Erica Lloves Calviño
Francisco José Ribadas Pena
elloves@uvigo.es
ribadas@uvigo.es
Departamento de Informática
Universidade de Vigo

Seguridad en servidores Web.
Módulo mod_security

Ferramentas de seguridade en GNU/Linux
Curso de Extensión Universitaria

26 de xuño de 2009

Indice

- Seguridad en servidores WEB
- Ataques a servidores WEB
- Herramientas de detección/protección
- Módulo Apache *mod_security*
  - Funcionalidades
  - Reglas y configuración
  - Instalación en Debian/Ubuntu

Seguridad en servidores WEB

- Los servidores WEB son una de las mayores fuentes de vulnerabilidades uno de los principales objetivos de los ataques.
  - Es uno de los servicios más usados
    - Disponible en la mayor parte de organizaciones
  - Firewalls no suelen bloquear puerto http (80)
- Fuentes de vulnerabilidades:
  - el propio servidor WEB (bugs, defectos configuración)
  - las páginas HTML y aplicaciones WEB alojadas
- Delegar en otros mecanismos de seguridad/prevención no siempre basta
  - firewalls + detección de intrusiones
  - usar SSL/TLS no soluciona todos los problemas

Defectos/bugs del servidor y la configuración

- Agujeros de seguridad o fallos en la protección del servidor (errores de implementación y/o configuración)
  - Uso del servidor WEB como punto de entrada para nuevos ataques contra la red interna de la organización
  - Permiten la ejecución de código atacante con los permisos del usuario que ejecuta el servidor WEB
- Mecanismos de prevención:
  - Ejecutar servidor con los mínimos privilegios
    - Evitar ejecución SUID (aplicaciones ejecutadas por *root*)
  - Actualización constante (disponer de últimos parches)
  - Log y monitorización de la actividad del servidor
  - Mostrar mínima información sobre el servidor y el sistema
  - Ejecución *enjaulada* (chroot)
    - Definir un entorno *chroo*t donde se ejecutará el servidor WEB aislado del resto del sistema de ficheros de al máquina
    - Minimiza el riesgo de que ataques contra el servidor WEB pudieran afectar a otros elementos del equipo/red

Acceso a información sensible

- Finalidad última de los ataques mal intencionados
  - Acceso a información sensible
    - Robo de passwords, contraseñas, nº tarjeta,...
  - Acceso a información personal
    - cookies, hábitos de navegación, ...
- Técnicas:
  - Escucha del tráfico: *sniffing*
  - Suplantación de páginas web: *phishing*
- Medidas de prevención
  - Evitar canales inseguros
  - Mecanismos criptográficos y cifrado de info. sensible
  - Cifrado de la conexión (protocolo SSL/TLS [https])
    - autentica a los extremos (servidor y/o cliente)
    - asegura confidencialidad e integridad del tráfico
  - Concienciación y “formación” del usuario

Ejecución de código de forma remota

- Aprovechar defectos en diseño de páginas HTML y aplicaciones WEB (generalmente fallos de validación)
  - *cross-site scripting* XSS: vulnerabilidades en la validación de datos que permiten introducir código HTML incrustado
  - *inyección SQL*: inserción de instruciones SQL que serán reenvidas a la BD sin ser validadas
  - *buffer overflow*: inserción de código máquina arbitrario aprovechando *buffers* de entrada no limitados
- Tipo de ataque/vulnerabilidad más frecuente
  - Abundancia y variedad de aplicaciones WEB
  - Deficiencias en el desarrollo (poca atención a seguridad)

    Mientras no se solucionan los problemas en las aplicaciones WEB inseguras (tarea de los desarrolladores), establecer mecanismos para mitigar sus peligros es tarea de los administradores de red.
- Medidas de prevención:
  - Filtrado y validación de entradas en propia aplicación WEB
  - Filtros y detectores de intrusiones integrados en el servidor WEB (*mod_security*)

Cross Site Scripting XSS

- Ejecución de código script arbitrario "incrustado" por un tercero en campos de datos que son enviados tal cual desde el servidor WEB al navegador
  - Hay datos que serán interpretados como código que modifica el comportamiento o aspecto de la página
- **Peligro:** cuando se ejecuta un script se tiene acceso a la página actual y a toda la información contenida en la misma
  - obtener cookie del usuario (robando su id de sesión)
  - redirigir a una página alternativa
  - presentar contenido falso (para que introduzca info. privada)
- Etiquetas <script> </script>
  - Introducir en algún campo de la Web o en un parámetro de una petición dinámica:

```
<script>alert("Esto es vulnerable a XSS")</script>
```

Cross Site Scripting XSS

- Inyección de scritps en otras etiquetas:

  `<img src="javascript:alert("Vulnerable a XSS")>`

  - La mayor parte de etiquetas que admitan los atributos STYLE, SRC, HREF o TYPE son susceptibles de ser usadas:`<html>`,`<body>`,`<embed>`,`<frame>`,`<frameset>`,`<applet>`,`<iframe>`,`<layer>`,`<meta>`,`<object>`,`<style>`
- **Protección:** Filtrar los datos que el usuario introduce.
  - No tan fácil: ofuscación/codificación de las cadenas introducidas para que pasen desapercibidas
  - No se pueden detectar scripts introducidos dentro de etiquetas HTML

Inyección SQL

- Inserción de sentencias/comandos SQL en datos de entrada
  - SQL posibilita la comunicación de datos entre WEB y almacenamiento → generación dinámica de sentencias SQL usando los parámetros que proporciona el usuario
  - Posibilidad de cambiar la naturaleza de la petición que va a ejecutar el servidor de BD
    - Introducción de sentencias o porciones de sentencias SQL en la URL, en campos de la página o en parámetros usados para generar sentencias SQL dinámicamente
- Este ataque es posible cuando hay entradas de usuario que se envíen directamente de aplicación Web a la BD
- **Objetivo:** acceso no autorizado o no restringido a la base de datos (potencialidad de daño enorme)
  - Obtener información sensible contenida en una base de datos sin ser un usuario autorizado del sistema
  - Modificar los datos almacenados
  - Destruir la base de datos

Inyección SQL

- Ejemplo: login contra BD
  - Parámetros entrada: nombre + clave
  - Los parámetros de entrada no se comprueban
- Código de login en el servidor (php)

```
$SQLquery =  "SELECT * FROM users";
$SQLquery .= " WHERE user_login = '".$_POST[“nombre”]."'";
$SQLquery .= " AND user_pass ='".$_POST[“clave”].”'”;

$Dbresult=db_query($SQLQuery);
If ($Dbresult){
   //Existe el usuario y la clav => acceder
}
else {
   //Acceso no permitido
}
```

Inyección SQL

- Si como nombre de usuario se escribe **' OR 1=1 #** (el password puede quedar en blanco) la consulta real siempre devolvería resultados:

  "SELECT * FROM users WHERE user_login = '' OR 1=1 # AND user_pass ='"

  - Asumimos que el servidor BD usa ' como símbolo de cadenas y # como marca de comentario
- Ejecutación de comandos SQL arbitrarios
  - Concatener nuevas sentencias SQL separadas por **;**
  - Permite modificar los contenidos de la BD
- En el caso anterior, usando como nombre de usuario

```
'; INSERT INTO users (user_login, user_password)
   VALUES ('atacante','') #
```

- La sentencia SQL sería:

```
"SELECT * FROM users WHERE user_login = '';
INSERT INTO users (user_login, user_password)
VALUES ('atacante','') # AND user_pass ='"
```

Herramientas disponibles

- Escáneres de seguridad WEB
  - Analizan el servidor y las páginas y servicios alojados
  - Realizan comprobaciones de seguridad (ataques típicos)
    - defectos en la configuración del servidor
    - defectos en páginas/servicios WEB alojados
  - Ejemplos
    - *nikto*: http://www.cirt.net/code/nikto.shtml
    - *WebScarab*: http://www.owasp.org/index.php/OWASP_WebScarab_Project
- Filtros (*firewalls* de aplicación)
  - Integrados en el propio servidor WEB
  - Validan todas las peticiones recibidas, realizando transformaciones y rechazando las que sean sospechosas
  - Ejemplos
  - *mod_security*: www.modsecurity.org

mod_security

- Módulo *open source* para el servidor web Apache
- Soporta detección y prevención de intrusiones en aplicaciones WEB
  - Protección contra ataques conocidos y contra 'posibles' ataques nuevos.
- Añade nivel extra de seguridad 'antes' de servidor WEB
  - Complementario a firewall de paquetes y detectores de intrusiones
- Funciona como “firewall” de aplicación, filtrando tráfico al nivel de aplicaciones *http* y *https*
  - Funcionamiento controlado por reglas y patrones sobre el contenido de los mensajes *http/https* (*análogo a SNORT*)
  - Integrado en servidor => accede al tráfico cifrado SSL después de descifrado y justo antes de su 'proceso' por el servidor (*SNORT no puede analizar tráfico cifrado*)

Características:

- Funciona de modo transparente para las aplicaciones WEB alojadas
  - permite asegurar aplicaciones WEB complejas y/o que no puedan ser modificadas (código 'heredado', aplicaciones privativas,...)
  - no requiere intervención de los desarrolladores originales de las aplicaciones WEB
- Ofrece protección temporal ante vulnerabilidades recién descubiertas que aún no tengan parche de seguridad
  - actualiz. de reglas en la web: www.modsecurity.org
- Comprende todos los detalle de losprotocolos *http* y *https* y permite un control y filtrado 'fino' de los accesos permitidos

Funcionalidades

- ***Filtrado de peticiones:*** las peticiones entrantes son analizadas antes de pasarlas al servidor Apache a a sus módulos
  - uso de expresiones regulares para expresar las reglas
- ***Filtrado de las respuestas del servidor:*** puede analizar/modificar los mensajes de respuesta del servidor (normales o erroneos)
- ***Técnicas anti-evasión:*** los parámetros recibidos y paths de los ficheros (html, php, ...) son normalizados para evitar mecanismos de evasión
  - elimina barras dobles (//) y referencias al propio directorio (./)
  - decodifica las URLs (caracteres especiales)
  - detecta y elimina bytes nulos (%00) [usados en desbordamiento de buffer]

Funcionalidades (cont.)

- ***Soporte HTTP/HTTPS completo:*** puede especificar reglas hasta el nivel de elementos concretos de páginas HTML (enlaces, campos de formularios)
  - procesa el tráfico HTTPS despues de descrifrado
- ***Análisis de datos POST:*** análisis de los datos enviados 'dentro' de peticiones POST
- ***Auditoria y log:*** capacidades completas de *logging* de las peticiones procesadas para el análisis posterior
  - IPs, *timestamps*, método HTTP, URI, datos
- ***chroot integrado:*** soporta funcionalidades para implantar jaulas chroot (aisla servidor)
- ***manejo info. de identificación del servidor:*** se puede configurar y limitar la info. que el servidor ofrece de si mismo (versión, mensajes de error, ...)

mod_security

```
<IfModule mod_security.c>
    SecFilterEngine On                      # Turn the filtering engine On or Off
    SecFilterCheckURLEncoding On            # Make sure that URL encoding is valid
    SecFilterCheckUnicodeEncoding Off  # Unicode encoding check
    SecFilterForceByteRange 0 255           # Only allow bytes from this range


    SecAuditEngine RelevantOnly             # Only log suspicious requests
    SecAuditLog logs/audit_log              # The name of the audit log file
    SecFilterDebugLog logs/modsec_debug_log
    SecFilterDebugLevel 0                   # Debug level set to a minimum


    SecFilterScanPOST On   # Should mod_security inspect POST payloads


    SecFilterDefaultAction "deny,log,status:500" # By default log and deny suspicious
                                        #  requests with HTTP status 500
    SecRule REQUEST_URI     "login_failed\.php" chain
    SecRule ARG_username    "^admin$"           log,exec:/home/apache/bin/notagain.pl
    ...
</IfModule>
```

Reglas mod_security

- Definidas mediante expresiones regulares

  `SecRule [variable/elementos petición] [patrón] [acciones]`

  ```
  SecRule REQUEST_URI|QUERY_STRING ataque
  SecRule ARGS "drop[[:space:]]table"  deny,log
  ```

- Elementos analizados (separados por |)
  - elementos de cabeceras HTTP
  - variables de entorno y variables del servidor
  - cookies
  - variables definidas en las páginas HTML
  - carga de peticiones POST
  - sobre páginas HTML o sobre la salida generada por los lenguajes de script (php, etc)
- Pueden definirse reglas específicas para distintos directorios o servidores virtuales dentro de la estructura de la web

Reglas mod_security *(cont.)*

- Acciones posibles:
  - **deny** rechazo silencioso (sin redirección a mensaje de error)
  - **status:nnn** rechazar petición con un *status code* (403,404,500,...)
  - **allow** detiene procesam. reglas y permite la petición
  - **redirect:*url*** redirige la petición a la *url* indicada
  - **log** ó **nolog** *loggin* (o no) de la petición
  - **exec:cmd** ejecución de un comando externo
  - **pass** ignora la regla actual y pasa a la siguiente
  - **pause:*n*** retiene la petición *n* milisegundos

Reglas mod_security *(cont.)*

- Modelos de funcionamiento: esquemas generales a la hora de definir las reglas

```
SecDefaultAction action1,action2,action3
                 (lista de acciones por defecto)
```

Modelo positivo: permitimos lo que sabemos que es seguro (política por defecto DROP)

- mejor rendimiento
- menos falsos positivos
- más difícil de implantar (dar permiso a todo lo que tenemos en el servidor)

- Modelo negativo: denegamos lo que es peligroso (política por defecto ACCEPT)
  - más fácil de implantar
  - más falsos positivos
  - mayor tiempo de procesamiento

Otras funcionalidades

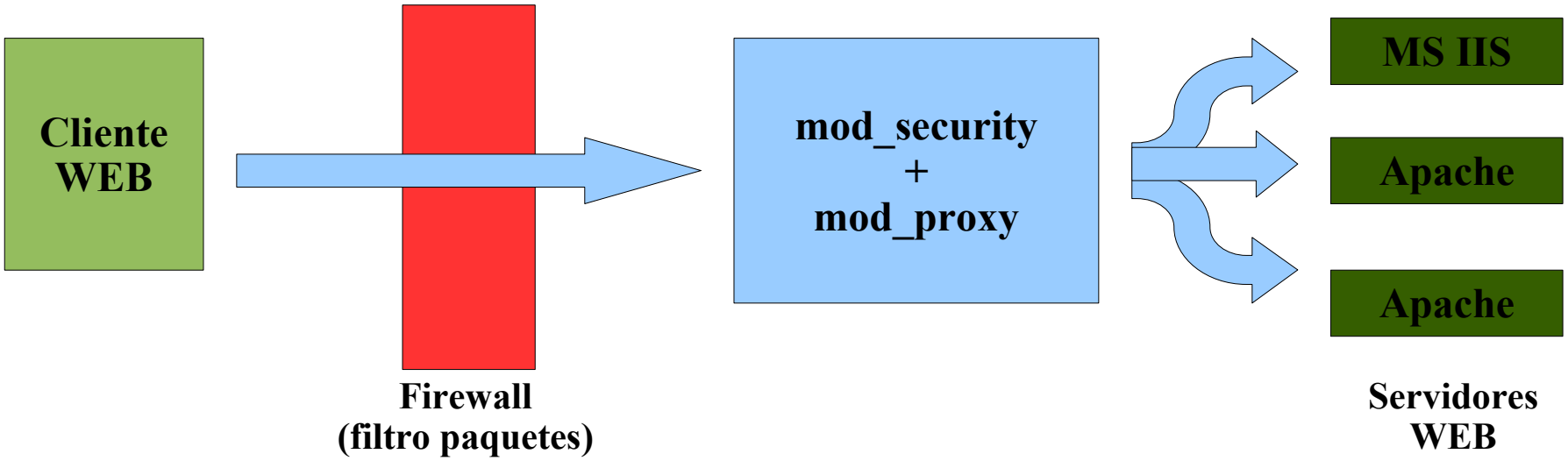
- **chroot interno**
  - *mod_security* permite configurar la ejecución de Apache en un entorno *chroot* interno
    - encierra al servidor en un sistema de directorios "/" falso
    - evita el acceso a directorios y ficheros sensibles (/etc, /dev, /usr/bin)
  - El entorno *chroot* es implantado una vez el servidor está en ejecución

    ```
    SecChrootPath /chroot/var/www
    ```
- **Cambiar la identificación del servidor WEB**
  - *mod_security* gestiona el cambio de la "firma" del servidor (nombre, versión, módulos instalados) que lo identifica en los mensajes HTTP enviados
  - Dificulta a los atacantes la identificación del servidor

    ```
    SecServerSignature "mi servidor"
    ```

Configuraciones avanzadas de *mod_security*

- La instalación básica de *mod_security* protege al servidor web donde sea instalado como módulo
- Combinado con los módulos *mod_proxy* y *mod_proxy_http* se puede implantar una configuración de *Proxy Inverso* (reparto de peticiones)
  - Se implanta un “firewall HTTP” que protege a varios servidores WEB
  - Permite proteger servidores no Apache



Instalación en Debian/Ubuntu (no paquetes oficiales)

- Opciones:
  - Compilación desde código fuente
  - Paquetes no oficiales (http://etc.inittab.org/~agi/debian/)
- Instalación:
  - Editar fichero /etc/apt/sources.list y añadir

```
deb http://etc.inittab.org/~agi/debian/libapache-mod-security2 ./
```

  - Actualizar e instalar

```
apt-get update
apt-get install libapache2-mod-security2
```

  - Instalar módulo Apache
    - en debian: `$ a2enmod mod_security2`
    - manualmente: `/etc/apache2/apache.conf`
  - Editar fichero de configuración y añadir reglas

`/usr/share/doc/mod-security2-common/examples/modsecurity.conf-minimal`
`/usr/share/doc/mod-security2-common/examples/rules/`

  - Reglas actualizadas disponibles en www.modsecurity.org