Instituto Superior Universitario Tecnológico del Azuay

Nombre: Steven Vallejo

Carrera: Tecnología Superior en Ciberseguridad - V4A

Tema: Manual de pentesting - Viajes Seguros Web

Fecha del informe: 21/07/2025

Periodo abril - agosto 2025

Vulnerabilidades implementadas en "Viajes Seguros S.A."

1. A01: Broken Access Control (Control de Acceso Roto)

- **Tipo específico:** Referencia Insegura a Objeto Directo (IDOR).
- **Ubicación:** profile.php
- **Resumen:** La página muestra la información del perfil basándose en el **UUID** proporcionado en la URL (profile.php?uuid=...). Sin embargo, el script **nunca verifica si el usuario que ha iniciado sesión es el propietario de ese UUID**. Esto permite que un atacante, después de obtener el UUID de otra víctima (por ejemplo, a través de la Inyección SQL), pueda ver sus datos privados simplemente cambiando el UUID en la URL.

2. A02: Cryptographic Failures (Fallos Criptográficos)

- **Tipo Específico:** Fuga de Token de Reseteo de Contraseña.
- **Ubicación:** forgot_password.php (donde se origina el fallo) y logs/password_resets.log (donde se expone).
- **Resumen:** Aunque la aplicación genera un token de reseteo de contraseña criptográficamente seguro, comete un fallo crítico al escribir este token sensible en un archivo de log (password_resets.log) por un supuesto "propósito de depuración". Combinado con la mala configuración del servidor que permite el listado de directorios, este archivo de log se vuelve públicamente accesible, permitiendo a un atacante robar los tokens y secuestrar las cuentas de otros usuarios.

3. A03: Injection SQL (Inyección)

- **Tipo específico:** Inyección SQL Ciega (Time-based y Boolean-based).
- **Ubicación:** search_results.php (la vulnerabilidad se origina en el campo de búsqueda del index.php).
- **Resumen:** El script toma el término de búsqueda del usuario y lo inserta directamente en una consulta SQL LIKE **sin ningún tipo de saneamiento o consultas preparadas**. Se han implementado manejadores de errores (try-catch) para que la aplicación no muestre errores de sintaxis SQL, convirtiéndola en una inyección "ciega". Un atacante está forzado a usar técnicas avanzadas (basadas en tiempo o contenido) para confirmar la vulnerabilidad y exfiltrar datos de la base de datos.

4. A04: Insecure Design (Diseño Inseguro)

- **Tipo específico:** Confianza Excesiva en los Datos del Cliente (Manipulación de Parámetros).
- **Ubicación:** El flujo entre confirm_purchase.php (donde se origina) y purchase.php (donde se explota).
- **Resumen:** El proceso de compra se realiza en dos pasos. En el segundo paso, la página de confirmación envía el precio final del viaje al script de procesamiento (purchase.php) como un **parámetro oculto en el formulario**. El servidor, por un fallo de diseño, **confía ciegamente en este precio** y lo inserta en la base de datos sin volver a verificarlo. Esto permite a un atacante interceptar la petición y modificar el parámetro final_price a cualquier valor (ej. 1.00), cometiendo fraude.

5. A05: Security Misconfiguration (Configuración de Seguridad Incorrecta)

- **Tipo específico:** Listado de directorios habilitado.
- **Ubicación:** La configuración del servidor web, activada a través del archivo .htaccess en la raíz del proyecto.
- **Resumen:** El servidor está configurado con la opción Options +Indexes, lo que deshabilita la protección por defecto que impide a los usuarios ver el contenido de los directorios que no tienen un archivo index. Esto permite a cualquier persona navegar a directorios como /includes/ o /logs/ y ver una lista completa de todos los archivos que contienen, facilitando el descubrimiento de archivos sensibles como db.php o, en nuestro caso, el crítico password_resets.log.

IMPORTANTE: Tener instalado “sqlmap”, Burp Suite o OWASP ZAP

A01 - Broken Access Control

Objetivo: Aprender a ejecutar una cadena de ataque compleja, utilizando una vulnerabilidad de Inyección SQL para obtener los identificadores necesarios para explotar una vulnerabilidad de IDOR.

Escenario: Eres un pentester que ha iniciado sesión con una cuenta estándar. Has descubierto que los perfiles de usuario se acceden a través de UUIDs no predecibles (profile.php?uuid=...). Tu búsqueda inicial de endpoints de API obvios que filtren estos UUIDs ha fracasado. El sistema parece robusto a primera vista. Sin embargo, ya has confirmado la existencia de una **Inyección SQL Ciega** en el buscador.

Reflexión: "No puedo adivinar los UUIDs y no hay una fuga de información evidente. Pero tengo una Inyección SQL Ciega. Eso significa que puedo hacerle cualquier pregunta a la base de datos. En lugar de pedir contraseñas, ¿puedo pedirle que me revele los UUIDs de todos los usuarios?"

Paso 1: Confirmación de la Inyección SQL Ciega (Requisito Previo)

- Este paso es idéntico al del módulo A03. El estudiante debe haber confirmado primero que puede controlar la base de datos a través de una inyección basada en contenido o en tiempo en search_results.php.

Paso 2: Exfiltración de UUIDs usando sqlmap

IMPORTANTE: Haber realizado el paso 1 y paso 2 de la vulnerabilidad A02 para llegar a este punto

Reflexión: "Ahora voy a apuntar mi herramienta de inyección SQL, sqlmap, a la tabla users, pero mi objetivo esta vez no son las contraseñas. Mi objetivo son los identificadores uuid que necesito para mi siguiente ataque."

1. **Abre una Terminal o Símbolo del sistema.**
2. **Ejecuta** un comando de sqlmap que se centre específicamente en extraer las columnas username y uuid de la tabla users.
    - **Comando:**

      ```
      python sqlmap.py -u"http://localhost/viajes_seguros/search_results.php?destination=Egipto" -D viajes_db -T users -C "username,uuid" --dump
      ```

- **Desglose del Comando:**
  - -u "...": La URL vulnerable.
  - -D viajes_db: La base de datos objetivo.
  - -T users: La tabla objetivo.
  - -C "username,uuid": Le decimos a sqlmap que ignore el resto y se concentre en exfiltrar solo estas dos columnas.
  - --dump: Extrae y muestra el contenido.

3. **Analiza los resultados:** sqlmap te devolverá una tabla limpia con la información que necesitas.

username	**uuid**
admin	**[UUID_DEL_ADMIN]**
X	[UUID_DE_X]

Conclusión: "Hemos utilizado una vulnerabilidad de Inyección SQL no para robar contraseñas, sino como una herramienta de reconocimiento para obtener los identificadores secretos (UUIDs) que necesitamos para nuestro próximo ataque. Esta es una técnica avanzada de encadenamiento de vulnerabilidades."

Paso 3: Explotación del IDOR con la información obtenida

Reflexión: "Tengo el UUID del administrador. Ahora solo tengo que usarlo en el endpoint de perfiles y ver si la validación de autorización falla."

1. **Obtén el UUID del objetivo:** De la salida de sqlmap, copia el uuid del usuario admin.
2. **Construye la URL de ataque:**
    - Asegúrate de estar logueado con tu cuenta de prueba (no la de admin).
    - Navega a: http://localhost/viajes_seguros/profile.php?uuid=[UUID_DEL_ADMIN_PEGADO_AQUÍ]

3. **Verificación del impacto:**
    - **Resultado:** La página carga el perfil del admin, mostrando sus datos privados.

Conclusión: "Hemos completado la cadena de ataque. Demostramos que, aunque los desarrolladores intentaron proteger los perfiles con UUIDs, otra vulnerabilidad en una parte completamente diferente de la aplicación (la búsqueda) permitió a un atacante obtener esos UUIDs y eludir la protección.."

A02 - Cryptographic Failures

Objetivo: Aprender a identificar y explotar el uso de criptografía débil. En este escenario, realizaremos una **cadena de explotación completa**: usaremos una Inyección SQL Ciega para mapear la base de datos y obtener la información de nuestro objetivo, y luego explotaremos una fuga de información para tomar control de su cuenta.

Escenario: Eres un pentester que acaba de confirmar dos vulnerabilidades preliminares:

1. **Inyección SQL Ciega (A03)** en el buscador.
2. **Listado de Directorios (A05)** en la carpeta /logs/.

Ahora, vas a utilizar estas vulnerabilidades de forma combinada para lograr un objetivo de alto impacto: comprometer la cuenta del administrador.

Paso 1: Enumeración de la base de datos con sqlmap

Reflexión: "Confirmé una Inyección SQL Ciega, pero no sé nada sobre la estructura de la base de datos. No sé cómo se llama, qué tablas contiene, ni qué columnas son interesantes. Mi primer paso es usar sqlmap para que haga este trabajo de reconocimiento por mí, utilizando la vulnerabilidad que ya encontré."

1. **Abre una Terminal o Símbolo del Sistema.** Asegúrate de estar en el directorio donde tienes sqlmap.
2. **Verificar inyección ciega basada en contenido**

    1. **Establecer una línea base:**
        - Ve a tu navegador y busca un destino que **SÍ exista**, por ejemplo, "Egipto". La página te muestra la tarjeta de Egipto. Esta es nuestra respuesta para "verdadero".
        - Ahora, busca algo que **NO exista**, como "Atlantida". La página te muestra el mensaje "No se encontraron resultados". Esta es nuestra respuesta para "falso".

2. **Formular la pregunta y el payload:**

    - Queremos hacer una pregunta simple que sepamos que es verdadera, por ejemplo: 1=1.
    - Construimos el payload: Egipto' AND 1=1#
    - **Significa:** "Busca un destino que se llame 'Egipto' Y donde 1 sea igual a 1". Como ambas condiciones son verdaderas, la consulta debería devolver el resultado de "Egipto".

3. **Ejecutar la prueba (en el navegador de Burp Suite utilizando el Repetidor):**

    - Ve a la barra de direcciones de tu navegador y pega la URL codificada: http://localhost/viajes_seguros/search_results.php?destination=Egipto%27%20AND%201=1%23
    - **Resultado:** ¡La página muestra la tarjeta de "Egipto"! Esto significa que la condición "verdadera" funcionó.
    - **¿Cómo codificar?** Haz clic derecho > **Convert selection > URL > URL-encode key characters** (o presiona Ctrl+U en Windows/Linux, Cmd+U en Mac).

      IMPORTANTE: Si encuentras dudas de este proceso, se encuentra más detallado en el proceso **A03 - Inyección SQL Ciega (Blind SQLi)**

4. **Probar la Condición Falsa:**

    - Ahora, probemos con una condición falsa: 1=2.
    - Construimos el payload: Egipto' AND 1=2#
    - Pega la URL codificada: http://localhost/viajes_seguros/search_results.php?destination=Egipto%27%20AND%201=2%23
    - **Resultado:** La página ahora muestra "No se encontraron resultados"
    - **Conclusión:** Acabas de confirmar una **Inyección SQL Ciega basada en contenido**. Puedes hacer cualquier pregunta a la base de datos y saber si la respuesta es verdadera o falsa basándose en si la página muestra la tarjeta de Egipto o no.

3. **Descubrir el nombre de la base de datos:**
    - **Ejecuta** sqlmap con la opción --dbs para que intente listar todas las bases de datos a las que tiene acceso.

Comando:

```
python sqlmap.py -u "http://localhost/viajes_seguros/search_results.php?destination=Egipto" --dbs
```

- **Análisis del resultado:** sqlmap trabajará y, después de confirmar la vulnerabilidad, te mostrará una lista de bases de datos disponibles. Entre ellas, verás una que destaca: viajes_db.

- Se utiliza **destination=Egipto** como un valor verdadero que nos permite
- **Conclusión:** "Sabemos que la base de datos de la aplicación se llama viajes_db."

4. **Descubrir las tablas:**
    - Ahora que conoces el nombre de la base de datos, úsalo para listar las tablas que contiene.

Comando:

```
python sqlmap.py -u "http://localhost/viajes_seguros/search_results.php?destination=Egipto" -D viajes_db --tables
```

- **Análisis del resultado:** sqlmap te mostrará las tablas dentro de viajes_db. Verás destinations, purchases, y la más prometedora: users.
- **Tu conclusión didáctica:** "Hemos encontrado la tabla de usuarios. Este es casi siempre el objetivo principal en una base de datos. El siguiente paso es ver qué tipo de información contiene."

5. **Descubrir las columnas de la tabla users:**
    - **Acción:** Pídele a sqlmap que te muestre la estructura de la tabla users.

```
python sqlmap.py -u "http://localhost/viajes_seguros/search_results.php?destination=Egipto" -D viajes_db -T users --columns
```

    - **Análisis del resultado:** Verás una lista detallada de las columnas, incluyendo id, uuid, username, email, password, y is_admin.
    - **Conclusión:** "El mapa está completo. La columna is_admin es la clave para identificar a nuestro objetivo. Ahora, vamos a extraer la información."

Paso 2: Obtención de inteligencia - Exfiltrar los datos del objetivo

Reflexión: "Tengo el mapa completo de la tabla users. Ahora voy a extraer la información específica que necesito para mi siguiente ataque: el nombre de usuario y el correo electrónico de cualquier cuenta que sea de administrador."

1. **Volcar (Dump) el contenido relevante:**
    - **Acción:** Pídele a sqlmap que extraiga y te muestre el contenido de las columnas username, email y is_admin.

```
python sqlmap.py -u "http://localhost/viajes_seguros/search_results.php?destination=Egipto" -D viajes_db -T users -C "username,email,is_admin" --dump
```

2. **Analiza los datos exfiltrados:** sqlmap te mostrará una tabla con los datos solicitados.

username	email	is_admin
admin	**admin@viajesseguros.com**	**1**

Conclusión: La exfiltración ha sido un éxito. Hemos identificado positivamente que el usuario admin, cuyo correo es admin@viajesseguros.com, tiene privilegios de administrador. Este es nuestro objetivo.

Paso 3: Explotación de la fuga de Token de Reseteo

Reflexión: "Ahora que tengo el correo del objetivo, voy a usar la otra vulnerabilidad que encontré (el log expuesto) para secuestrar su cuenta. Voy a forzar al sistema a generar un token para el admin y lo capturaré del archivo de log."

1. **Provoca la generación del Token:**
    - Ve a la página forgot_password.php.
    - Introduce el correo del administrador que acabas de descubrir: admin@viajesseguros.com.
    - Haz clic en "Enviar Instrucciones".

2. **Captura el Token filtrado:**
    - Navega directamente al archivo de log: http://localhost/viajes_seguros/logs/password_resets.log.
    - Copia el token que aparece en la última línea, asociado al usuario admin.

3. **Ejecuta el secuestro de cuenta:**
    - Construye la URL de ataque: http://localhost/viajes_seguros/reset_password.php?token=[PEGA_EL_TOKEN_DEL_ADMIN_AQUÍ]
    - Navega a esa URL y establece una nueva contraseña.

4. **Verificación final:**
    - Ve a login.php e inicia sesión como admin con la nueva contraseña.
    - **Conclusión:** "Hemos completado la cadena de ataque. Usamos una Inyección SQL Ciega para mapear la base de datos y obtener la identidad de nuestro objetivo. Luego, usamos esa información para explotar una Configuración de Seguridad Incorrecta que filtraba tokens de reseteo, lo que representa un Fallo Criptográfico en la protección de un secreto. El resultado es un compromiso total de la cuenta del administrador."

A03 - Inyección SQL Ciega (Blind SQLi)

Objetivo: Aprender a identificar, verificar y comprender el impacto de una Inyección SQL Ciega (Blind SQL Injection) basada en tiempo. Esta es una vulnerabilidad muy sigilosa, ya que la aplicación no nos dará pistas obvias como mensajes de error.

Paso 1: Preparación del ataque y análisis de la petición base

Reflexión: "Antes de atacar, necesito entender cómo funciona la comunicación normal. Voy a capturar una búsqueda legítima para tener una plantilla sobre la cual trabajar."

1. **Abre Burp Suite** y asegúrate de que tu navegador esté configurado para usarlo como proxy. La intercepción debe estar desactivada (Proxy > Intercept is off).
2. En la aplicación web, realiza una búsqueda normal. Por ejemplo, busca "China".
3. Ve a la pestaña **Proxy > HTTP history** en Burp Suite.
4. Busca la petición que acabas de hacer. Se verá así:
   GET /viajes_seguros/search_results.php?destination=China HTTP/1.1
5. **Envía esta petición a Burp Repeater.** Haz clic derecho sobre ella y selecciona **"Send to Repeater"**. El Repeater es nuestro "laboratorio" personal, donde podemos modificar y reenviar esta petición una y otra vez para ver cómo reacciona el servidor.

Paso 2: Sondeo de la vulnerabilidad - ¿La Aplicación es "Ciega"?

Reflexión: "Ahora voy a intentar 'romper' la consulta. Si veo un error SQL detallado, es una inyección basada en error. Si la página simplemente se queda en blanco o no muestra resultados, podría ser una inyección 'ciega', lo que significa que es vulnerable pero está ocultando las pruebas."

1. Ve a la pestaña **Repeater**.
2. En la línea de la petición, modifica el parámetro destination. Cambia China por China'.
   - GET /viajes_seguros/search_results.php?destination=China' HTTP/1.1
3. Haz clic en **"Send"**.
4. **Analiza la Respuesta:** En el panel de "Response", verás un código 200 OK, pero si miras la pestaña "Render", la página estará vacía o mostrará un mensaje de "No se encontraron resultados".
   - **Conclusión:** La aplicación no nos ha regalado un error detallado. Significa que el desarrollador intentó manejar los errores, pero no necesariamente solucionó la vulnerabilidad de raíz. Nuestra hipótesis ahora es que tenemos una **Inyección SQL Ciega**.

Paso 3: Verificación con una carga útil basada en tiempo

Reflexión: "Si la aplicación no me 'habla' con datos o errores, la forzaré a hablarme con *tiempo*. Le pediré a la base de datos que espere 5 segundos. Si la página tarda 5 segundos en cargar, significa que mi comando fue ejecutado. El tiempo se convierte en mi canal de comunicación."

1. En el Repeater, modifica de nuevo el parámetro destination. Usa este payload completo:
   China' AND SLEEP(5)#
   Se verá así
   GET /viajes_seguros/search_results.php?destination=China' AND SLEEP(5)# HTTP/1.1
2. **Codifica el Payload:** Los servidores web a menudo bloquean caracteres como ' y # en la URL. Debemos codificarlos para que pasen desapercibidos.
   - Selecciona con el ratón **solo la parte que inyectaste**: ***' AND SLEEP(5)#***
   - Haz clic derecho > **Convert selection > URL > URL-encode key characters** (o presiona Ctrl+U en Windows/Linux, Cmd+U en Mac).
   - Tu payload se transformará en: ***China%27%20AND%20SLEEP(5)%23***
   - La línea completa se verá así: GET ***/viajes_seguros/search_results.php?destination=China%27%20AND%20SLEEP(5)%23 HTTP/1.1***

3. **Ejecuta el ataque:** Haz clic en **"Send"** y observa inmediatamente la esquina inferior derecha de la ventana de Response.
4. **Analiza el temporizador:** Verás que la respuesta tarda en llegar. El temporizador mostrará un valor **superior a 5000 milisegundos**.
   - **Conclusión:** Se ha demostrado de forma irrefutable que el servidor es vulnerable a una Inyección SQL Ciega. El retraso de 5 segundos es la "respuesta" de la base de datos confirmando que ejecutó tu comando.

A04 - Diseño Inseguro

Objetivo: Manipular la lógica de negocio para comprar un viaje a un precio fraudulento.

- **Reflexión:** "El proceso de compra en dos pasos es sospechoso. El primer paso me lleva a una página de confirmación. El segundo paso envía la confirmación. Es posible que en el segundo paso, el servidor confíe en los datos que mi navegador le envía, en lugar de verificar el precio de nuevo."

Paso 1: Manipulación de parámetros del cliente

1. **Análisis del flujo:**
    - Inicia sesión. Ve a la página de un destino caro (ej. Egipto, $2,150.50).
    - Haz clic en "Reservar Viaje" para llegar a confirm_purchase.php.
    - En Burp Suite, activa la intercepción (**Proxy > Intercept is on**).
    - En la página, haz clic en **"Confirmar y Pagar"**.

2. **Intercepción y modificación:**
    - Burp Suite capturará la petición POST a purchase.php.
    - Observa el cuerpo de la petición (pestaña **Proxy > Intercept > Body**). Verás los parámetros: destination_id=6 y final_price=2150.50.
    - **Hipótesis:** El servidor confía en este parámetro final_price.
    - **Acción:** Cambia el valor de final_price a 1.00 (o incluso 0.01).
    - Haz clic en **"Forward"** para enviar la petición manipulada. Desactiva la intercepción.

3. **Verificación del fraude:**
    - Serás redirigido a tu panel de control.
    - Busca la nueva tarjeta del viaje a Egipto.
    - **Resultado final:** La tarjeta muestra **"Precio Pagado: $1.00"**.
    - **Conclusión:** Has explotado un fallo de diseño crítico, demostrando que la confianza del servidor en los datos del cliente puede ser abusada para cometer fraude.

A05 - Security Misconfiguration

Objetivo: Encontrar fallos de configuración que expongan información sensible.

- **Reflexión:** "Antes de lanzar ataques complejos, siempre busco errores básicos. Los desarrolladores a menudo olvidan proteger directorios o dejan archivos de depuración expuestos."

Paso 1: Listado de directorios (Directory Listing)

1. **Hipótesis:** Basado en el mapa del sitio en **Burp Suite** -> **Target** -> **Site Map**. Se encuentran directorios como /includes/ y /css/ son interesantes. Si no están bien configurados, el servidor podría mostrarme su contenido.
2. **Prueba de explotación:**
    - En tu navegador, navega a http://localhost/viajes_seguros/includes/.
    - **Resultado:** En lugar de un error 403 Forbidden, ves una lista de archivos. Esto es una vulnerabilidad.

3. **Descubrimiento crítico:** Ahora prueba con la carpeta logs/, que no aparece en el mapa del sitio pero es un nombre común.
    - Navega a http://localhost/viajes_seguros/logs/.
    - **Resultado:** Ves un archivo: password_resets.log. El nombre es extremadamente sospechoso.

4. **Explotación:**
    - Haz clic en password_resets.log.
    - **Conclusión:** Has encontrado un log que expone tokens de reseteo de contraseña.