Introdicción al Hacking Ético

Lic. Bruno Zappellini Apu. Emiliano De Marco A. German Bianchini Apu. Lucas. Krmpotic Apu. Maximiliano Aguila

2019

Section 4

Unidad 4: Pentesting de Aplicaciones Web

- Las aplicaciones web se han convertido desde hace varios años, en una de las formas preferidas por los usuarios de interactuar con servicios en internet. No obstante, tambien se ha convertido en una de las principales vías de ataque para delincuentes informaticos. Por este motivo es importante conocer exactamente cuales son las principales técnicas y vectores de ataque que permiten detectar y ecplotar vulnerabilidades en Aplicaciones Web.

Antes de Comenzar:

Algunos conceptos basicos

- Arquitectura CLIENTE - SERVIDOR
- Aplicaciones Web
- Tipos de Aplicaciones Web

CLIENTE - SERVIDOR

- El modelo cliente-servidor, describe el proceso de interacción entre la computadora local (el cliente) y la remota (el servidor).

Proceso:

- El cliente le hace peticiones (requests) al servidor, el cual procesa dicho requerimiento y retorna los resultados al cliente apropiado.
- Por lo general, los clientes y los servidores se comunican entre sí a través de una red.

Arquitectura Basica CLIENTE - SERVIDOR

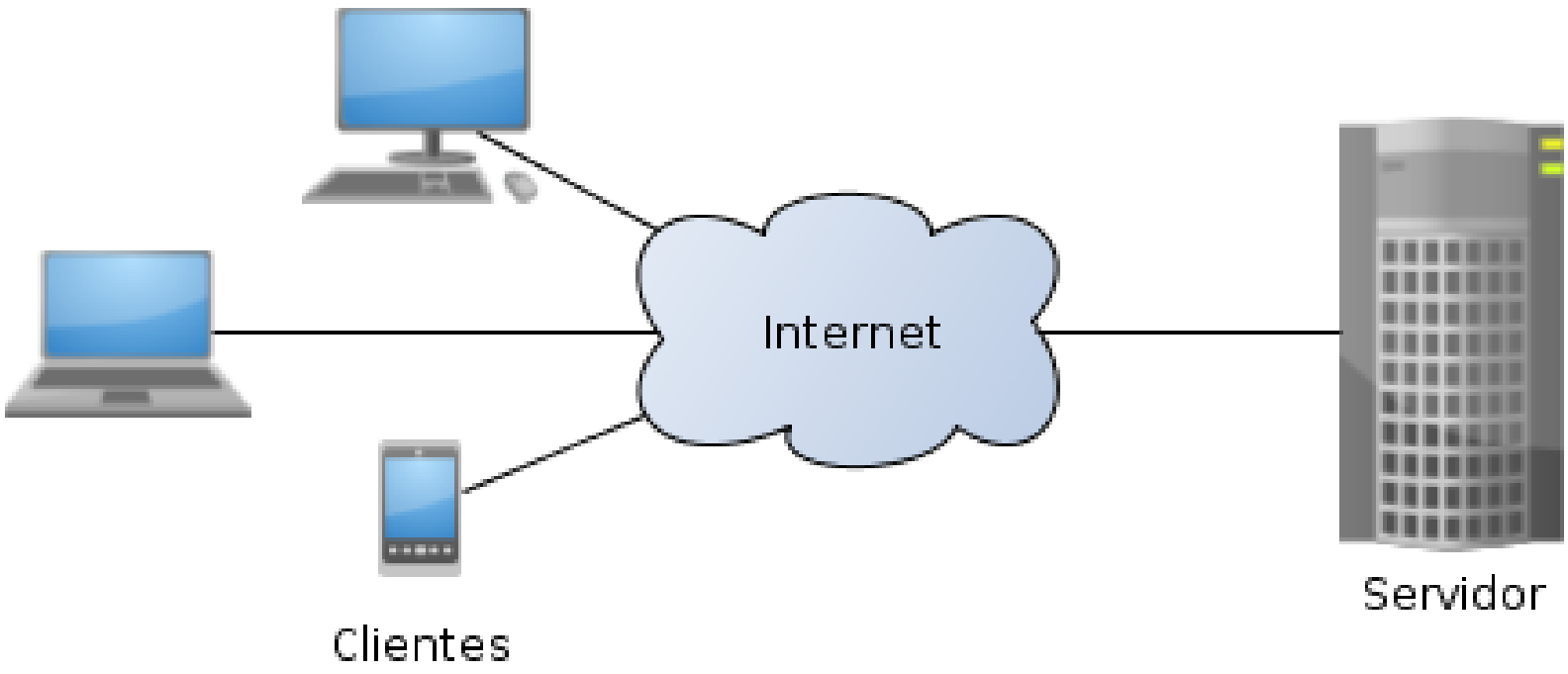

Figura 1: Cliente - Servidor

Aplicaciones Web

- Se denomina **aplicación web** a aquellas herramientas que los usuarios pueden utilizar accediendo a un servidor web a través de internet o de una intranet mediante un navegador.
- Las aplicaciones web son populares debido a lo práctico del navegador web como cliente ligero, a la independencia del sistema operativo, así como a la facilidad para actualizar y mantener aplicaciones web sin distribuir e instalar software a miles de usuarios potenciales.

Figura 2: app-web

Tipos de Aplicaciones Web

Estáticas

Una Web estática es aquella página enfocada principalmente a mostrar una información permanente, donde el navegante se limita a obtener dicha información, sin poder interactuar con la página web visitada. La mayor interactividad en este tipo de aplicaciones es mediante hipervinculos.

- Las paginas se crean mediante HTML.

Dinámicas

Una web dinámica es aquella que contiene funcionalidad dentro de la propia web, otorgando mayor interactividad y funcionalidad al usuario.

- Se construyen haciendo uso de otros lenguajes de programación (PHP, Python, etc), con lo cual podemos definir las funciones y características que se deben cumplir de acuerdo a nuestras necesidades.

Seguridad en Aplicaciones web

¿Por que es importante?

- En la actualidad el crecimiento de internet ha impactado directamente en la seguridad de la información manejada cotidianamente. Sitios de comercio electrónico, servicios, bancos e incluso redes sociales contienen información sensible que en la mayoría de los casos resulta ser muy importante.
- Dado que estas aplicaciones están disponibles desde cualquier lugar y por cualquier persona, son particularmente vulnerables a los ataques de cibercriminales.
- Uno de los puntos más críticos de la seguridad en Internet son las herramientas que interactúan de forma directa con los usuarios.

- En Writing Secure Code, Howard y LeBlanc remarcan que el sitio web del departamento de Cyber Crimen del Gobierno de Estados Unidos detalla casos criminales en los que las organizaciones sufren perdidas que típicamente exceden los USD$100.000.

OWASP

Open Web Application Security Project.

"... es una comunidad abierta dedicada a habilitar a las organizaciones para desarrollar, comprar y mantener aplicaciones confiables. Todas la herramientas, documentos, foros y capítulos de OWASP son gratuitos y abierto a cualquiera interesado en mejorar la seguridad de aplicaciones. Abogamos por resolver la seguridad de aplicaciones como un problema de gente, procesos y tecnología porque las soluciones más efectivas incluyen mejoras en todas estas áreas."

OWASP Framework

OWASP propone un framework de testing de aplicaciones como marco de referencia que comprende técnicas y tareas apropiadas para las fases de desarrollo del software, con el fin de ayudar a las organizaciones a construir sus propios procesos de testing más completos y orientados.

Etapas Framework OWASP

El framework propuesto por OWASP consiste en las siguientes etapas:

- Previo al desarrollo Definir politicas, estandares y procesos de documentación adecuados. En esta etapa se hace enfasis a la importancia de la documentación, a las medidas y a las métricas definidas para usarse a lo largo del ciclo de vida del producto.

- Durante el análisis y diseño: Se marca la importancia de los requerimientos de seguridad durante el proceso de definición de los requerimientos. Los mismos deben testearse, probando si las asunciones hechas son correctas y si existen huecos en su definición.
- Durante el desarrollo Se centra en el análisis del código en busca de defectos de seguridad.
- Durante el deployment Se sugiere la realización de pentestings en el sistema.

- Mantenimiento y operaciones Sugiere definir procesos para la revisión de como se administra tanto lo operacional como la infraestructura de la aplicación. A su vez, propone chequeos periodicos para la revisión de la seguridad del sistema y revisiones de los cambios realizados en el mismo con el fin de que estos no introduzcan brechas de seguridad.

TOP 10 OWASP

OWASP Top 10 - 2017
A1:2017-Injection
A2:2017-Broken Authentication
A3:2017-Sensitive Data Exposure
A4:2017-XML External Entities (XXE)
A5:2017-Broken Access Control
A6:2017-Security Misconfiguration
A7:2017-Cross-Site Scripting (XSS)
A8:2017-Insecure Deserialization
A9:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring

Figura 3: top-10

Injection

La Inyección consiste en engañar al interprete de una aplicación web con el fin de que este ejecute comandos no intencionados o nos de acceso a datos no autorizados. Existen multiples tipos de Inyección, siendo la más común la Inyección SQL.

Ejemplo: Supongamos que tenemos un input donde el usuario debe ingresar su nombre de usuario y contraseña con el fin de averiguar algún dato personal tal como su nombre. La consulta que espera ejecutar el interprete es:

```
sql = "SELECT nombre FROM users WHERE uname ='" +
uname + "' AND password = '" + password + ""'
database.execute(sql)
```

- Si bien la aplicacion espera recibir algo como:

uname = Bob7
password = 123456

- y la query se completaría como:

"SELECT nombre FROM users WHERE uname ='Bob7' AND password = '123456' "

- Un usuario mal intencionado podría mandar algo como

uname = Bob7
password = 0' OR 1=1 #

- Lo cual se reemplazaría como:

"SELECT nombre FROM users WHERE id ='Bob7' AND password = '0' OR 1=1 #"'

Esto provocaría que la base de datos busque todos los nombres de la tabla users que cumplan las condiciones id =‘Bob7’ Y password = ‘0’ o 1=1. Como 1=1 siempre da verdadero, todos los registros en la tabla cumplirían la condición y la aplicación devolvería al atacante los nombres de todos los usuarios registrados en la tabla.

¿Cuándo es vulnerable una aplicación?

- Cuando la información ingresada por el usuario no es validada, filtrada o saneada.
- Cuando las cadenas ingresadas por el usuario se concatenan de manera directa a la consulta SQL en lugar de usar variables parametrizadas y/o procedimientos almacenados.

Algunas maneras de evitarlo:

- Saneando las entradas.
- Desarrollando con APIs y frameworks que ya realicen controles soble las entradas.
- Utilizando whitelists que verifiquen las entradas antes de llevarlas al interprete.
- Escapando caracteres especiales que den indicios de un intento de Inyección de código.

Broken Authentication

Esta vulnerabilidad se da cuando las funciones de la aplicación dedicadas a la autenticación de los usuarios y a la administración de las sesiones se implementan de manera incorrecta, permitiendo a un atacante asumir la identidad temporal o permanente de otro usuario.

¿Qué son los IDs de sesión?

Es un número identificador que se genera para cada sesión de usuario con el fin de identificar sus peticiones al servidor.

¿Cuándo es vulnerable una aplicación?

- Cuando el sistema de autenticación permite ataques automáticos de fuerza bruta.
- Permite que los usuarios utilicen contraseña debiles como “123456” “contraseña”, etc.
- Cuenta con mecanismos de recuperación de contraseña debiles.
- Almacena los usuarios y contraseñas en texto plano, pobremente encriptados o no hasheados.
- Expone datos de la sesión en la url
- Los ID de sesión no se invalidan apropiadamente (tras cerrar sesión o un timeout)

Algunas maneras de evitarlo son:

- Utilizando autenticación multi-factor
- Contar con una lista de contraseñas débiles no permitidas.
- Exigir que la contraseña cumpla ciertos criterios. (Como los establecidos por NIST 800-63 B's) (National Institute of Standards and Technology)
- Controlar los mensajes de error al crear cuentas y solicitar recuperación de contraseña para evitar indicios de cuentas creadas.
- Limitar la cantidad de intentos fallidos de inicio de sesión antes de bloquear el usuario.
- Utilizar gestores de sesiones que generen ID de sesión aleatorios tras el login. Evitar la visualización de los IDs en la URL e invalidarlos tras el logout o un tiempo de inactividad coherete.

Exposición de datos sensibles

Agentes y Vectores de Ataque

- Tener en consideracion quien puede tener acceso a sus datos sensibles y el respaldo de los mismos.
- Los atacantes tipicamente no quiebran la criptografia de forma directa, sino algo mas como robar claves, realizar ataques “man-in-the-middle”, robar datos del servidor que en algunos casos esta en texto plano. En caso de obtener datos, realizar ataques por fuerza bruta.

Debilidades de Seguridad

- La vulnerabilidad mas comun es simpemente no cifrar datos sensibles (como una password).
- Cuando se emplea alguna tecnica de cifrado, tambien es muy comun realizar cifrados con algoritmos debiles y tecnicas debiles de hashing.

Impacto

- Los fallos de este tipo comprometen todos los datos que deberian estar protegidos.
- Tipicamente estos datos son sensibles, como puede ser una historia clinica, credenciales, datos personales, tarjetas de credito, etc.

¿La Aplicacion es Vulnerable?

Lo primero es determinar las necesidades de protección de los datos. Por ejemplo, contraseñas, números de tarjetas de crédito, registros médicos, información personal y datos sensibles del negocio requieren protección adicional, etc. Para todos estos datos:

- ¿Se transmite datos en texto plano? Esto se refiere a protocolos como HTTP, SMTP, TELNET, FTP. El tráfico en Internet es especialmente peligroso. Verifique también todo el tráfico interno, por ejemplo, entre servidores web o sistemas de backend.
- ¿Se utilizan algoritmos criptográficos obsoletos o débiles?
- ¿se aplica cifrado?

¿Como Prevenirlo?

- Asegurese de cifrar los datos sensibles.
- Garantizar algoritmos de cifrado y protocolos actualizados y sólidos.
- Use la gestión de claves adecuada.
- Cifre todos los datos en tránsito con protocolos seguros-
- Deshabilite el autocompletado automatico en el llenado de formularios que incluyen datos sensibles.
- Deshabilite tambien el cacheado de paginas que contengan datos sensibes.
- Almacene las contraseñas utilizando hashing

XML External Entities

Los procesadores XML viejos o mal configurados evaluan las referencias a fuentes externas contenidas dentro de documentos XML. Los atacantes pueden explotar procesadores XML vulnerables subiendo documentos XML con referencias a contenido malicioso. Los mismos pueden usarse para extraer datos, ejecutar request remotas, escanear el sistema, realizar ataques de denegación de servicio u otros ataques.

Pérdida de Control de Acceso

Agentes y Vectores de Ataque

- La explotación del control de acceso es una habilidad esencial de los atacantes.
- Las herramientas SAST y DAST pueden detectar la ausencia de controles de acceso pero, en el caso de estar presentes, no pueden verificar si son correctos.

- **SAST:** Las herramientas de análisis de código fuente , también conocidas como herramientas de prueba de seguridad de aplicaciones estáticas (SAST), están diseñadas para analizar el código fuente y / o versiones compiladas de código para ayudar a encontrar fallas de seguridad.
- **DAST:** Los escáneres de vulnerabilidades de aplicaciones web son herramientas automatizadas que escanean aplicaciones web, normalmente desde el exterior, para buscar vulnerabilidades de seguridad como secuencias de comandos entre sitios , inyección de SQL , inyección de comandos , recorrido de rutas y configuración de servidor insegura. Esta categoría de herramientas se conoce con frecuencia como herramientas de pruebas de seguridad de aplicaciones dinámicas

Debilidades de Seguridad

- Las debilidades del control de acceso son comunes debido a la falta de detección automática y a la falta de pruebas funcionales efectivas por parte de los desarrolladores de aplicaciones.
- La detección de fallas en el control de acceso no suele ser cubierto por pruebas automatizadas, tanto estáticas como dinámicas.

Impacto

- El impacto incluye atacantes anónimos actuando como usuarios o administradores; usuarios que utilizan funciones privilegiadas o crean, acceden, o eliminan cualquier registro.

¿La Aplicacion es Vulnerable?

Las restricciones de control de acceso implican que los usuarios no pueden actuar fuera de los permisos previstos. Típicamente, las fallas conducen a la divulgación, modificación o destrucción de información no autorizada de los datos, o a realizar una función de negocio fuera de los límites del usuario.

Las vulnerabilidades comunes de control de acceso incluyen:

- Elevación de privilegios. Actuar como un usuario sin iniciar sesión, o actuar como un administrador habiendo iniciado sesión como usuario estándar.
- Manipulación de metadatos, como reproducir un token de control de acceso JWT (JSON Web Token), manipular una cookie o un campo oculto para elevar los privilegios, o abusar de la invalidación de tokens JWT.
- Forzar la navegación a páginas autenticadas como un usuario no autenticado o a páginas privilegiadas como usuario estándar.
- Acceder a una API sin control de acceso mediante el uso de verbos POST, PUT y DELETE.

¿Como Prevenirlo?

El control de acceso sólo es efectivo si es aplicado del lado del servidor o en la API, donde el atacante no puede modificar la verificación de control de acceso o los metadatos.

- Implemente los mecanismos de control de acceso una vez y reutilícelo en toda la aplicación, incluyendo minimizar el control de acceso HTTP
- Los modelos de dominio deben hacer cumplir los requisitos exclusivos de los límites de negocio de las aplicaciones.
- Deshabilite el listado de directorios del servidor web y asegúrese que los metadatos/fuentes de archivos (por ejemplo de GIT) y copia de seguridad no estén presentes en las carpetas públicas.

- Registre errores de control de acceso y alerte a los administradores cuando corresponda.
- Limite la tasa de acceso a las APIs para minimizar el daño de herramientas de ataque automatizadas.
- Los tokens JWT deben ser invalidados luego de la finalización de la sesión por parte del usuario.

Security Misconfiguration

Es una de las vulnerabilidad más comunes, normalmente resultado de utilizar configuraciones no seguras que vienen por defecto. No solo basta con configurar apropiadamente las aplicaciones, librerías y frameworks, sino que también debe actualizarlas ya que los últimos parches suelen solucionar brechas de seguridad ya difundidas.

¿Cuándo es vulnerable mi aplicación?

- Cuando utiliza herramientas, librerías, etc., desactualizadas y/o con brechas de seguridad conocidas.
- Cuando tiene instaladas/habilitadas características innecesarias como puertos abiertos no usados.
- Tiene habilitados usuarios por defecto con sus contraseñas originales.
- Las excepciones no se manejan apropiadamente y los mensajes de error muestran demasiada información.

Algunas maneras de evitarlo:

- Utilizar plataformas mínimas solo con las herramientas y características usadas.
- Revisar y actualizar las configuraciones de las herramientas acorde a sus últimas actualizaciones.
- Ejecutar escaneos automatizados para verificar la efectividad de las configuraciones.

Cross-Site Scripting (XSS)

Agentes y Vectores de Ataque

Existen herramientas automatizadas que permiten detectar y explotar las tres formas de XSS.

Debilidades de Seguridad

XSS es la segunda vulnerabilidad más frecuente en OWASP Top 10, y se encuentra en alrededor de dos tercios de todas las aplicaciones. Las herramientas automatizadas pueden detectar algunos problemas XSS en forma automática, particularmente en tecnologías maduras como PHP.

Impacto

El impacto de XSS es moderado para el caso de XSS Reflejado y XSS en DOM, y severa para XSS Almacenado, que permite ejecutar secuencias de comandos en el navegador de la víctima, para robar credenciales, secuestrar sesiones, o la instalación de software malicioso en el equipo de la víctima.

¿La Aplicacion es Vulnerable?

Existen tres formas usuales de XSS para atacar a los navegadores de los usuarios:

- **XSS Reflejado:** La aplicación o API utiliza datos sin validar, suministrados por un usuario y codificados como parte del HTML o Javascript de salida. Un ataque exitoso permite al atacante ejecutar comandos arbitrarios (HTML y Javascript) en el navegador de la víctima. Típicamente el usuario interactua con un enlace, o alguna otra página controlada por el atacante, como un ataque con publicidad maliciosa, o similar.

¿La Aplicacion es Vulnerable?

- **XSS Almacenado:** La aplicación o API almacena datos proporcionados por el usuario sin validar ni sanear, los que posteriormente son visualizados o utilizados por otro usuario o un administrador. Usualmente es considerado como de riesgo de nivel alto o crítico.
- **XSS Basados en DOM:** Frameworks en JavaScript, aplicaciones de página única o APIs incluyen datos dinámicos, controlables por un atacante.

Los ataques XSS incluyen el robo de la sesión, apropiación de la cuenta, evasión de autentificación, reemplazo de nodos DOM,

¿Como Prevenirlo?

Prevenir XSS requiere mantener los datos no confiables separados del contenido activo del navegador.

- Utilizar frameworks seguros que, por diseño, automáticamente codifican el contenido para prevenir XSS. Ejemplo: React JS.
- Codificar los datos de requerimientos HTTP no confiables en los campos de salida HTML (cuerpo, atributos, JavaScript, CSS, o URL) resuelve los XSS Reflejado y XSS Almacenado.
- Habilitar una Política de Seguridad de Contenido (CSP) es una defensa profunda para la mitigación de vulnerabilidades XSS.

- Content Security Policy (CSP) es una capa adicional de seguridad que ayuda a detectar y mitigar ciertos tipos de ataques, incluidos Cross Site Scripting (XSS) y ataques de inyección de datos. Estos ataques se utilizan para todo, desde el robo de datos hasta la destrucción del sitio y la distribución de malware.

Insecure Deserialization

Conceptos

- Serialización: Transformación de un objeto digital a un formato que pueda persistirse en un disco, enviarse a través de una red, etc.
- Deserialización: Transformación inversa a la serialización.

- En esta vulnerabilidad, el atacante puede modificar el objeto serializado con el fin de que, al momento de deserializarlo, esto pueda resultar en ejecución de código malicioso, ataques de injección y accesos no autorizados.

Algunas formas de evitarlo:

- Implementando chequeos de integridad
- Aislar el código que ejecuta la deserialización en entornos con pocos privilegios.
- Monitorear las deserializaciones y alertar cuando un usuario deserializa constantemente.

Uso de Componentes con Vulnerabilidades Conocidas

Agentes y Vectores de Ataque

Es sencillo obtener exploits para vulnerabilidades ya conocidas pero la explotación de otras requieren un esfuerzo considerable, para su desarrollo y/o personalización.

Debilidades de Seguridad

El desarrollo basado fuertemente en componentes de terceros, puede llevar a que los desarrolladores no entiendan qué componentes se utilizan en la aplicación o API y, mucho menos, mantenerlos actualizados.

Impacto

- Mientras que ciertas vulnerabilidades conocidas conllevan impactos menores, algunas de las mayores brechas registradas han sido realizadas explotando vulnerabilidades conocidas en componentes comunes.
- Dependiendo del activo que se está protegiendo, este riesgo puede ser incluso el principal de la lista.

¿La Aplicacion es Vulnerable?

Es potencialmente vulnerable si:

- No conoce las versiones de todos los componentes que utiliza (tanto del lado del cliente como del servidor). Esto incluye componentes utilizados directamente como sus dependencias anidadas.
- El software es vulnerable, no posee soporte o se encuentra desactualizado. Esto incluye el sistema operativo, servidor web o de aplicaciones, DBMS, APIs y todos los componentes, ambientes de ejecución y bibliotecas.

¿Como Prevenirlo?

Remover dependencias, funcionalidades, componentes, archivos y documentación innecesaria y no utilizada.

- Utilizar una herramienta para mantener un inventario de versiones de componentes (por ej. frameworks o bibliotecas) tanto del cliente como del servidor. Por ejemplo, Dependency Check y retire.js.
- Monitorizar continuamente fuentes como CVE y NVD en búsqueda de vulnerabilidades en los componentes utilizados.

¿Como Prevenirlo?

- Utilizar herramientas de análisis automatizados. Suscribirse a alertas de seguridad de los componentes utilizados.
- Obtener componentes únicamente de orígenes oficiales utilizando canales seguros. Utilizar preferentemente paquetes firmados con el fin de reducir las probabilidades de uso de versiones manipuladas maliciosamente.
- Supervisar bibliotecas y componentes que no poseen mantenimiento o no liberan parches de seguridad para sus versiones obsoletas o sin soporte.

Insufficient Logging & Monitoring

- El monitoreo y el guardado de logs ineficiente permite a los atacantes persistir sus ataques, extenderlos a más sistemas, extraer grandes cantidades de datos y causar mayores daños antes de ser detectados y detenidos. Según OWASP, al año 2016, identificar una brecha de seguridad tomaba un promedio de 191 días. Casi siempre, identificados por terceros en lugar de por procesos y monitoreos internos.

Algunas maneras de evitarlo:

- Registrar todos los logins, validaciones fallidas de inputs, errores de control de acceso.

Ejercicios !!!

- SQL Injection
- XSS (Cross Site Scripting - DOM)

SQL Injection

EJERCICIO 1:

- Traer el nombre de todas las tablas y descubrir cual contiene a los usuarios.

SQL Injection

EJERCICIO 1:

- Traer el nombre de todas las tablas y descubrir cual contiene a los usuarios.
- **AYUDA:** INFORMATION_SCHEMA.TABLES tiene las tablas de la base de datos y el campo que tiene el nombre de la tabla es table_name.

SQL Injection

EJERCICIO 1:

- Traer el nombre de todas las tablas y descubrir cual contiene a los usuarios.
- **AYUDA:** INFORMATION_SCHEMA.TABLES tiene las tablas de la base de datos y el campo que tiene el nombre de la tabla es table_name.
- **RESPUESTA:** ' UNION SELECT null, table_name FROM INFORMATION_SCHEMA.TABLES; #

EJERCICIO 2

- Descubrir el nombre de las columnas de la tabla de usuarios

EJERCICIO 2

- Descubrir el nombre de las columnas de la tabla de usuarios
- **RESPUESTA:** ' UNION SELECT null, column_name FROM INFORMATION_SCHEMA.COLUMNS WHERE table_name = 'users'; #

EJERCICIO 3

- Hacer que la app nos muestre los usuarios con sus contraseñas

EJERCICIO 3

- Hacer que la app nos muestre los usuarios con sus contraseñas
- **RESPUESTA:** ' UNION SELECT user, password FROM users; #

XSS (DOM)

EJERCICIO 1

- Cambiar el color del texto en el “container” a blanco.

XSS (DOM)

EJERCICIO 1

- Cambiar el color del texto en el “container” a blanco.
- **RESPUESTA:** ?default=English<script>document.getElementById(“container”).style.color=“white”</script>

EJERCICIO 2

- Eliminar el container

EJERCICIO 2

- Eliminar el container
- **RESPUESTA 1:** ?default=English<script>var n = document.body.childNodes[0]; n.parentNode.remove(n) </script>

EJERCICIO 2

- Eliminar el container
- **RESPUESTA 1:** ?default=English<script>var n = document.body.childNodes[0]; n.parentNode.remove(n) </script>
- **RESPUESTA 2:** ?default=English<script>var e = document.body;e.parentNode.removeChild(e); </script>

EJEMPLO XSS DOM

http://localhost:3000/vulnerabilities/xss_d/?default=English
<script> var s = document.createElement("span"); s.innerHTML = ' Cerveza Gratis';
document.getElementById("container").appendChild(s);</script>