando el conocimiento de la estructura de la base de datos para robar credenciales.
4. **Herramienta Utilizada:** Explotación manual a través de la interfaz web de WebGoat.

Evidencia del Impacto

La inyección fue exitosa y la aplicación devolvió en sus resultados **las credenciales de autenticación (usuario y contraseña) de todos los usuarios del sistema** almacenadas en la tabla user_system_data. Esto demuestra la capacidad de un atacante para **escalar un fallo de inyección en una funcionalidad menor hacia el robo completo de las credenciales del sistema**, lo que podría llevar a un compromiso total de la aplicación.



5. Password Reset - Security Questions.

Nivel de Riesgo Alto: Permite el restablecimiento no autorizado de la contraseña de otro usuario, lo que lleva al secuestro completo de la cuenta.

Descripción de la Vulnerabilidad

La funcionalidad de "Olvidé mi contraseña" permite a los usuarios restablecer sus credenciales respondiendo una **pregunta de seguridad predefinida** ("¿Cuál es tu color favorito?"). La implementación es insegura porque:

1. **La pregunta es débil y común** (color favorito), con un **espacio de respuestas muy limitado** (unos pocos colores comunes).
2. **No existe un mecanismo de lock-out** o límite de intentos, permitiendo pruebas ilimitadas.
3. Las respuestas de otros usuarios pueden **adivinarse mediante fuerza bruta** (probando colores comunes) o, en un caso real, **investigarse mediante OSINT** (redes sociales).

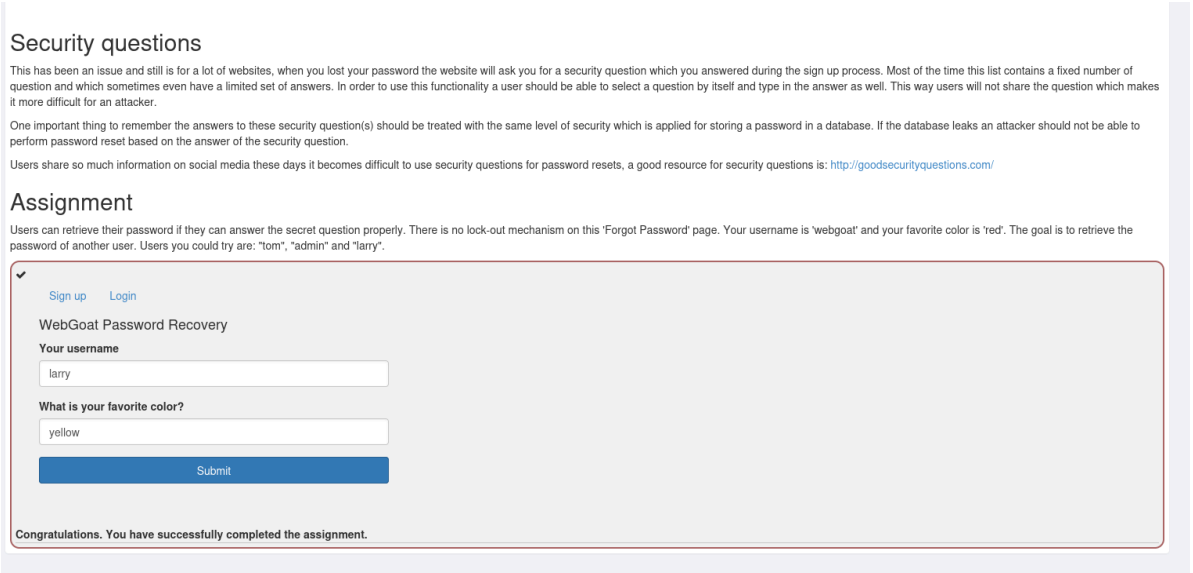
Explotación

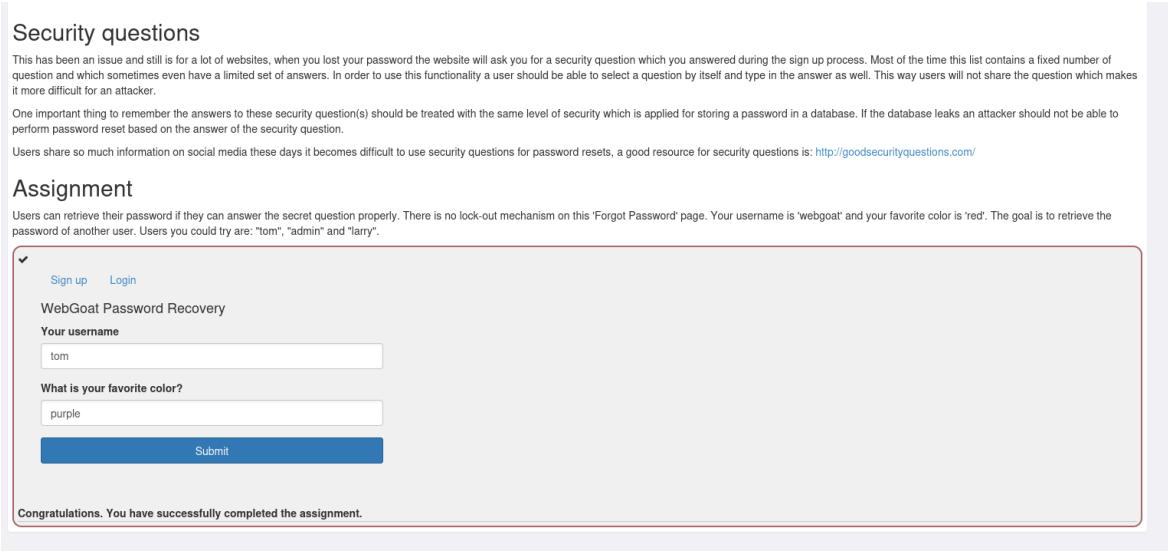
1. **Vector de Ataque:** Formulario de recuperación de contraseña.
2. **Técnica: Adivinación por fuerza bruta / enumeración de respuestas**. Se probaron respuestas comunes para el campo "color favorito" contra nombres de usuario conocidos (tom, admin, larry).
3. **Metodología:** Se ingresó el nombre de usuario objetivo (ej: tom) y se probaron sistemáticamente colores hasta encontrar la combinación correcta:
   - Usuario larry → **yellow** (amarillo)
   - Usuario tom → **purple** (púrpura)
   - Usuario admin → **green** (verde)
4. **Herramienta Utilizada:** Explotación manual a través de la interfaz web de WebGoat.

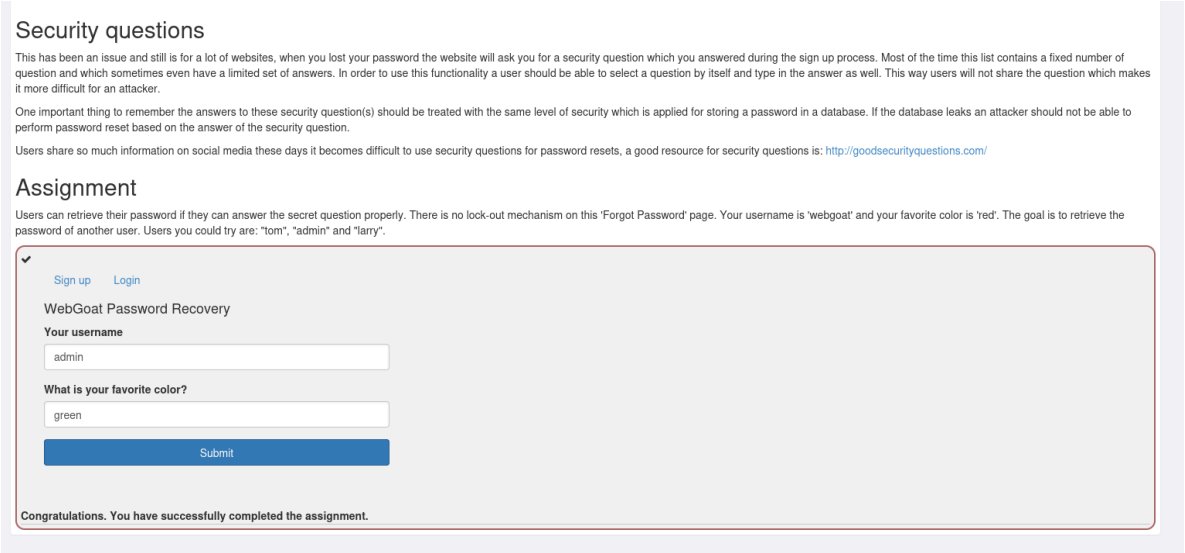
Evidencia del Impacto

Se podría **restablecer la contraseña y obtener acceso a las cuentas de otros usuarios** (tom, admin, larry) explotando la debilidad de la pregunta de seguridad y la falta de control de intentos. Esto demuestra que un mecanismo diseñado para **recuperar el**

acceso de forma segura puede convertirse en una puerta trasera para que un atacante tome el control de cuentas.







6. Data Definition Language (DDL).

Nivel de Riesgo Medio: Permite modificar permanentemente la estructura de la base de datos, afectando la integridad y disponibilidad.

Descripción de la Vulnerabilidad

La aplicación es vulnerable a un tipo avanzado de Inyección SQL que permite la ejecución de comandos de **Lenguaje de Definición de Datos (DDL)**. Estos comandos (CREATE, ALTER, DROP) **no manipulan los datos, sino la estructura misma de la base de datos** (esquema). Un atacante que consiga inyectar DDL puede añadir, modificar o eliminar tablas, columnas, índices y otros objetos, comprometiendo gravemente la integridad y la disponibilidad del sistema.

Explotación

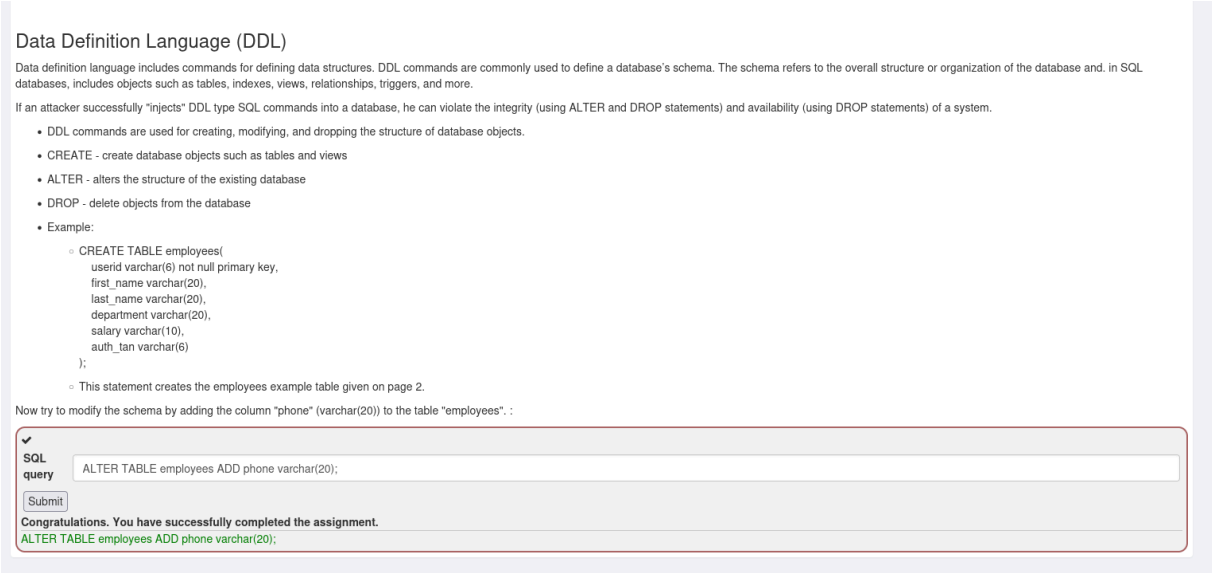
1. **Vector de Ataque:** Campo de entrada vulnerable que permite la inyección de una sentencia SQL completa.
2. **Payload Inyectado:** ALTER TABLE employees ADD phone varchar(20);
3. ALTER TABLE employees: Comando DDL que modifica la estructura de la tabla existente employees.
    - ADD phone varchar(20): Añade una nueva columna llamada phone con tipo de datos varchar(20).

4. **Técnica:** Inyección SQL clásica con concatenación de sentencias mediante ;, específicamente utilizando un comando ALTER del subconjunto DDL de SQL.
5. **Herramienta Utilizada:** Explotación manual a través de la interfaz web de WebGoat.

Evidencia del Impacto

La ejecución exitosa del comando ALTER TABLE **cambió permanentemente la estructura de una tabla crítica** de la base de datos. Este ataque demuestra que un vulnerabilidad de inyección puede escalar más allá del robo o manipulación de datos:

- **Integridad Comprometida:** La estructura de los datos es ahora diferente a la diseñada y esperada por la aplicación.
- **Pérdida de Disponibilidad Potencial:** Un comando DROP TABLE o ALTER TABLE ... DROP COLUMN inyectado podría eliminar tablas o columnas esenciales, provocando que la aplicación deje de funcionar.
- **Preparación para Ataques Futuros:** Un atacante podría crear tablas o columnas maliciosas para usarlas en fases posteriores de un ataque.



7. Data Manipulation Language (DML).

Nivel de Riesgo Medio: Permite la modificación no autorizada de datos de negocio sensibles, comprometiendo directamente la integridad de la información.

Descripción de la Vulnerabilidad

La aplicación es vulnerable a Inyección SQL que permite la ejecución de comandos de **Lenguaje de Manipulación de Datos (DML)**. A diferencia del DDL (que afecta la estructura), los comandos DML (SELECT, INSERT, UPDATE, DELETE) **operan directamente sobre los datos almacenados**. En este escenario, la aplicación otorga implícitamente privilegios elevados, permitiendo a un atacante modificar registros de la base de datos sin ninguna autenticación, violando la integridad de los datos.

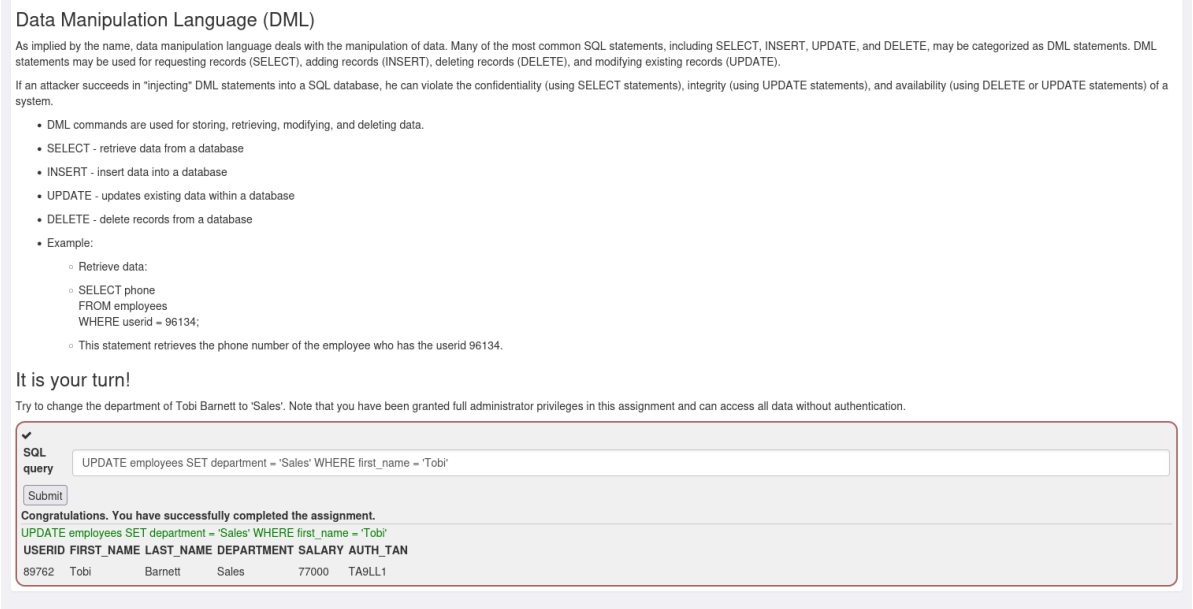
Explotación

1. **Vector de Ataque:** Campo de entrada vulnerable que permite la inyección de una sentencia SQL completa.
2. **Payload Inyectado:** UPDATE employees SET department = 'Sales' WHERE first_name = 'Tobi'
    - UPDATE employees: Comando DML que modifica registros en la tabla employees.
    - SET department = 'Sales': Establece el valor de la columna department a 'Sales' (Ventas).
    - WHERE first_name = 'Tobi': Filtra la modificación específicamente al registro del empleado llamado 'Tobi Barnett'.
3. **Técnica:** Inyección SQL clásica con concatenación de sentencias, utilizando un comando UPDATE del subconjunto DML de SQL para modificar un dato empresarial crítico (la asignación departamental).
4. **Herramienta Utilizada:** Explotación manual a través de la interfaz web de WebGoat.

Evidencia del Impacto

La ejecución exitosa del comando UPDATE **modificó permanentemente un dato operacional sensible** en la base de datos. Este ataque demuestra una violación directa de la integridad:

- **Datos de Negocio Comprometidos:** Se alteró la información de asignación de personal (departamento), lo que en un entorno real podría afectar a la nómina, la cadena de mando o la asignación de responsabilidades.
- **Privilegios Excesivos:** La aplicación concedió implícitamente permisos de escritura total sin verificar la identidad o los roles del usuario, un fallo crítico de control de acceso.



8. Reflected Cross-Site Scripting (XSS)

Nivel de Riesgo Medio: Impacta a un usuario individual a través de ingeniería social, requiere interacción y no compromete directamente datos del servidor.

Descripción de la Vulnerabilidad

La aplicación contiene un formulario donde uno de los campos de entrada no valida ni sanitiza correctamente la entrada del usuario antes de incluirla en la respuesta HTTP. Esto permite a un atacante inyectar código JavaScript malicioso que se refleja instantáneamente en la página que ve la víctima. A diferencia del XSS almacenado, el payload no se guarda en el servidor; se ejecuta en el contexto de la sesión de la víctima en el momento en que esta visita una URL manipulada.

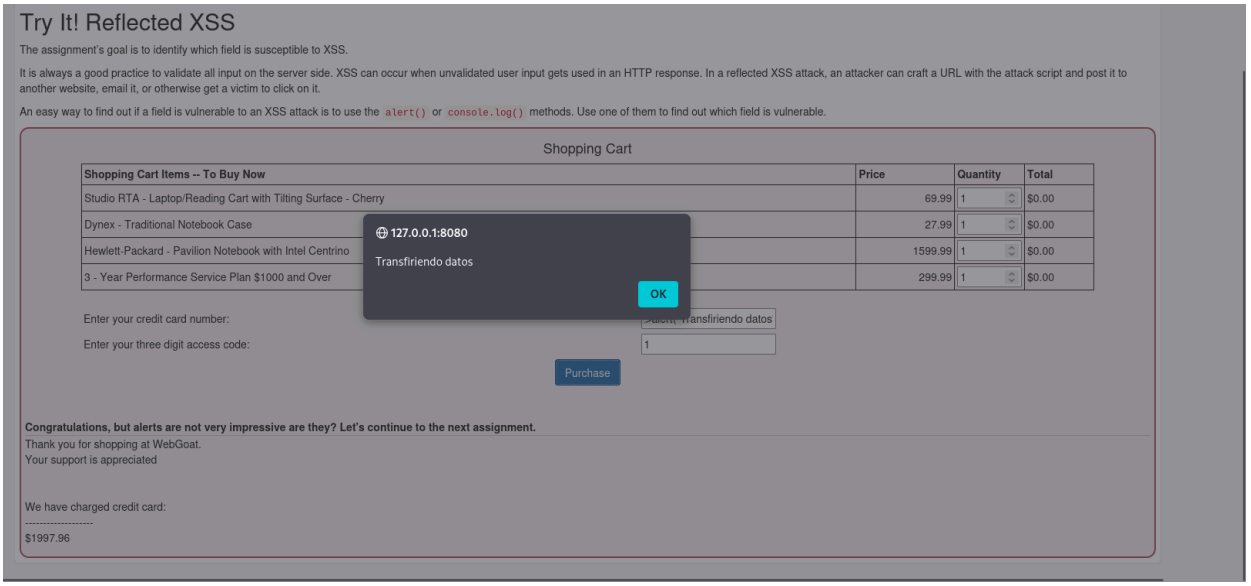
Explotación

1. **Vector de Ataque:** Campo de entrada del formulario (identificado como "Enter your three digital access code" u otro similar).
2. **Payload Inyectado:** Un script simple como <script>alert('Transferiendo datos')</script> o similar.
3. **Técnica:** Prueba de inyección básica de XSS reflejado utilizando la función alert() para verificar la ejecución de código arbitrario.
4. **Herramienta Utilizada:** Explotación manual a través de la interfaz web de WebGoat.

Evidencia del Impacto

La inyección del script resultó en la **ejecución no autorizada de código en el navegador**. La aparición del pop-up confirma que un atacante podría, en un escenario real:

- **Robar la cookie de sesión** de la víctima y secuestrar su sesión.
- **Redirigir a la víctima** a un sitio web falso.
- **Realizar acciones en nombre del usuario**.
- **Registrar las pulsaciones de teclas** dentro de la aplicación.



9. CSRF: Post a Review on Someone Else's Behalf.

Nivel de Riesgo Medio: Permite realizar acciones no autorizadas en nombre de un usuario autenticado.

Descripción de la Vulnerabilidad

La aplicación contiene un formulario para publicar comentarios o reseñas que es vulnerable a **Cross-Site Request Forgery (CSRF)**. Esta vulnerabilidad permite que un atacante engañe a un usuario autenticado para que, sin su conocimiento, envíe una petición HTTP (en este caso, un POST con un comentario) a la aplicación web. El ataque explota la confianza que la aplicación tiene en la sesión del navegador, ya que este envía automáticamente las cookies de autenticación con cada petición al dominio.

Explotación

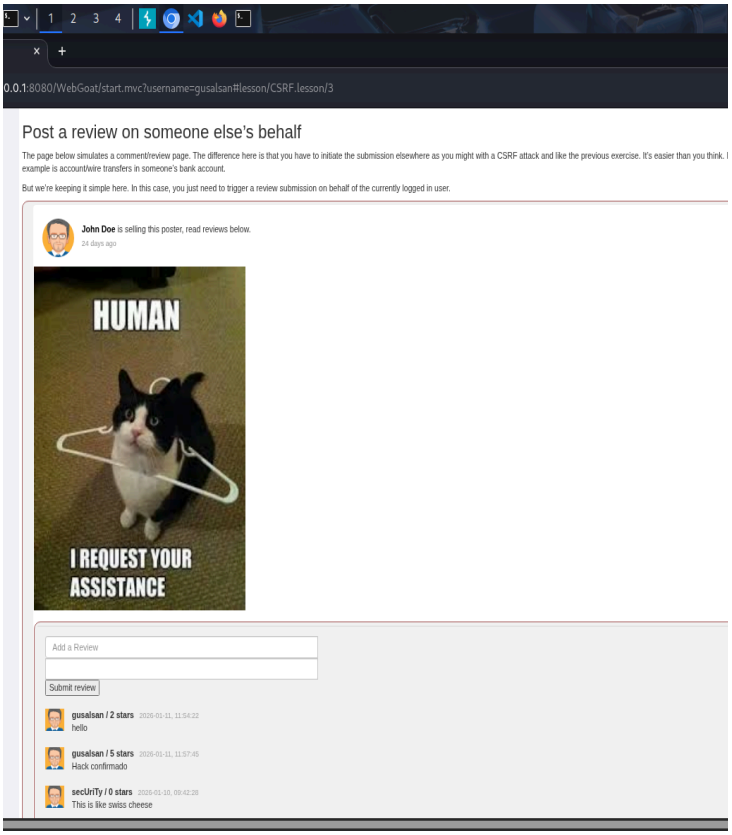
1. **Vector de Ataque:** Formulario de envío de reseñas. El ataque se ejecuta **desde una ubicación externa** (otra pestaña, un email con un enlace, una imagen cargada en un foro) que hace que el navegador de la víctima envíe la petición maliciosa.
2. **Técnica: Cross-Site Request Forgery**. Se elaboró una petición HTTP POST que replica la acción de enviar una reseña, con parámetros controlados por el atacante.
3. **Acción Maliciosa Realizada:** Se publicaron con éxito reseñas con los textos **"hello"** y **"hack confirmado"** en nombre del usuario autenticado en ese momento, sin su consentimiento.
4. **Herramienta Utilizada:** Se puede realizar manualmente creando una página HTML maliciosa o, comúnmente, utilizando las funcionalidades de **Burp Suite**.

Evidencia del Impacto

El ataque fue exitoso, y las reseñas **"hello"** y **"hack confirmado"** aparecieron publicadas en el sistema como si el usuario legítimo las hubiera escrito. Esto demuestra que un atacante podría:

- **Cambiar configuraciones de la cuenta** (email, contraseña).

- **Publicar contenido malicioso o fraudulento** en nombre del usuario, dañando su reputación.
- **Propagar el ataque** si la acción permite a su vez publicar nuevos contenidos CSRF.



10. Exploitation of a Vulnerable Third-Party Component (jQuery UI).

Nivel de Riesgo Bajo: La vulnerabilidad está en un componente de frontend ya parcheado, y su explotación directa en un entorno moderno y actualizado es limitada.

Descripción de la Vulnerabilidad

La aplicación depende de una versión antigua y conocida como vulnerable de una librería de terceros: **jQuery UI 1.10.4**. Un componente específico de esta librería no sanitizaba correctamente la entrada del usuario en el parámetro closeText, permitiendo la inyección de código HTML/JavaScript. Este es un ejemplo clásico de

que **la seguridad de una aplicación depende de todos sus componentes**, no solo del código propio. La misma aplicación, al actualizar a la versión **jQuery UI 1.12.0**, mitiga la vulnerabilidad sin cambiar una línea de su propio código.

Explotación

1. **Vector de Ataque:** Campo de entrada que controla el texto (closeText) del botón de cierre de un cuadro de diálogo de jQuery UI.
2. **Payload Inyectado:** <script>alert('XSS')</script>
3. **Técnica:** Cross-Site Scripting (XSS) reflejado, explotando una vulnerabilidad específica en jQuery UI 1.10.4 y anteriores.
4. **Herramienta Utilizada:** Explotación manual a través de la interfaz web de WebGoat.

Evidencia del Impacto y Conexión con Reconocimiento

Este hallazgo es la **prueba práctica y final** de un riesgo identificado en la **fase de reconocimiento**. La herramienta **Wappalyzer** detectó automáticamente la presencia de jQuery UI 1.10.4 en la aplicación. La explotación exitosa valida que la mera detección de un componente desactualizado puede ser un vector de ataque directo y explotable.

3.3. Post-Explotación.

La explotación exitosa de las vulnerabilidades demostró una cadena de consecuencias técnicas graves que suelen extenderse más allá del punto de entrada inicial. La pérdida de confidencialidad fue total, permitiendo el robo de bases de datos completas con información sensible. Este acceso, unido a la capacidad de ejecutar comandos DML y DDL, comprometió no solo la integridad de los datos sino también la estructura misma del sistema, pudiendo corromper su funcionamiento. Simultáneamente, fallos en los mecanismos de sesión y autenticación otorgaron a un atacante la capacidad de suplantar identidades y realizar acciones remotas en nombre de usuarios legítimos, estableciendo una posición desde la que persistir y escalar privilegios dentro del entorno comprometido.

Para contrarrestar esta progresión, la defensa debe operar en capas que combinen prevención, contención y respuesta. La base técnica es fundamental: implementar consultas parametrizadas, sanitización estricta de salida, tokens anti-CSRF y un estricto principio de mínimo privilegio en cuentas de aplicación y base de datos. Sobre esta base, es crítico desplegar mecanismos de **detección activa**, como la monitorización de consultas SQL anómalas y cambios en el esquema de la base de datos, junto con un **plan de respuesta ágil** que incluya la capacidad de revertir cambios mediante copias de seguridad validadas y procedimientos para revocar accesos comprometidos. En esencia, la seguridad efectiva reside tanto en negar el acceso inicial como en limitar drásticamente el movimiento lateral y el impacto cuando un atacante logra superar las defensas perimetrales.

3.4. Posibles mitigaciones.

Para abordar las vulnerabilidades identificadas y fortalecer la postura de seguridad general de la aplicación, se recomienda implementar las siguientes contramedidas de forma transversal y priorizada:

1. DEFENSA CONTRA INYECCIONES.

- **Parámetro Obligatorio**: **Consultas Parametrizadas**. Utilizar este mecanismo de forma estricta para **todas** las interacciones con la base de datos, tanto para datos tipo texto como numéricos. Esta es la defensa principal y más efectiva.
- **Principio de Mínimo Privilegio**: La cuenta de la aplicación en la base de datos debe operar con los **permisos estrictamente necesarios**.
- **Validación de Entrada Estricta**: En el servidor, validar que la entrada del usuario cumpla con un formato esperado utilizando listas blancas cuando sea posible.

2. GESTIÓN SEGURA DE AUTENTICACIÓN Y SESIÓN.

- **Protección de Credenciales**: **Nunca almacenar contraseñas en texto plano**. Utilizar funciones de *hash* fuertes, lentas y saladas (como bcrypt o Argon2). Para restablecimientos, usar **tokens de un solo uso y de corta vida** en lugar de preguntas de seguridad.

- **Identificadores de Sesión Robustos**: Generar IDs de sesión largos, criptográficamente aleatorios y regenerarlos tras el login. Establecer tiempos de expiración adecuados.
- **Mecanismos Anti-Fuerza Bruta**: Implementar **límite de intentos fallidos** y temporizadores de bloqueo en todas las funcionalidades de login y recuperación.
- **Control de Acceso Mandatorio**: Verificar los permisos del usuario para cada operación o acceso a datos en el servidor.

3. SEGURIDAD EN EL FRONTEND Y CONTRA ATAQUES AL CLIENTE.

- **Sanitización y Codificación de Salida**: Tratar **toda la salida generada por el usuario** que se refleje en HTML, JavaScript o CSS, codificando caracteres especiales según el contexto de salida.
- **Tokens Anti-CSRF**: Proteger todos los formularios y solicitudes state-changing (POST, PUT, DELETE) con **tokens únicos, secretos y vinculados a la sesión del usuario**, verificándolos en el servidor.

4. GESTIÓN DE DEPENDENCIAS Y CONFIGURACIÓN.

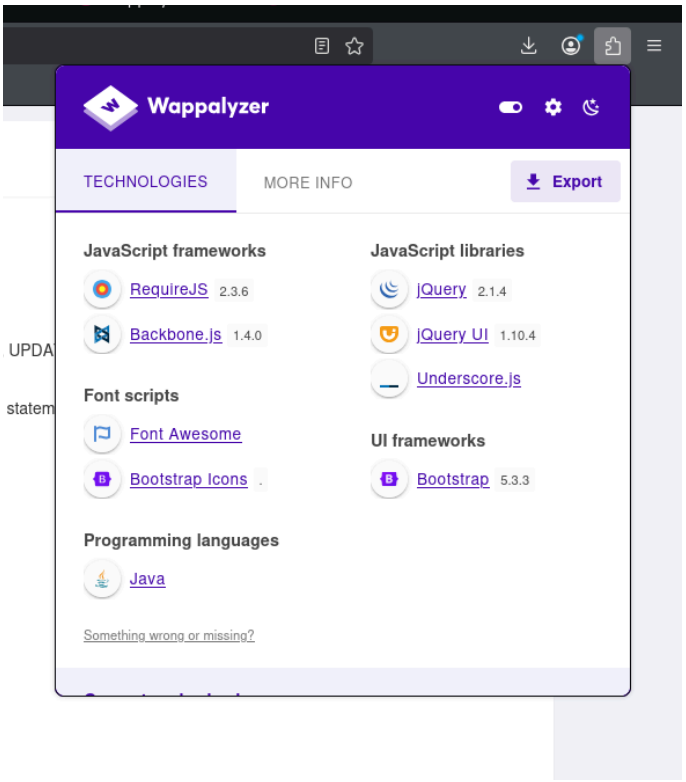
- **Inventario y Actualización Proactiva**: Mantener un **inventario actualizado de todos los componentes de terceros** (librerías, frameworks, plugins) y establecer un proceso para **monitorizar e instalar parches de seguridad** de manera prioritaria y ágil.
- **Configuración Segura por Defecto**: Asegurar que la configuración de todos los componentes (servidor de aplicaciones, base de datos, frameworks) siga las **guías de hardening** y no contenga configuraciones de debug, puertos administrativos abiertos o mensajes de error detallados expuestos al público.
- **Segregación y Segmentación**: Separar componentes y datos según su criticidad y función para limitar el alcance de un potencial compromiso.

3.5. Herramientas utilizadas.

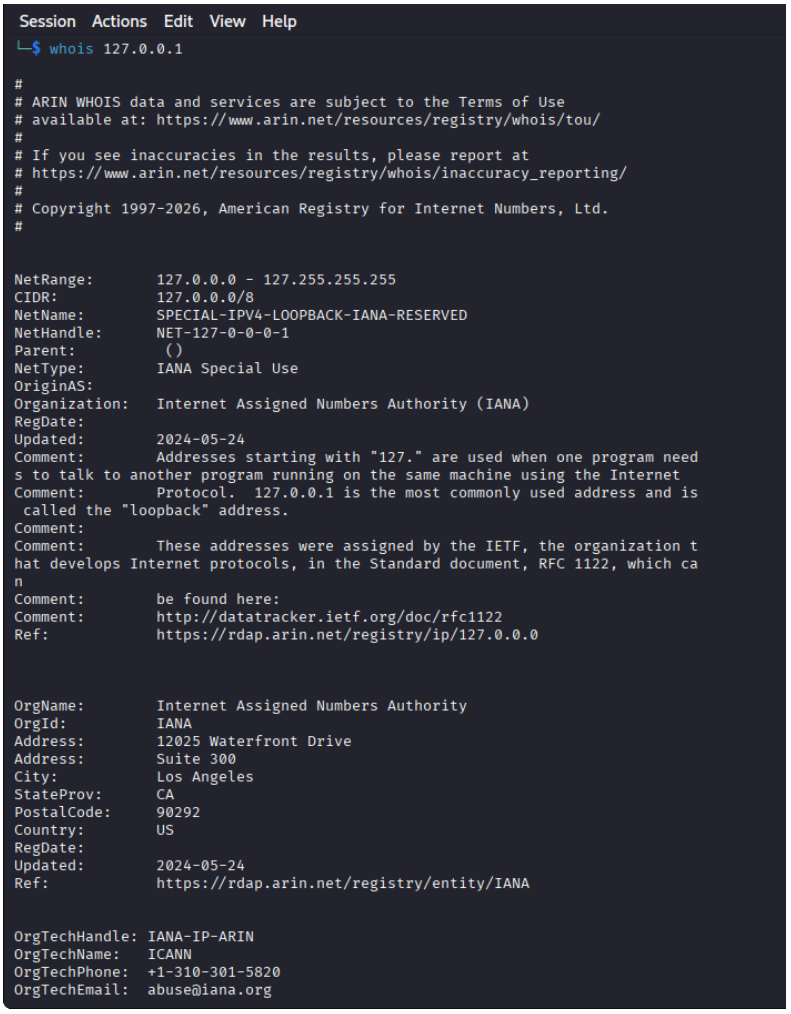
Durante la auditoría se empleó un conjunto de herramientas estándar de la industria, alineadas con la metodología de pruebas de penetración. Su uso se categorizó en fases de reconocimiento, análisis y explotación.

Reconocimiento Pasivo y Fingerprinting

- **Wappalyzer**: Para la identificación pasiva de tecnologías en la capa de aplicación. Su uso fue crucial para detectar el stack tecnológico: **Java** en el backend y, de manera crítica, librerías frontend antiguas y específicas como **jQuery 2.1.4** y **jQuery UI 1.10.4**.



- **WHOIS (comando CLI)**: Utilizada para consultar información de registro de dominios. Confirmó que el objetivo (127.0.0.1) era un entorno local sin registros públicos.



Escaneo de Red y Servicios

- **Nmap**: Herramienta fundamental para el descubrimiento de activos. Se empleó para identificar los servicios activos en el host objetivo, revelando dos instancias de **Apache Tomcat** en los puertos **8080** (aplicación principal WebGoat) y **9090** (herramienta auxiliar WebWolf), así como para inferir el sistema operativo base (**Linux**).

```
┌──(kali㉿kali)-[~]
└─$ nmap -sS -sV -O 127.0.0.1
Starting Nmap 7.95 ( https://nmap.org ) at 2026-01-13 04:15 EST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000027s latency).
Not shown: 998 closed tcp ports (reset)
PORT     STATE SERVICE VERSION
8080/tcp open  http    Apache Tomcat (language: en)
9090/tcp open  http    Apache Tomcat (language: en)
Device type: general purpose
Running: Linux 5.X|6.X
OS CPE: cpe:/o:linux:linux_kernel:5 cpe:/o:linux:linux_kernel:6
OS details: Linux 5.0 - 6.2
Network Distance: 0 hops

OS and Service detection performed. Please report any incorrect results at ht
tps://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 7.63 seconds
```

Fuzzing Web y Descubrimiento de Contenido

- **DIRB**: Escáner de fuzzing web utilizado para descubrir directorios, archivos y rutas de aplicación no vinculadas públicamente. Su ejecución permitió enumerar endpoints clave como /login, /registration y confirmar la estructura de aplicación Java mediante la detección del directorio /WEB-INF, sin encontrar archivos sensibles expuestos de forma obvia.

```
┌──(kali㉿kali)-[~]
└─$ dirb http://127.0.0.1:8080/WebGoat/

-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Tue Jan 13 04:21:40 2026
URL_BASE: http://127.0.0.1:8080/WebGoat/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://127.0.0.1:8080/WebGoat/ ----

+ http://127.0.0.1:8080/WebGoat/css (CODE:302|SIZE:0)
+ http://127.0.0.1:8080/WebGoat/favicon.ico (CODE:302|SIZE:0)
+ http://127.0.0.1:8080/WebGoat/fonts (CODE:302|SIZE:0)
+ http://127.0.0.1:8080/WebGoat/images (CODE:302|SIZE:0)
+ http://127.0.0.1:8080/WebGoat/js (CODE:302|SIZE:0)
+ http://127.0.0.1:8080/WebGoat/login (CODE:200|SIZE:1929)
+ http://127.0.0.1:8080/WebGoat/logout (CODE:302|SIZE:0)
+ http://127.0.0.1:8080/WebGoat/meta-inf (CODE:302|SIZE:0)
+ http://127.0.0.1:8080/WebGoat/META-INF (CODE:302|SIZE:0)
+ http://127.0.0.1:8080/WebGoat/plugins (CODE:302|SIZE:0)
+ http://127.0.0.1:8080/WebGoat/registration (CODE:200|SIZE:4188)
+ http://127.0.0.1:8080/WebGoat/web-inf (CODE:302|SIZE:0)
+ http://127.0.0.1:8080/WebGoat/WEB-INF (CODE:302|SIZE:0)


-----------------
END_TIME: Tue Jan 13 04:21:43 2026
DOWNLOADED: 4612 - FOUND: 13
```

Proxy de Intercepción y Análisis de Tráfico

- **Burp Suite Community Edition**: Configurado como proxy local, fue la herramienta central para la interceptación, modificación, reenvío y análisis detallado de todo el tráfico HTTP/HTTPS entre el navegador y la aplicación WebGoat. Permitió estudiar parámetros, cookies, sesiones y realizar pruebas de manipulación manual de peticiones, sirviendo como plataforma de análisis durante todas las fases de prueba.

Entorno de Pruebas y Explotación Manual

- **Interfaz Web de WebGoat**: El entorno objetivo en sí mismo, que sirvió como la plataforma principal para la **explotación manual y dirigida** de las vulnerabilidades deliberadas. Todas las pruebas de inyección SQL, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF) y lógica de negocio se ejecutaron a través de su interfaz.
- **Navegador Web y Consola de Desarrollo (F12)**: Cliente principal para interactuar con la aplicación. La consola de desarrollador se utilizó para depuración, verificación de la ejecución de código JavaScript y análisis de respuestas del servidor.