UTN FACULTAD REGIONAL CÓRDOBA
INGENIERÍA EN SISTEMAS DE INFORMACIÓN

Desarrollo de Software
Notas de Clase

Clase 13

SEGURIDAD EN SITIOS WEB (*)

¡Internet es un lugar peligroso! Con mucha frecuencia escuchamos sobre sitios web que dejan de estar disponibles debido a ataques de denegación de servicio, o presentan información modificada (y con frecuencia adulterada) en sus páginas de inicio. En otros casos a nivel de grandes empresas, millones de contraseñas, direcciones de correo electrónico y detalles de tarjetas de crédito han sido filtrados al dominio público, exponiendo a los usuarios del sitio web tanto al bochorno personal como a riesgo financiero.

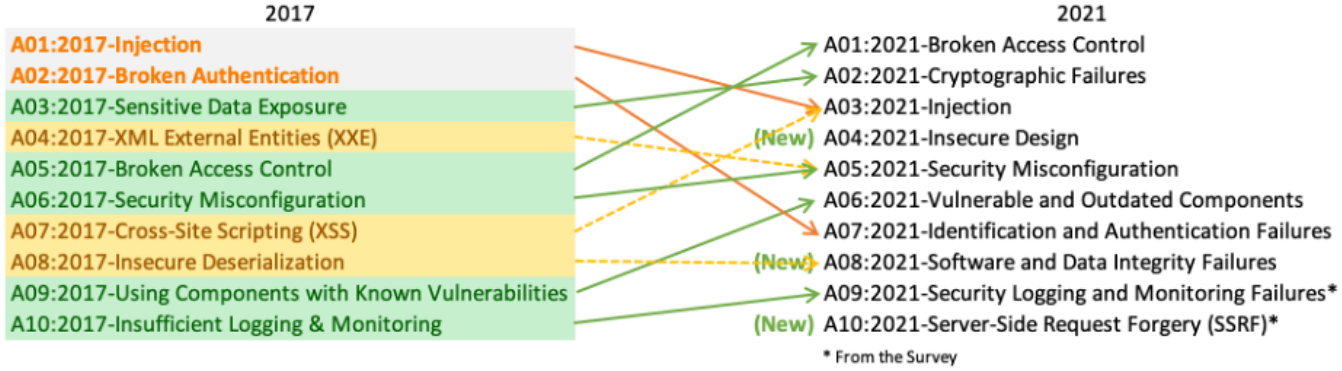
El propósito de la seguridad web es prevenir ataques de esta (o de cualquier otra) clase. Mas formalmente, la seguridad es la acción/práctica de proteger sitios web del acceso, uso, modificación, destrucción o interrupción, no autorizados.

La seguridad de sitios web eficaz requiere de esfuerzos de diseño a lo largo de la totalidad del sitio web: en tu aplicación web, en la configuración del servidor web, en tus políticas para crear y renovar contraseñas, y en el código del lado cliente. Al mismo tiempo que todo esto suena muy inquietante, la buena noticia es que si estás usando un framework web de lado servidor, es casi seguro que habilitará por defecto mecanismos de defensa robustos y bien pensados contra gran cantidad de los ataques más comunes. Otros ataques pueden mitigarse por medio de la configuración de tu servidor web, por ejemplo habilitando HTTPS. Finalmente, hay herramientas de escaneado de vulnerabilidades disponibles públicamente que pueden ayudarte a averiguar si has cometido algún error obvio.

OWASP (*)

El Open Web Application Security Project ® (OWASP) es una fundación sin fines de lucro que trabaja para mejorar la seguridad del software. A través de proyectos de software de código abierto liderados por la comunidad, la Fundación OWASP es la fuente para que los desarrolladores y tecnólogos protejan la web. El OWASP Top 10 es un documento de concienciación estándar para desarrolladores y seguridad de aplicaciones web. Representa un amplio consenso sobre los riesgos de seguridad más críticos para las aplicaciones web.
https://owasp.org/www-project-top-ten/

Vulnerabilidades mas comunes detectadas en sitios web alrededor del mundo



Amenazas contra la seguridad de sitios web (*)

A continuación describiremos algunas de las amenazas más comunes para los sitios web y cómo son mitigadas. A medida que vayas leyendo, fíjate cómo las amenazas tienen éxito cuando la aplicación web, ¡o confía o no es lo suficientemente paranoica acerca de los datos que vienen del explorador web!

1- OWASP - Broken Access Control

- Vulnerabilidad en la que un usuario puede acceder a recursos o realizar acciones que no debería tener permiso para hacerlo, debido a una mala implementación de controles de acceso.
- Controles de acceso en una aplicación web mal implementados o no se verifican adecuadamente lo que permite que un usuario pueda acceder a información o funcionalidades que deberían estar restringidas a otro tipo de usuarios o roles
- 61 % de todas las brechas de seguridad involucran el control de acceso

Mitigaciones

- No confíe en la ofuscación
- Negar el acceso anónimo.
- Utilizar un único mecanismo de control de acceso para toda la aplicación:
- Auditar y testear el control de acceso continuamente
- Minimizar el uso de recursos con CORS
- Deshabilita la lista de directorios del servidor web
- Registrar y alertar ante fallas repetidas
- Limita la tasa de llamadas a la API
- Forzar que todas las solicitudes pasen por los chequeos decontroles de acceso
- Dar el menor permiso para el acceso como sea posible

2 - OWASP - Cryptographic Failures

- Vulnerabilidades que se producen cuando se utiliza criptografía de manera incorrecta o se confía en algoritmos criptográficos débiles o defectuosos.
- Puede ocurrir en diferentes aspectos de una aplicación web, como el almacenamiento de contraseñas, la transmisión de datos confidenciales, la gestión de claves de cifrado y la autenticación de usuarios.
- Para Verificar algoritmos de cifrado de sitios web https://www.ssllabs.com/
- Solo en septiembre 2022 36millones de registros fueron vulnerados

3 - OWASP - Inyección SQL

Las vulnerabilidades de Inyección SQL habilitan que usuarios maliciosos ejecuten código SQL arbitrario en una base de datos, permitiendo que se pueda acceder a los datos, se puedan modificar o borrar, independientemente de los permisos del usuario. Un ataque de inyección con éxito, podría falsificar identidades, crear nuevas identidades con derechos de administración, acceder a todos los datos en el servidor o destruir/modificar los datos para hacerlos inutilizables.

Esta vulnerabilidad está presente si la entrada del usuario que se pasa a la sentencia SQL subyacente puede cambiar el significado de la misma. Por ejemplo, considera el código de abajo, que pretende listar todos los usuarios con un nombre en particular (userName) que ha sido suministrado en un formulario HTML:

```
statement = "SELECT * FROM users WHERE name = '" + userName + "';"
```

Si el usuario introduce su nombre real, la cosa funciona como se pretende. Sin embargo un usuario malicioso podría cambiar completamente el comportamiento de esta sentencia SQL a la nueva sentencia de abajo, simplemente especificando para userName el texto siguiente:

```
a';DROP TABLE users; SELECT * FROM userinfo; --
```

La sentencia modificada crea una sentencia SQL válida que borra la tabla users y selecciona todos los datos de la tabla userinfo (revelando la información de todos los usuarios). Esto funciona por que la primera parte del texto inyectado **a';** completa la sentencia original (' es el símbolo para indicar una cadena literal en SQL) y la parte final de la sentencia **--** es un comentario en SQL, que ignora el resto de la sentencia original.

```
SELECT * FROM users WHERE name = 'a';DROP TABLE users; SELECT * FROM userinfo; --
';
```

La manera de evitar esta clase de ataque es asegurar que cualquier dato de usuario que se pasa a un query SQL no puede cambiar la naturaleza del mismo. Una forma de hacer ésto es eludir ('escape') todos los caracteres en la entrada de usuario que tengan un significado especial en SQL.

Nota: La sentencia SQL trata el carácter **'** como el principio y el final de una cadena de texto. Colocando el carácter barra invertida \ delante, "eludimos" el símbolo ('), y le decimos a SQL que lo trate como un carácter de texto (como parte de la misma cadena).

En la sentencia de abajo eludimos el carácter '. SQL interpretará ahora como "nombre" la cadena de texto completa mostrada en negrita (!un nombre muy raro desde luego, pero no dañino¡)

SELECT * FROM users WHERE name = **'a';DROP TABLE users; SELECT * FROM userinfo WHERE 1 = 1**';

Los frameworks web con frecuencia tienen cuidado de hacer por tí la elusión de caracteres. Django, por ejemplo se asegura que cualquier dato de usuario que se pasa a los conjuntos de queries (modelo de queries) está corregido.

4 OWASP - Insecure Design

- Vulnerabilidad que se origina en la etapa de diseño y arquitectura de una aplicación web. Esto ocurre cuando el diseño de la aplicación no considera adecuadamente los riesgos de seguridad y no se implementan los controles de seguridad necesarios para proteger los datos y funcionalidades críticas.
- Afecta a diferentes áreas de una aplicación, como la autenticación y autorización, la gestión de sesiones, el manejo de datos sensibles, la validación de entrada, y la implementación de controles de acceso.

5- OWASP Security misconfiguration

- Vulnerabilidad que se produce cuando una aplicación web está configurada de manera incorrecta o no se implementan adecuadamente los controles de seguridad necesarios. Esto puede ocurrir en diferentes niveles, incluyendo el servidor web, la base de datos, la plataforma de desarrollo y la aplicación en sí.
- Motivos: uso de contraseñas débiles, la exposición de información de depuración en el servidor web, la ejecución de aplicaciones con permisos excesivos, la falta de parches y actualizaciones de software, y la falta de auditorías y pruebas de seguridad.

Prevención

- Configura el servidor web, la base de datos y la plataforma de desarrollo de acuerdo con las mejores prácticas de seguridad.
- Utiliza contraseñas seguras y cambia las contraseñas predeterminadas.
- Instala actualizaciones y parches de seguridad tan pronto como estén disponibles.
- Limita el acceso a los recursos según sea necesario y evita los permisos excesivos.
- Desactiva o limita la información de depuración del servidor web.
- Utiliza herramientas de automatización para escanear y auditar la configuración de seguridad de la aplicación.
- Realiza pruebas regulares de seguridad para identificar y mitigar cualquier vulnerabilidad de configuración de seguridad incorrecta.

6 - OWASP - Vulnerable and Outdated Components

- Presencia de componentes de software obsoletos o con vulnerabilidades conocidas en una aplicación web
- Bibliotecas, frameworks, módulos y otros componentes de software que se utilizan en la aplicación.
- Ej: Log4j2 shodan.io

Prevencion:

- Mantener un inventario actualizado de todos los componentes de software utilizados en la aplicación.
- Utilizar sólo componentes de software confiables y seguros de fuentes de confianza.
- Realizar pruebas de seguridad regulares en los componentes de software para identificar y mitigar vulnerabilidades conocidas.
- Mantener los componentes de software actualizados con las últimas versiones y parches de seguridad.
- Configurar la aplicación para alertar al equipo de seguridad cuando se descubren nuevas vulnerabilidades en los componentes de software utilizados.

7 - OWASP - Identification and Authentication Failures

- vulnerabilidad que se produce cuando una aplicación web no implementa adecuadamente los mecanismos de identificación y autenticación de los usuarios
- Por ejemplo contraseñas débiles, la transmisión no segura de información de autenticación, la falta de autenticación de los usuarios en áreas críticas de la aplicación, y la falta de gestión adecuada de sesiones

8 - OWASP - Software and Data Integrity Failures

- Vulnerabilidad que se produce cuando los datos o el software de una aplicación web son modificados o dañados de manera malintencionada o accidental. Esto puede dar lugar a problemas de seguridad,

como la manipulación de datos, la ejecución de código malicioso y la denegación de servicio.
- Ejemplos: explotación de vulnerabilidades en el software de la aplicación, la inyección de código malicioso en la aplicación, la modificación de datos críticos, y la corrupción de archivos de configuración y datos.

9 - OWASP - "Security Logging and Monitoring Failures

- Vulnerabilidad que se produce cuando una aplicación web no tiene adecuados mecanismos de registro y monitoreo de seguridad. Esto puede dar lugar a problemas de seguridad, como la falta de visibilidad sobre los eventos de seguridad y la falta de respuesta a los incidentes de seguridad.
- la falta de registro de eventos de seguridad importantes, la falta de alertas de seguridad en tiempo real, la falta de respuesta a incidentes de seguridad, y la falta de capacidad de rastreo de actividad de usuario

10 - OWASP - Cross Site Request Forgery (CSRF) (*)

Los ataques de CSRF permiten que un usuario malicioso ejecute acciones usando las credenciales de otro usuario sin el conocimiento o consentimiento de éste.

Este tipo de ataque se explica mejor con un ejemplo. John es un usuario malicioso que sabe que un sitio en particular permite a los usuarios que han iniciado sesión enviar dinero a una cuenta específica usando una petición HTTP POST que incluye el nombre de la cuenta y una cantidad de dinero. John construye un formulario que incluye los detalles de su banco y una cantidad de dinero como campos ocultos, y lo envía por correo electrónico a otros usuarios del sitio (con el botón de Enviar camuflado como enlace a un sitio "hazte rico rápidamente").

Si el usuario pincha el botón de enviar, se envía al servidor una petición HTTP POST que contiene los detalles de la transacción y todos los cookies de lado-cliente que el explorador asocia con el sitio (añadir cookies asociados con el sitio es un comportamiento normal de los exploradores). El servidor comprobará los cookies, y los usará para determinar si el usuario ha iniciado sesión o no y si tiene permiso para hacer la transacción.

El resultado es que cualquier usuario que pinche en el botón Enviar mientras tiene la sesión iniciada en el sitio comercial hará la transacción. ¡John se hará rico!

Nota: El truco aquí es que John no necesita tener acceso a los cookies del usuario (o acceso a sus credenciales) — El explorador del usuario almacena esta información, y la incluye automáticamente en todas las peticiones al servidor asociado.

Una manera de prevenir este tipo de ataque por parte del servidor es requerir que la petción POST incluya una palabra secreta específica del usuario generada por el sitio (la palabra secreta podría proporcionarla el servidor cuando envía el formulario web que se usa para hacer transferencias). Esta aproximación evita que John pueda crear su propio formulario, porque necesitaría conocer la palabra secreta que el servidor ha proporcionado para el usuario. Incluso si conociera esta palabra y creara un formulario para un usuario en particular, no podría usar el mismo formulario para atacar a todos los usuarios.

Los frameworks web incluyen con frecuencia tales mecanismos de prevención de CSRF.

Otras amenazas y vulnerabilidades

Otros ataques/vulnerabilidades que se solapan con las anteriores:

Cross-Site Scripting (XSS) (*)

XSS es un término que se usa para describir una clase de ataques que permiten al atacante inyectar scripts de lado cliente, a través del sitio web, hasta los exploradores de otros usuarios. Como el código inyectado va del servidor del sitio al explorador, se supone de confianza, y de aquí que pueda hacer cosas como enviar al atacante la cookie de autorización al sitio del usuario. Una vez que el atacante tiene la cookie pueden iniciar sesión en el sitio como si fuera el verdadero usuario y hacer cualquier cosa que pueda hacer éste. Dependiendo de que sitio sea, esto podría incluir acceso a los detalles de su tarjeta de crédito, ver detalles de contactos o cambiar contraseñas, etc.

Nota: Las vulnerabilidades XSS han sido históricamente más comunes que las de cualquier otro tipo.

Hay dos aproximaciones principales para conseguir que el sitio devuelva scripts inyectados al explorador — se conocen como vulnerabilidades XSS reflejadas y persistentes.

La mejor defensa contra las vulnerabilidades XSS es eliminar o deshabilitar cualquier etiqueta que pueda contener instrucciones para ejecutar código. En el caso del HTML ésto incluye etiquetas como **script**, **object**, **embed**, y **link**.

El proceso de modificar los datos del usuario de manera que no puedan utilizarse para ejecutar scripts o que afecten de otra forma la ejecución del código del servidor, se conoce como "desinfección de entrada" (input sanitization). Muchos frameworks web desinfectan automáticamente la entrada del usuario desde formularios HTML, por defecto.

Clickjacking (*)

En este tipo de ataque, el usuario malicioso secuestra las pulsaciones de ratón dirigidas a un sitio visible por encima de los demás y las redirige a una página escondida por debajo. Esta técnica se usaría, por ejemplo, para presentar un sitio bancario legítimo pero capturar las credenciales de inicio de sesión en un **iframe** invisible controlado por el atacante. Alternativamente podría usarse para conseguir que el usuario pinchara sobre un botón en un sitio visible, pero al hacerlo realmente estuviera sin advertirlo pinchando en otro botón completamente diferente. Como defensa, tu sitio puede protegerse de ser embebido en un iframe de otro sitio configurando las cabeceras HTTP apropiadamente.

Denegación de Servicio (*)

(Denial of Service (en-US), DoS). DoS se consigue normalmente inundando el sitio objetivo con peticiones espúreas de manera que se interrumpa el acceso a los usuarios legítimos. Las peticiones pueden simplemente ser numerosas, o consumir individualmente gran cantidad de recursos (ej. lecturas lentas, subidas de grandes archivos, etc.) Las defensas contra DoS normalmente trabajan mediante la indentificación y el bloqueo de tráfico "malo" permitiendo sin embargo que atraviesen los mensajes legítimos. Estas defensas se encuentran típicamente dentro o antes del servidor (no son parte de la aplicación web misma).

Salto de Directorios/Revelación de Archivos

En este tipo de ataque un usuario malicioso intenta acceder a partes del sistema de archivos del servidor web a los que no debería tener acceso. Esta vulnerabilidad ocurre cuando el usuario es capaz de pasar nombres de archivo que incluyen caracteres del sistema de navegación (ej. ../../). La solución es desinfectar la entrada antes de usarla.

Inclusión de Archivos

En este ataque un usuario es capaz de especificar, para mostrar o ejecutar, un archivo "no intencionado para ello" en los datos que le pasa al servidor. Una vez ha sido cargado este archivo podría ejecutarse en el servidor web o en el lado cliente (llevando a un ataque XSS). La solución es desinfectar la entrada antes de usarla.

Inyección de Comandos

Los ataques de inyección de comandos permiten a un usuario malicioso ejecutar comandos del sistema arbitrarios en el sistema operativo del host. La solución es desinfectar la entrada de usuario antes de que pueda ser usada en llamadas al sistema.

Este listado no es exhaustivo, existen muchas otras amenazas. Para obtener información actualizada podríamos consultar el trabajo de una fundacion de codigo abierto dedicada a combatir las causas que hacen el software inseguro: https://owasp.org/ Lo importante es tomar medidas concretas para poder mitigarlas.

Medidas concretas para la seguridad (*)

- Casi todos los exploits de las secciones anteriores tienen éxito cuando la aplicación web confía en los datos que vienen del explorador. Sea lo que sea que hagas para mejorar la seguridad de tu sitio web, deberías desinfectar todos los datos originados por el usuario antes de ser mostrados en el explorador, usados en queries SQL o pasados en una llamada al sistema operativo o fichero de sistema.
- Usar una gestión de contraseñas más efectiva. Fomentar las contraseñas fuertes y que se cambien con regularidad. Considerar para tu sitio web la autenticación de dos factores, de manera que, además de la contraseña, el usuario tenga que introducir algún otro código de autenticación (normalmente alguno que se distribuye mediante algún hardware que sólo tiene el usuario, como un código en un mensaje de texto enviado a su teléfono móvil).
- Configurar tu servidor web para usar HTTPS y HTTP Strict Transport Security (en-US) (HSTS). HTTPS encripta los datos enviados entre el cliente y el servidor. Esto asegura que las credenciales de incio de sesión, cookies, datos POST e información de cabecera permanecen menos disponibles a los atacantes.
- Seguir la pista a las amenazas más populares (aquí puedes acceder a la lista actual OWASP) y atacar las vulnerabilidades más comunes primero.
- Usar herramientas de escanéo de vulnerabilidades para realizar pruebas automáticas de seguridad en tu sitio (más adelante, si tu sitio web llega a ser super exitoso puedes también encontrar bugs por medio de ofrecer recompensas por encontrar bugs como hace Google).
- Almacena y muestra sólo los datos que necesiten serlo. Por ejemplo, si tus usuarios deben almacenar información sensible como los detalles de las tarjetas de crédito, sólo muestra lo suficiente del número de tarjeta de manera que pueda ser identificada por el usuario, y no suficiente para que pueda ser copiado por el atacante y usado en otro sitio. El patrón más común hoy en día es mostrar sólo los 4 últimos dígitos del número de la tarjeta de crédito.

Los frameworks web pueden ayudar a mitigar muchas de las vulnerabilidades más comunes.