OWASP TOP 10

LAS 10 VULNERABILIDADES DE SEGURIDAD MAS CRITICAS EN APLICACIONES WEB

VERSION 2007 EN ESPAÑOL

TABLA DE CONTENIDOS

SOBRE ESTA VERSIÓN

Esta versión de Top 10 2007 en español ha sido desarrollada como parte de las actividades del capitulo OWASP Spanish para la comunidad de desarrolladores y seguridad informática de habla hispana.

Participaron de esta traducción:

- Fabio Cerullo
- Jaime Blasco
- Miguel Tubia
- David Echarri
- Emilio Casbas
- Miguel Macias
- Guillermo Correa
- Luis Martinez Bacha
- Camilo Vega
- Jesús Gómez
- Rodrigo Marcos
- Juan Carlos Calderón
- Javier Fernández-Sanguino

Para saber más sobre los eventos y actividades desarrolladas por el capítulo OWASP-Spanish, suscríbase a la lista de distribución OWASP-Spanish.

INTRODUCCIÓN

¡Bienvenido al OWASP Top 10 2007! Esta versión totalmente revisada lista las vulnerabilidades más serias de aplicaciones Web, discute como protegerse de ellas, y provee enlaces para mayor información.

OBJETIVO

El objetivo principal del OWASP Top 10 es educar a desarrolladores, diseñadores, arquitectos y organizaciones sobre las consecuencias de las vulnerabilidades más comunes encontradas en aplicaciones Web. El Top 10 provee métodos básicos para protegerse de dichas vulnerabilidades – un gran comienzo para un programa de seguridad en programación segura.

Seguridad no es un evento único. Es insuficiente programar de manera segura sólo una vez. En 2008, este Top 10 habrá cambiado, y sin cambiar una línea de código en su aplicación, usted podría ser vulnerable. Por favor revise los consejos en la sección ¿Donde sigo ahora? para mayor información.

Una iniciativa de programación segura debe abordar todas las etapas del ciclo de vida de un programa. Aplicaciones Web seguras sólo son posibles cuando un SDLC seguro es utilizado. Los programas seguros son seguros por diseño, durante el desarrollo, y por defecto. Existen al menos 300 problemas que afectan la seguridad general de una aplicación Web. Estos son detallados en la Guía OWASP, la cual es de lectura esencial para cualquiera desarrollando aplicaciones Web hoy en día.

Este documento es sobre todo, un recurso de estudio, y no un estándar. Por favor no adopte este documento como una política o estándar sin antes hablar con nosotros. Si usted necesita una política o estándar de codificación segura, OWASP dispone de proyectos en progreso sobre políticas o estándares de codificación segura. Por favor considere unirse o asistir financieramente con estos esfuerzos.

AGRADECIMIENTOS

Quisiéramos agradecer a MITRE por distribuir públicamente y de manera gratuita los datos de “Vulnerability Type Distribution in CVE”. El proyecto OWASP Top 10 es dirigido y patrocinado por Aspect Security.

Líder de Proyecto: Andrew van der Stock (Director Ejecutivo, Fundación OWASP)

Co-autores: Jeff Williams (Presidente, Fundación OWASP), Dave Wichers (Presidente de la Conferencia, Fundación OWASP)

Quisiéramos agradecer a nuestros revisores:

- Raoul Endres por su ayuda en poner nuevamente en movimiento el Top 10 y sus valiosos comentarios.
- Steve Christey (MITRE) por una extensiva revisión y su contribución de información sobre MITRE CWE.
- Jeremiah Grossman (White Hat Security) por una extensiva revisión y su contribución de información acerca del éxito (o fracaso) de medios automáticos de detección.
- Sylvan von Stuppe por su revisión ejemplar del presente documento.
- Colin Wong, Nigel Evans, Andre Gironda, Neil Smithline por sus comentarios vía e-mail.

RESUMEN

A1 – Secuencia de Comandos en Sitios Cruzados (XSS)	Las fallas de XSS ocurren cuando una aplicación toma información originada por un usuario y la envía a un navegador Web sin primero validar o codificar el contenido. XSS permite a los atacantes ejecutar secuencias de comandos en el navegador Web de la víctima que pueden secuestrar sesiones de usuario, modificar sitios Web, insertar contenido hostil, etc.
A2 – Fallas de Inyección	Las fallas de inyección, en particular la inyección SQL, son comunes en aplicaciones Web. La inyección ocurre cuando los datos proporcionados por el usuario son enviados e interpretados como parte de una orden o consulta. Los atacantes interrumpen el intérprete para que ejecute comandos no intencionados proporcionando datos especialmente modificados.
A3 – Ejecución de ficheros malintencionados	El código vulnerable a la inclusión remota de ficheros (RFI) permite a los atacantes incluir código y datos maliciosos, resultando en ataques devastadores, tales como la obtención de control total del servidor. Los ataques de ejecución de ficheros malintencionados afectan a PHP, XML y cualquier entorno de trabajo que acepte ficheros de los usuarios.
A4 – Referencia Insegura y Directa a Objetos	Una referencia directa a objetos ("direct object reference") ocurre cuando un programador expone una referencia hacia un objeto interno de la aplicación, tales como un fichero, directorio, registro de base de datos, o una clave tal como una URL o un parámetro de formulario Web. Un atacante podría manipular este tipo de referencias en la aplicación para acceder a otros objetos sin autorización.
A5 – Falsificación de Petición en Sitios Cruzados (CSRF)	Un ataque CSRF fuerza al navegador validado de una víctima a enviar una petición a una aplicación Web vulnerable, la cual entonces realiza la acción elegida por el atacante a través de la víctima. CSRF puede ser tan poderosa como la aplicación siendo atacada.
A6 – Revelación de Información y Gestión Incorrecta de Errores	Las aplicaciones pueden revelar, involuntariamente, información sobre su configuración, su funcionamiento interno, o pueden violar la privacidad a través de una variedad de problemas. Los atacantes pueden usar esta vulnerabilidad para obtener datos delicados o realizar ataques más serios.
A7 – Perdida de Autenticación y Gestión de Sesiones	Las credenciales de cuentas y los testigos de sesión (session token) frecuentemente no son protegidos adecuadamente. Los atacantes obtienen contraseñas, claves, o testigos de sesión para obtener identidades de otros usuarios.
A8 – Almacenamiento Criptográfico Inseguro	Las aplicaciones Web raramente utilizan funciones criptográficas adecuadamente para proteger datos y credenciales. Los atacantes usan datos débilmente protegidos para llevar a cabo robos de identidad y otros crímenes, tales como fraude de tarjetas de crédito.
A9 – Comunicaciones Inseguras	Las aplicaciones frecuentemente fallan al cifrar tráfico de red cuando es necesario proteger comunicaciones delicadas.
A10 – Falla de restricción de acceso a URL	Frecuentemente, una aplicación solo protege funcionalidades delicadas previniendo la visualización de enlaces o URLs a usuarios no autorizados. Los atacantes utilizan esta debilidad para acceder y llevar a cabo operaciones no autorizadas accediendo a esas URLs directamente.

Tabla 1: Las Top 10 vulnerabilidades de aplicaciones Web en 2007

METODOLOGÍA

Nuestra metodología para el Top 10 2007 fue simple: tomar el MITRE Vulnerability Trends for 2006, y filtrar los Top 10 problemas de *seguridad en aplicaciones Web*. El ranking es el siguiente:

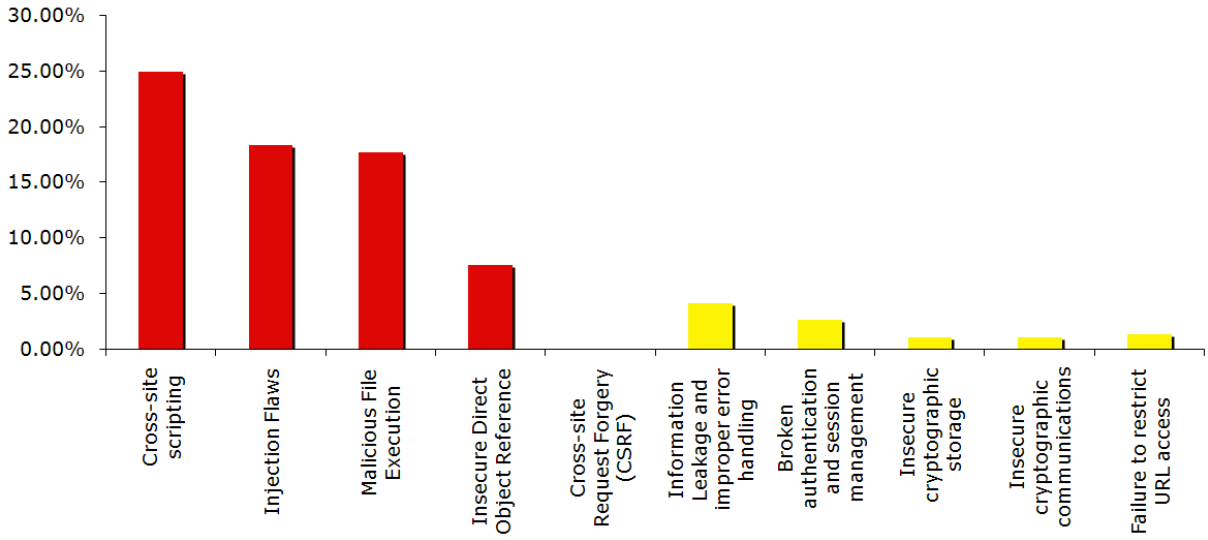

Figura 2: Datos de MITRE sobre las Top 10 vulnerabilidades de aplicaciones Web en 2006

A pesar de que intentamos preservar una relación uno-a-uno entre los datos sobre vulnerabilidades proporcionados por MITRE y nuestros encabezados de sección, hemos cambiado deliberadamente algunas de las siguientes categorías para adecuarse mejor a las causas fundamentales. Si se encuentra interesado en el informe completo de MITRE 2006, hemos incluido una hoja de calculo Excel en el sitio Web OWASP.

Todas las recomendaciones de protección detalladas aquí proveen soluciones a los tres entornos más comunes en aplicaciones Web: Java EE, ASP.NET, y PHP. Otros entornos comúnmente utilizados, tales como Ruby o Perl se pueden fácilmente adaptar las recomendaciones a sus necesidades específicas.

POR QUE HEMOS ELIMINADO ALGUNAS CUESTIONES IMPORTANTES

Las entradas sin validar son un desafío mayor para cualquier equipo de desarrollo, y son la raíz de muchos problemas de seguridad en aplicaciones. De hecho, muchos de los otros ítems en esta lista recomiendan validar entradas como parte de la solución. Nosotros recomendamos crear un mecanismo de validación de entradas centralizado como parte de vuestras aplicaciones Web. Para mayor información, lea los siguientes artículos sobre validación de entradas en OWASP:

- http://www.owasp.org/index.php/Data_Validation
- http://www.owasp.org/index.php/Testing_for_Data_Validation

Desbordamientos de pila, desbordamientos de enteros y cuestiones de formato en cadenas de caracteres son vulnerabilidades extremadamente serias para programas escritos en lenguajes tales como C o C++. La resolución de dichos problemas es cubierta por las comunidades de seguridad no especializadas en aplicaciones Web, tales como SANS, CERT, y vendedores de herramientas de lenguajes de programación. Si su código está escrito en un lenguaje que puede sufrir desbordamientos de pila, lo alentamos a leer el siguiente contenido en OWASP:

- http://www.owasp.org/index.php/Buffer_overflow
- http://www.owasp.org/index.php/Testing_for_Buffer_Overflow

Negación de servicio (NdS) es un ataque serio que puede afectar cualquier sitio escrito en cualquier lenguaje. El ranking sobre NdS de MITRE es insuficiente para alcanzar el Top 10 este año. Si desea conocer más sobre negación de servicio, le aconsejamos visite el sitio OWASP y la Guía de Testeo:

- http://www.owasp.org/index.php/Category:Denial_of_Service_Attack
- http://www.owasp.org/index.php/Testing_for_Denial_of_Service

La gestión insegura de configuraciones afecta a todos los sistemas hasta cierto punto, particularmente PHP. Sin embargo, el ranking de MITRE no nos permite incluir este tema este año. Cuando distribuya su aplicación, deberá consultar la última Guía OWASP y la Guía de Testeo OWASP para obtener información detallada acerca de gestión insegura de configuraciones y testeo:

- http://www.owasp.org/index.php/Configuration
- http://www.owasp.org/index.php/Testing_for_infrastructure_configuration_management

¿PORQUE AGREGAMOS ALGUNAS CUESTIONES IMPORTANTES?

Falsificación de Petición en Sitios Cruzados (CSRF) es la mayor nueva incorporación en esta edición del OWASP Top 10. Si bien los datos en bruto de MITRE la colocan en posición #36, creemos que es lo suficientemente importante como para protegerse de ella hoy en día, particularmente para aplicaciones de alto coste y aplicaciones que manejan información sensible. CSRF es más común de lo que su actual ranking indica, y puede ser extremadamente peligrosa.

Criptografía Los usos inseguros de criptografía no son las posiciones #8 y #9 en cuanto a problemas de seguridad en aplicaciones Web tal como lo indican los datos de MITRE, pero representan una causa fundamental de muchos problemas de privacidad y cumplimiento de normativas (particularmente para la conformidad con PCI DSS 1.1).

VULNERABILIDADES, NO ATAQUES

La edición previa del Top 10 contenía una mezcla de ataques, vulnerabilidades y contramedidas. Esta vez en cambio, nos centramos solamente en vulnerabilidades, aunque la terminología comúnmente utilizada en esta edición a veces combina las vulnerabilidades y los ataques. Si las organizaciones utilizan este documento para asegurar sus aplicaciones, y reducir los riesgos en sus negocios, esto dará lugar a una reducción directa en la probabilidad de:

- **Ataques de "Phishing"** que pueden explotar cualquiera de estas vulnerabilidades, particularmente XSS, y comprobaciones débiles o no existentes de autenticación y autorización (A1, A4, A7, A10)
- **Violaciones de Privacidad** debido a una validación insuficiente, reglas de negocio y comprobaciones de autorización débiles (A2, A4, A6, A7, A10)
- **Robo de identidad** debido a insuficientes o no existentes controles criptográficos (A8 y A9), inclusión de archivos remoto (A3) y autenticación, reglas de negocio, y comprobaciones de autenticación (A4, A7, A10)
- **Toma de control de sistemas, alteración de datos o destrucción de datos** a través de inyecciones (A2) e inclusión de archivos remotos (A3)
- **Perdidas financieras** debido a transacciones no autorizadas y ataques CSRF (A4, A5, A7, A10)

- **Pérdida de reputación** a través de la explotación de cualquiera de estas vulnerabilidades (A1 … A10)

Cuando una organización supera el preocuparse por controles reactivos, y comienza a reducir proactivamente riesgos aplicables a sus negocios, entonces mejorará el cumplimiento de los regímenes normativos, reducirá costos operativos y es de esperar que como resultado tenga sistemas más robustos y seguros.

SESGOS

La metodología descripta aquí necesariamente sesga el Top 10 hacia descubrimientos realizados por la comunidad de investigadores de seguridad. Este patrón de descubrimiento es similar a los métodos de ataque real, en particular en lo que se refiere a atacantes principiantes ("script kiddy"). La protección de su software contra el Top 10 ofrecerá un mínimo de protección contra las formas más comunes de ataque, pero lo que es más importante es que ayudará a fijar un curso para mejorar la seguridad de su software.

SEMEJANZAS

Se han producido cambios en los encabezados, aun donde los contenidos eran similares con los anteriores. No utilizamos más el nombre de esquemas XML, ya que no se ha mantenido al día con las vulnerabilidades actuales, los ataques y las contramedidas. La siguiente tabla muestra como esta edición se asemeja con la edición Top 10 2004, y el ranking completo de MITRE:

OWASP Top 10 2007	OWASP Top 10 2004	MITRE 2006 Ranking
A1. Secuencia de Comandos en Sitios Cruzados (XSS)	A4. Secuencia de Comandos en Sitios Cruzados (XSS)	1
A2. Fallas de inyección	A6. Fallas de inyección	2
A3. Ejecución de ficheros malintencionados (NUEVO)		3
A4. Referencia insegura y directa a objetos	A2. Falla de Control de Accesos (dividido en 2007 T10)	5
A5. Falsificación de Petición en Sitios Cruzados (CSRF) (NUEVO)		36
A6. Revelación de información y gestión incorrecta de errores	A7. Gestión Inapropiada de Errores	6
A7. Autenticación y Gestión de Sesiones disfuncionales	A3. Autenticación y Gestión de Sesiones disfuncionales	14
A8. Almacenamiento Criptográfico Inseguro	A8. Almacenamiento Inseguro	8
A9. Comunicaciones Inseguras (NUEVO)	Discutido bajo A10. Gestión Insegura de Configuraciones	8
A10. Falla de restricción de acceso a URL	A2. Falla de Control de Accesos (dividido en 2007 T10)	14
<eliminado en 2007>	A1. Entradas sin validar	7
<eliminado en 2007>	A5. Desbordamiento de Pila	4, 8, y 10
<eliminado en 2007>	A9. Negación de Servicio	17
<eliminado en 2007>	A10. Gestión Insegura de Configuraciones	29

A1 – SECUENCIA DE COMANDOS EN SITIOS CRUZADOS (XSS)

La secuencia de comandos en sitios cruzados, más conocida como XSS, es en realidad un subconjunto de inyección HTML. XSS es la más prevaleciente y perniciosa problemática de seguridad en aplicaciones Web. Las fallas de XSS ocurren cuando una aplicación toma información originada por un usuario y la envía a un navegador Web sin primero validarla o codificando el contenido.

XSS permite a los atacantes ejecutar secuencias de comandos en el navegador Web de la víctima, quienes pueden secuestrar sesiones de usuario, modificar sitios Web, insertar contenido hostil, realizar ataques de phising, y tomar control del navegador Web del usuario utilizando secuencias de comando maliciosas. Generalmente JavaScript es utilizado, pero cualquier lenguaje de secuencia de comandos soportado por el navegador de la victima es un potencial objetivo para este ataque.

ENTORNOS AFECTADOS

Todas las plataformas de aplicación Web son vulnerables a este tipo de ataque.

VULNERABILIDAD

Existen tres tipos conocidos de secuencias de comandos en sitios cruzados: *reflejado, almacenado e inyección DOM.* XSS *reflejado* es el más fácil para explotar – una página reflejara información suministrada por el usuario directamente de vuelta al usuario:

```
echo $_REQUEST['userinput'];
```

XSS *almacenado* toma información hostil, la almacena en un fichero, una base de datos, u otro sistema de almacenamiento, y luego en una etapa posterior, muestra dicha información sin filtrado alguno. Esto es extremadamente peligroso en sistemas de administración de contenidos (CMS), blogs, o foros, donde un gran número de usuarios accederá a la información introducida por otros usuarios.

Con ataques basados en *inyección DOM*, el código JavaScript del sitio Web y sus variables son manipuladas, en lugar de los elementos HTML.

Alternativamente, los ataques pueden ser una mezcla o un híbrido de los tres tipos mencionados anteriormente. El peligro con la secuencia de comandos en sitios cruzados no es el tipo de ataque, sino la posibilidad del mismo. Un comportamiento fuera del estándar o inesperado del navegador Web puede introducir sutiles vectores de ataque.

Los ataques son implementados generalmente en JavaScript, que es un poderoso lenguaje de secuencia de comandos. Utilizar JavaScript permite a los atacantes manipular cualquier aspecto de la pagina mostrada, incluyendo agregar nuevos elementos (tales como una pantalla de conexión falsa que envía nuestras credenciales a un sitio hostil), manipular cualquier aspecto del árbol interno DOM, y borrar o cambiar la manera en la cual una pagina es visualizada. JavaScript permite la utilización de XmlHttpRequest, el cual es utilizado en sitios con tecnologías AJAX, incluso si el sitio de la victima no utiliza AJAX actualmente.

Algunas veces es posible evadir la política que indica al navegador Web enviar la información a su origen utilizando XmlHttpRequest – y por lo tanto reenviar la información de la victima a sitios hostiles, y crear complejos gusanos y zombis maliciosos que se mantendrán activos mientras el navegador Web se encuentre abierto. Los ataques AJAX no tienen por qué ser visibles o requerir la interacción del usuario para realizar peligrosos ataques CSRF

(falsificación de petición en sitios cruzados). Para mayor información sobre este tipo de ataques por favor diríjase al apartado A-5.

VERIFICANDO LA SEGURIDAD

El objetivo es verificar que todos los parámetros en la aplicación sean validados y/o codificados antes de ser incluidos en paginas HTML.

Verificación automatizada: herramientas automáticas de testeo de penetración son capaces de detectar XSS *reflejado* a través de inyección de parámetros, pero generalmente fallan a la hora de encontrar XSS persistente, particularmente si la salida del vector XSS inyectado es restringida a través de pruebas de autorización (tales como si un usuario introduce información hostil que puede ser visualizada solamente por los administradores de la aplicación). Herramientas automáticas de escaneo de código fuente pueden encontrar APIs débiles o peligrosas pero normalmente no pueden determinar si la validación o codificación se ha realizado, lo cual puede resultar en falsos positivos. Ninguna herramienta es capaz de encontrar XSS basado en DOM, lo cual significa que las aplicaciones basadas en Ajax frecuentemente se encontrarán en riesgo si sólo se ha realizado una verificación automatizada.

Verificación manual: si se ha utilizado un mecanismo centralizado de validación y codificación, la manera más eficiente de controlar la seguridad es revisando el código. Si en cambio, se ha utilizado una implementación distribuida, entonces la verificación tomará considerablemente mucho más tiempo. El testeo toma mucho tiempo ya que la superficie de ataque en la mayoría de las aplicaciones es muy amplia.

PROTECCIÓN

La mejor protección para XSS es una combinación de validación de listas blancas (“whitelists”) de toda la información entrante y una apropiada codificación de la información saliente. La validación permite la detección de ataques, y la codificación previene cualquier inyección de secuencia de comandos de ejecutarse exitosamente en el navegador.
Prevenir XSS en una aplicación entera requiere una solución de arquitectura consistente en:

- **Validación de entrada.** Utilizar un mecanismo estándar de validación de entrada para validar toda la información entrante contra longitud, tipo, sintaxis, y reglas de negocio antes de permitir que la información sea visualizada o almacenada. Utilizar una estrategia de validación de “aceptar solo lo bueno”. Rechazar la información inválida en lugar de rehabilitar posible información hostil. No olvidar que los mensajes de errores pueden también contener información inválida.
- **Codificación fuerte de salida.** Asegurar que toda la información suministrada por el usuario sea apropiadamente codificada (sea HTML o XML dependiendo en el mecanismo de salida) antes de devolver la página, mediante la codificación de todos los caracteres a excepción de un pequeño subgrupo. Este es el enfoque Anti-XSS de la librería Microsoft, y próximamente utilizado en la librería OWASP PHP Anti-XSS. También, configurar la codificación de caracteres para cada página de salida, lo cual reducirá la exposición a algunas variantes.
- **Especificación de la codificación de salida.** (tales como ISO 8859-1 o UTF 8). No permitir que el atacante elija esto en nombre de los usuarios.
- **No utilizar la validación de lista negra “blacklist”** para detectar XSS en la entrada o para validar la salida. Buscar y reemplazar solo algunos caracteres (*“<” “>” y otros caracteres similares o frases tales como “script”*) es una técnica débil y ha sido atacada satisfactoriamente con anterioridad. Incluso una etiqueta “*<b>*” sin verificar es inseguro en algunos contextos. XSS tiene un sorprendente número de variantes que hacen sencillo evitar la validación de listas negras.

- **Cuidado con los errores canónicos.** Los valores de entrada deben ser decodificados y canonizados a la representación interna de la aplicación antes de ser validados. Asegurarse que la aplicación no decodifique la misma entrada dos veces. Tales errores pueden ser usados para evadir los esquemas de listas blancas "whitelists" introduciendo entradas peligrosas luego de haber sido controladas.

Recomendaciones específicas de lenguaje:

- **Java: Utilizar mecanismos de salida Struts** tales como *<bean:write..>*, o utilizar el atributo JSTL por defecto *escapeXML="true"* en *<c:out...>*. NO utilizar *<%=....%>* desanidados (esto es, fuera de un mecanismo de salida codificado apropiadamente).
- **NET: Utilizar la librería Microsoft de Anti-XSS** versión 1.5 la cual se encuentra disponible gratuitamente en MSDN. No asignar a los campos de un formulario información directamente desde un objeto Request: *username.Text = Request.QueryString("username")*; sin utilizar esta librería. Comprender qué controles .NET automáticamente codifican la información de salida.
- **PHP: Asegurar que la salida es pasada a través de *htmlentities()* o *htmlspecialchars()*** o utilizar la librería OWASP PHP Anti-XSS que será lanzada próximamente. **Deshabilitar *register_globals*** si aun se encuentra habilitado.

EJEMPLOS

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4206
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-3966
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5204

REFERENCIAS

- CWE: CWE-79, Cross-Site scripting (XSS)
- WASC Threat Classification: http://www.webappsec.org/projects/threat/classes/cross-site_scripting.shtml
- OWASP – Cross site scripting, http://www.owasp.org/index.php/Cross_Site_Scripting
- OWASP – Testing for XSS, http://www.owasp.org/index.php/Testing_for_Cross_site_scripting
- OWASP Stinger Project (A Java EE validation filter) – http://www.owasp.org/index.php/Category:OWASP_Stinger_Project
- OWASP PHP Filter Project - http://www.owasp.org/index.php/OWASP_PHP_Filters
- OWASP Encoding Project - http://www.owasp.org/index.php/Category:OWASP_Encoding_Project
- RSnake, XSS Cheat Sheet, http://ha.ckers.org/xss.html
- Klein, A., DOM Based Cross Site Scripting, http://www.webappsec.org/projects/articles/071105.shtml
- .NET Anti-XSS Library - http://www.microsoft.com/downloads/details.aspx?FamilyID=efb9c819-53ff-4f82-bfaf-e11625130c25&DisplayLang=en

A2 – FALLAS DE INYECCIÓN

Las fallas de inyección, en particular la inyección SQL, son comunes en aplicaciones Web. Existen distintos tipos de inyecciones: SQL, LDAP, XPath, XSLT. HTML, XML, inyección de órdenes en el sistema operativo y otras muchas.

La inyección ocurre cuando los datos proporcionados por el usuario son enviados e interpretados como parte de una orden o consulta. Los atacantes interrumpen el intérprete para que ejecute comandos mal intencionados proporcionando datos especialmente modificados. Las fallas de inyección permiten a un atacante crear, modificar o borrar cualquier información arbitraria disponible en la aplicación. En el peor de los escenarios, estas fallas permiten a un atacante comprometer totalmente la aplicación y los sistemas subyacentes, incluso sobrepasar entornos con cortafuegos.

ENTORNOS AFECTADOS

Todas las plataformas de aplicación Web que usen intérpretes o invoquen otros procesos son vulnerables a las fallas de inyección. Esto incluye cualquier componente del marco de trabajo, que pueda usar intérpretes en la capa de bases de datos.

VULNERABILIDAD

Si la entrada de usuario se pasa a un intérprete sin validación o codificación, la aplicación es vulnerable.

Revisar si se proporciona la entrada de usuario a consultas dinámicas como:

PHP:

```
$sql = "SELECT * FROM table WHERE id = '" . $_REQUEST['id’] . "’";
```

Java:

```
String query = "SELECT user_id FROM user_data WHERE user_name = '" + req.getParameter("userID") +
"' and user_password = '" + req.getParameter("pwd") +"'";
```

VERIFICANDO LA SEGURIDAD

El objetivo es verificar que los datos del usuario no puedan modificar el significado de las órdenes y las consultas enviadas a ninguno de los intérpretes invocados por la aplicación.

Enfoques automatizados: Muchas herramientas de escaneo de vulnerabilidades buscan fallas de inyección, en particular, inyecciones de SQL. Las herramientas de análisis estático que buscan el uso de APIs inseguras del intérprete son prácticas, pero con frecuencia no pueden verificar que pueda existir una validación o codificación apropiada para proteger contra la vulnerabilidad.

Si la aplicación captura errores internos del servidor (501/500), o errores detallados de la base de datos, puede dificultar significativamente el uso de herramientas automáticas, pero el código puede seguir en riesgo. Las herramientas automáticas deben ser capaces de detectar inyecciones de LDAP/XML e inyecciones de XPath.

Enfoques manuales: El enfoque más eficiente y preciso para revisar el código que invoca intérpretes. La persona que revise el código verificará el uso de una API segura o que se haya realizado una validación o codificación

adecuada. El análisis puede consumir tiempo en exceso y puede conllevar una cobertura irregular debido a que la superficie de ataque de la mayoría de aplicaciones es demasiado grande.

PROTECCIÓN

Evitar el uso de intérpretes cuando sea posible. Si se tiene que invocar un intérprete, el mecanismo clave para evitar inyecciones es el uso de APIs seguras, como consultas con parámetros de asignación rigurosa y bibliotecas de mapeo relacional de objetos (ORM)
Estas interfaces manejan todo el escape de datos, o no requieren escaparlos. Tenga en cuenta que aunque las interfaces seguras resuelven el problema, se recomienda usar la validación de todas maneras para detectar ataques.

El uso de intérpretes es peligroso, hay que prestar especial atención en casos como:

- Validación de entrada: Usar un mecanismo estándar de validación de entradas para validar todas las entradas contra el tamaño, el tipo, la sintaxis y las reglas de negocio antes de aceptar los datos que se van a mostrar o almacenar. Usar una estrategia de validación para aceptar las entradas reconocidas como buenas. Rechazar las entradas inválidas en lugar de intentar desinfectar datos potencialmente hostiles. No olvidar que los mensajes de error pueden incluir datos inválidos.
- Usar APIS de consultas con parámetros de asignación rigurosa que reemplacen los parámetros de sustitución, incluso cuando se llame a procedimientos almacenados.
- Conceder los mínimos privilegios cuando se conecte a bases de datos y otros sistemas de bases de datos.
- Evitar los mensajes de error detallados que son útiles para un atacante.
- Usar procedimientos almacenados ya que generalmente no les afectan las inyecciones SQL.. Sin embargo, tener cuidado ya que estos pueden ser inyectables (como a través del uso de exec() o concatenando argumentos a el procedimiento almacenado).
- No usar interfaces de consultas dinámicas (como mysql_query() o similares)
- No usar funciones de escape simples, como addslashes() de PHP o funciones de reemplazo de caracteres como str_replace("'", "''). Estas son débiles y han sido explotadas satisfactoriamente por atacantes. Para PHP, usar mysql_real_escape_string() si se usa MySQL, o preferiblemente usar PDO que no requiere escapar.
- Cuando se usen mecanismos de escape simples, tenga en cuenta que las funciones simples de escape no pueden escapar los nombres de tabla. Los nombres de tabla deben ser sentencias SQL legales, y por lo tanto son completamente inapropiadas como datos de entrada del usuario.
- Preste atención a errores de canonización. Las entradas deben ser decodificadas y formalizadas en la representación interna de la aplicación actual antes de ser validadas. Asegurarse de que la aplicación no decodifica la misma entrada dos veces. Estos errores pueden ser utilizados para evadir esquemas de listas blancas introduciendo entradas peligrosas después de que hayan sido comprobadas.

Recomendaciones específicas del lenguaje:

- Java EE - Usar PreparedStatement de asignación rigurosa, u ORMs como Hibernate o Spring.
- .Net - usar consultas con parámetros de asignación rigurosa, como SQlCommand con SqlParameter o un ORM como Hibernate.
- PHP - Usar PDO con consultas parametrizadas de asignación rigurosa (usando bindParam())

EJEMPLOS

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5121
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4953
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4592

REFERENCIAS

- CWE: CWE-89 (SQL Injection), CWE-77 (Command Injection), CWE-90 (LDAP Injection), CWE-91 (XML Injection), CWE-93 (CRLF Injection), others.
- WASC Threat Classification:
  http://www.webappsec.org/projects/threat/classes/ldap_injection.shtml
  http://www.webappsec.org/projects/threat/classes/sql_injection.shtml
  http://www.webappsec.org/projects/threat/classes/os_commanding.shtml
- OWASP, http://www.owasp.org/index.php/SQL_Injection
- OWASP Guide, http://www.owasp.org/index.php/Guide_to_SQL_Injection
- OWASP Code Review Guide, http://www.owasp.org/index.php/Reviewing_Code_for_SQL_Injection
- OWASP Testing Guide, http://www.owasp.org/index.php/Testing_for_SQL_Injection
- SQL Injection, http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf
- Advanced SQL Injection, http://www.ngssoftware.com/papers/advanced_sql_injection.pdf
- More Advanced SQL Injection, http://www.nextgenss.com/papers/more_advanced_sql_injection.pdf
- Hibernate, an advanced object relational manager (ORM) for J2EE and .NET, http://www.hibernate.org/
- J2EE Prepared Statements, http://java.sun.com/docs/books/tutorial/jdbc/basics/prepared.html
- How to: Protect from SQL injection in ASP.Net,
  http://msdn2.microsoft.com/en-us/library/ms998271.aspx
- PHP PDO functions, http://php.net/pdo

A3 – EJECUCIÓN DE FICHEROS MALINTENCIONADOS

Las vulnerabilidades de ejecución de ficheros malintencionados se encuentran en muchas aplicaciones. Los programadores suelen usar o concatenar directamente una posible entrada hostil con funciones de archivo o conexiones, o confiar indebidamente en ficheros de entrada. En muchas plataformas, el entorno de trabajo permite el uso de referencias a objetos externos, como URLs o referencias al sistema de ficheros. Cuando los datos no son comprobados correctamente, esto puede llevar a que se incluya, procese o ejecute contenido arbitrario hostil y remoto por el servidor Web.

Esto permite a los atacantes realizar:

- Ejecución remota de código
- Instalación remota de rootkits y controlar el sistema completo.
- En Windows, se pueden controlar sistemas internos mediante el uso de encapsulación de ficheros SMB de PHP.

Este ataque es particularmente común en PHP, y se deben tomar precauciones extremas con cualquier función de archivo o flujos de datos para asegurarse que la entrada proporcionada por el usuario no influya en los nombres de ficheros.

ENTORNOS AFECTADOS

Todos los entornos de aplicaciones Web son vulnerables a la ejecución de ficheros malintencionados si aceptan nombres de ficheros o archivos desde los usuarios. Ejemplos comunes incluyen "assemblies" de .NET que permiten URL de nombres de ficheros como argumentos, o código que acepta nombres de fichero proveídos por el usuario para incluirlos en los archivos locales.

PHP es particularmente vulnerable a ataques de tipo "inclusión remota de ficheros" (RFI - remote file include) a través de la manipulación de parámetros con cualquier API basada en archivos o flujos de datos.

VULNERABILIDAD

Un fragmento común de código vulnerable es:

```
include $_REQUEST['filename'];
```

Eso no solo permite la evaluación de scripts hostiles remotos, sino que puede ser usado para acceder a servidores de archivos locales (si PHP está alojado en sistemas Windows) debido al soporte de SMB en los encapsuladores de los sistemas de ficheros de PHP.

Otros métodos de ataque incluyen:

- Datos hostiles que son enviados a archivos de sesiones, a datos de logs y a través de la subida de imágenes (típico de software de foros)
- Usar compresión o flujos de datos de audio, como zlib:// o ogg:// que no inspeccionan la bandera interna en el URL de PHP y permite acceder a recursos remotos incluso si allow_url_fopen o allow_url_include está deshabilitado
- El uso de encapsuladores de PHP, como php://entrada y otros que recogen datos de la petición POST en lugar de un archivo

- El uso de datos de PHP: wrappers como `data:;base64,PD9waHAgcGhwaW5mbygpOz8+`

Como esta lista es extensa (y cambia periódicamente), es vital usar una arquitectura segura y un diseño robusto a la hora de manejar datos proporcionados por los usuarios que influyen en la elección del nombre de archivos del servidor y su acceso.

Aunque se han proporcionado ejemplos en PHP, este ataque es también aplicable de diversas formas a .NET y J2EE. Las aplicaciones escritas en estos entornos de trabajo necesitan prestar especial atención para programar mecanismos seguros de acceso para asegurarse que los nombres de archivos proporcionados por o influenciados por los usuarios no permitan que se obvien los controles de seguridad.

Por ejemplo, es posible que documentos XML enviados por un atacante tengan DTD hostiles que fuercen al analizador XML a cargar DTD remotos, tratar y procesar los resultados. Una empresa de seguridad australiana ha demostrado con este método como escanear puertos detrás de los cortafuegos. Vea [SIF01] en las referencias de este capítulo para más información.

El daño que esta vulnerabilidad en particular causa es directamente proporcional a la fortaleza de los controles de aislamiento de la plataforma/caja de arena del entorno de trabajo. Como PHP raramente es aislado y no tiene ningún concepto de caja de arena o arquitectura de seguridad, el daño es mucho mayor en un ataque que en otras plataformas con confianza limitada o parcial, o que están contenidas en una caja de arena apropiada, como cuando una aplicación Web corre bajo JVM con el gestor de seguridad apropiadamente habilitado y configurado (que raramente es el que trae por omisión).

VERIFICANDO LA SEGURIDAD

Enfoques automatizados: las herramientas de escáner de vulnerabilidades tendrán dificultades para identificar los parámetros que son usados en un archivo adjunto o la sintaxis para hacerlos funcionar. Las herramientas de análisis estático pueden buscar el uso de APIs peligrosas, pero no pueden verificar que exista una validación o codificación apropiada para protegerse de la vulnerabilidad.

Enfoques manuales: una revisión de código es capaz de buscar código que pueda permitir incluir un fichero en la aplicación, pero hay muchos posibles errores que reconocer. Las pruebas también pueden ser capaces de detectar estas vulnerabilidades, pero identificar los parámetros en particular y la sintaxis correcta puede ser complicado.

PROTECCIÓN

La prevención de errores de inclusión de archivos remotos requiere una cuidadosa planificación en las fases de diseño y arquitectura. En general, una aplicación bien escrita no usará ninguna información proporcionada por el usuario en nombres de ficheros para ningún recurso del servidor (como imágenes, documentos de transformación XML y XSL o scripts), y tendrá configuradas reglas en el firewall para evitar conexiones salientes a Internet o internamente a cualquier otro servidor. Sin embargo, muchas aplicaciones heredadas continuarán teniendo la necesidad de aceptar datos proporcionados por los usuarios.

Entre las consideraciones más importantes están:

- **Usar un mapa de referencias indirectas a objetos** (ver la sección A4 para más detalles). Por ejemplo, donde se ha usado una vez un nombre de fichero parcialmente, considere usar cifrado de una vía en la

referencia parcial.

En lugar de:

```
<select name=”language”>
 <option value=”English”>English</option>
```

utilice

```
<select name=”language”>
 <option value=”78463a384a5aa4fad5fa73e2f506ecfc”>English</option>
```

Considere el uso de “sales” para prevenir ataques de fuerza bruta de la referencia indirecta al objeto. De forma alternativa, sólo use índices de valores como 1, 2, 3, y asegúrese que los límites del array son comprobados para detectar una falsificación de parámetros.

- **Utilice mecanismos de comprobación explícita de corrupciones**, si su lenguaje lo soporta. De otro modo, considere un esquema variable de nombres para ayudar con la comprobación de corrupciones.

```
$hostile = &$_POST; // refer to POST variables, not $_REQUEST

$safe[‘filename’]= validate_file_name($hostile[‘unsafe_filename’]); // make it safe
```

De este modo cualquier operación basada en una entrada hostil es inmediatamente evidente.

☒ `require_once($_POST[‘unsafe_filename’] . ‘inc.php’);`

☑ `require_once($safe[‘filename’] . ‘inc.php’);`

- **Validación estricta de la entrada del usuario** usando la estrategia "aceptar lo bueno conocido"
- **Añadir reglas en el cortafuegos** para evitar que los servidores Web realicen nuevas conexiones a sitios Web externos y a los sistemas internos. Para sistemas de alto valor, aislar el servidor Web en su propia VLAN o subred privada.
- **Comprobar cualquier fichero o nombre de fichero proporcionado por el usuario** para propósitos legítimos, que no pueden obviar otros controles. En caso de poder ser obviados, se podría contaminar los datos proporcionados por los usuarios en el objeto de sesión, avatares e imágenes, informes en PDF, archivos temporales, etc.
- **Considere implementar una jaula chroot** o algún otro mecanismo de caja de arena como la virtualización para aislar las aplicaciones entre sí.
- **PHP: deshabilite allow_url_fopen y allow_url_include** en php.ini y considere compilar PHP localmente sin incluir esta funcionalidad. Muy pocas aplicaciones necesitan esta funcionalidad y en estos casos estas opciones deberían habilitarse desde la base de la aplicación.
- **PHP: deshabilite register_globals y utilice E_STRICT para encontrar variables no inicializadas**
- **PHP: asegúrese de que todas las funciones de ficheros y flujos de datos (stream_*) son investigadas cuidadosamente.** Asegúrese de que los datos de usuario no son proporcionados a ninguna función que tenga como argumento un nombre de fichero, incluyendo:

```
include() include_once() require() require_once() fopen() imagecreatefromXXX() file()
file_get_contents() copy() delete() unlink() upload_tmp_dir() $_FILES move_uploaded_file()
```

- PHP: sea extremadamente cauto si pasa datos a system() eval() passthru() o ` (el operador de backtick).
- Con J2EE, asegúrese de que el administrador de seguridad está habilitado y correctamente configurado, y que la aplicación está solicitando permisos correctamente.
- Con ASP.NET, por favor consulte la documentación sobre confianza parcial, y diseñe sus aplicaciones de

forma que sean segmentadas en confianzas, de manera que la mayor parte de la aplicación funcione en el estado de menores permisos posibles.

EJEMPLOS

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0360
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5220
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4722

REFERENCIAS

- CWE: CWE-98 (PHP File Inclusion), CWE-78 (OS Command Injection), CWE-95 (Eval injection), CWE-434 (Unrestricted file upload)
- WASC Threat Classification: http://www.webappsec.org/projects/threat/classes/os_commanding.shtml
- OWASP Guide, http://www.owasp.org/index.php/File_System#Includes_and_Remote_files
- OWASP Testing Guide, http://www.owasp.org/index.php/Testing_for_Directory_Traversal
- OWASP PHP Top 5, http://www.owasp.org/index.php/PHP_Top_5#P1:_Remote_Code_Execution
- Stefan Esser, http://blog.php-security.org/archives/45-PHP-5.2.0-and-allow_url_include.html
- [SIF01] SIFT, Web Services: Teaching an old dog new tricks, http://www.ruxcon.org.au/files/2006/web_services_security.ppt
- http://www.owasp.org/index.php/OWASP_Java_Table_of_Contents#Defining_a_Java_Security_Policy
- Microsoft - Programming for Partial Trust, http://msdn2.microsoft.com/en-us/library/ms364059(VS.80).aspx

A4 – REFERENCIA INSEGURA Y DIRECTA A OBJETOS

Una referencia directa a objetos ("direct object reference") ocurre cuando un programador expone una referencia hacia un objeto interno de la aplicación, tales como un fichero, directorio, registro de base de datos, o una clave tal como una URL o un parámetro de formulario Web. Un atacante podría manipular este tipo de referencias en la aplicación para acceder a otros objetos sin autorización, a menos que se aplique un control de accesos como medida de prevención.

Por ejemplo, en las aplicaciones de banca en línea, es común utilizar el número de cuenta como clave primaria. Por consiguiente, se podría tener la tentación de usar esta clave directamente como parámetro en la interfaz Web. Aún en el caso de que el equipo de desarrollo hubiera utilizado consultas SQL *preparadas* ("parameterized SQL queries") para evitar una inyección SQL, podría ser posible que un atacante modificara este parámetro para ver o cambiar ***todas*** las demás cuentas, si no se verifica también que el usuario es el titular de la cuenta bancaria o está autorizado para acceder a la misma.

Este tipo de ataque fue el utilizado sobre el sitio Web *GST Start Up Assistance*, de la *Australian Taxation Office*, en el cual un atacante con credenciales válidas modificó el parámetro "ABN" (un identificador fiscal) presente en la URL. Dicho atacante consiguió obtener los detalles almacenados en el sistema de 17.000 compañías, y a continuación envió un correo electrónico a cada una de ellas. Fue un escándalo para la Dirección de las compañías afectadas. Aún cuando este tipo de vulnerabilidad es muy común, generalmente no se verifica en las aplicaciones actuales.

ENTORNOS AFECTADOS

Todos los entornos de desarrollo de aplicaciones de Web son vulnerables presentando referencias a objetos internos de la aplicación.

VULNERABILIDAD

Muchas aplicaciones presentan a los usuarios referencias a objetos internos. Un atacante podría manipular los parámetros de entrada a la aplicación cambiando estas referencias, saltándose de esta manera un control de accesos incorrectamente desarrollado. Con frecuencia, estas referencias apuntan a sistemas de ficheros y bases de datos, si bien cualquier otro elemento de la aplicación podría ser vulnerable por un problema de esta categoría.

Por ejemplo, en el caso de que el código utilizara la entrada del usuario para construir nombres de fichero o rutas de acceso a los mismos, podría ser posible que un atacante saliera del directorio donde está la aplicación, y accediera a otros recursos.

```
<select name="language"><option value="fr">Français</option></select>

…

require_once ($_REQUEST['language'']."lang.php");
```

Un código como este puede ser atacado suministrando como entrada una cadena como "../../../../etc/passwd%00" que utiliza la inyección de un byte nulo ("null byte injection", véase la guía OWASP para ampliar información), y acceder así a cualquier fichero del sistema atacado.

De igual forma, es habitual presentar referencias a claves de base de datos. Un atacante puede explotar la vulnerabilidad sencillamente adivinando o buscando otros valores válidos para la clave. A menudo son secuenciales. En el ejemplo que se presenta a continuación, un atacante puede cambiar el parámetro "cartID" para obtener la información asociada a cualquier "carrito de la compra" de la aplicación.

```
int cartID = Integer.parseInt( request.getParameter( "cartID" ) );

String query = "SELECT * FROM table WHERE cartID=" + cartID;
```

VERIFICANDO LA SEGURIDAD

El objetivo es comprobar que la aplicación no presente ninguna referencia directa a un objeto interno de la misma.

Enfoques automáticos: las herramientas de análisis automático de vulnerabilidades podrían tener dificultades para identificar parámetros manipulables, o si dicha manipulación ha tenido éxito. Las herramientas de análisis estático no pueden saber qué parámetros deben ser sometidos a un control de accesos antes de ser utilizados.

Enfoques manuales: mediante una revisión de código se puede rastrear el uso de cada parámetro crítico y saber si podría ser manipulado. Asimismo, mediante una prueba de intrusión se podría saber si dicha manipulación es posible. Sin embargo, ambas técnicas de análisis son costosas en tiempo y es posible que se dejen bastantes posibilidades sin explorar.

PROTECCIÓN

La mejor protección es evitar presentar al usuario cualquier referencia directa a un objeto, mediante el uso de un índice, un mapa de referencias indirectas, o cualquier otro método que sea fácil de verificar. Si es necesario utilizar una referencia directa, se deberá comprobar que el usuario está autorizado a usarla antes de hacerlo.

Es importante establecer una manera estándar para diseñar referencias a objetos de la aplicación:

- **Evitar presentar al usuario referencias a objetos privados de la aplicación siempre que sea posible, por ejemplo claves primarias o nombres de fichero.**
- **Validar cualquier referencia a un objeto privado extensivamente aceptando "sólo los conocidos".**
- **Verificar la autorización a todos los objetos referenciados.**

La mejor solución es el uso de un índice o un mapa de referencias que eviten la manipulación de parámetros.

```
http://www.example.com/application?file=1
```

En caso de que sea necesario presentar referencias a elementos de base de datos, hay que asegurarse de que las sentencias SQL y otros mecanismos de acceso a base de datos sólo permiten que se obtengan los registros autorizados:

```
int cartID = Integer.parseInt( request.getParameter( "cartID" ) );

User user = (User)request.getSession().getAttribute( "user" );

String query = "SELECT * FROM table WHERE cartID=" + cartID + " AND userID=" + user.getID();
```

EJEMPLOS

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0329
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4369

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-0229

REFERENCIAS

- CWE: CWE-22 (Path Traversal), CWE-472 (Web Parameter Tampering)
- WASC Threat Classification:
  http://www.webappsec.org/projects/threat/classes/abuse_of_functionality.shtml
  http://www.webappsec.org/projects/threat/classes/insufficient_authorization.shtml
- OWASP Testing Guide, http://www.owasp.org/index.php/Testing_for_business_logic
- OWASP Testing Guide, http://www.owasp.org/index.php/Testing_for_Directory_Traversal
- OWASP, http://www.owasp.org/index.php/Category:Access_Control_Vulnerability
- GST Assist attack details, http://www.abc.net.au/7.30/stories/s146760.htm

A5 – VULNERABILIDADES DE FALSIFICACIÓN DE PETICIÓN EN SITIOS CRUZADOS (CSRF)

Las vulnerabilidades de falsificación de petición en sitios cruzados no son un ataque nuevo, pero son simples y devastadores. Un ataque CSRF fuerza el navegador validado de una víctima a enviar una petición a una aplicación Web vulnerable, la cual entonces realiza la acción elegida a través de la víctima.

Esta vulnerabilidad está extremadamente extendida, ya que cualquier aplicación que:

- No dispone de pruebas de autorización para acciones vulnerables.
- Procesará una acción si se pueden pasar credenciales predefinidas en la petición (por ejemplo. http://www.example.com/admin/doSomething.ctl?username=admin&passwd=admin).
- Peticiones autorizadas basadas únicamente en credenciales que son automáticamente presentadas como la cookie de sesión si está actualmente conectado en la aplicación, o con la funcionalidad "Recuérdame" si no lo está, o un testigo de Kerberos si forma parte de una intranet integrada con una autenticación con Active Directory

está en riesgo. Desgraciadamente, hoy, la mayoría de las aplicaciones Web confían únicamente en las credenciales presentadas automáticamente tales como cookies de sesión, credenciales de autenticación básica, dirección IP de origen, certificados SSL, o credenciales de dominio Windows.

Esta vulnerabilidad es también conocida por otros nombres en inglés como Session Riding, One-Click Attacks, Cross Site Reference Forgery, Hostile Linking y Automation Attack. El acrónimo XSRF es también usado frecuentemente. OWASP y MITRE han estandarizado el término Cros Site Request Forgery CSRF.

ENTORNOS AFECTADOS

Todos los entornos de aplicación Web son vulnerables a CSRF.

VULNERABILIDAD

Un ataque CSRF típico contra un foro puede hacer que un usuario invoque alguna función, tal como la página de desconexión de la aplicación. La siguiente etiqueta en cualquier página Web vista por la víctima generará una petición que le hará salir de la aplicación:

```
<img src="http://www.example.com/logout.php">
```

Si un banco en línea permitiera a su aplicación procesar peticiones, tales como transferencia de fondos, podría permitir un ataque similar:

```
<img src="http://www.example.com/transfer.do?frmAcct=document.form.frmAcct
  &toAcct=4345754&toSWIFTid=434343&amt=3434.43">
```

Jeremiah Grossman en su conferencia de la BlackHat 2006 donde hablaba sobre Hacking de sitios de Intranet desde el exterior, demostró que es posible forzar a un usuario a realizar cambios en su router DSL sin su consentimiento; incluso aún si el usuario no conoce que el router DSL tiene una interfaz Web. Jeremiah utilizó el nombre predeterminado de la cuenta para realizar el ataque.

Todos estos ataques funcionan porque las credenciales de autorización del usuario (típicamente la cookie de sesión) es automáticamente incluida en tales peticiones por el navegador, incluso aunque el atacante no hubiera suministrado la credencial.

Si la etiqueta que contiene el ataque puede ser anotado (enviado a un foro o weblog) de una aplicación vulnerable, entonces la probabilidad de encontrar víctimas ya conectadas incrementa significativamente, similar al incremento del riesgo entre defectos XSS almacenados y reflejados. No es necesario que exista una vulnerabilidad XSS para que un ataque CSRF funcione, aunque cualquier aplicación con vulnerabilidad XSS es también susceptible a CSRF porque un ataque CSRF puede explotar la vulnerabilidad XSS para robar cualquier credencial suministrada de manera no automática que puede darse para proteger contra un ataque CSRF. Muchos gusanos de aplicación han usado una combinación de ambas técnicas.

Cuando se trabaja en la defensa de los ataques CSRF, debe enfocarse también en la eliminación de las vulnerabilidades XSS de su aplicación ya que tales defectos pueden ser usados para rodear todas las defensas contra CSRF que pueda colocar.

VERIFICANDO LA SEGURIDAD

El objetivo es verificar que la aplicación está protegida ante ataques CSRF mediante la generación y posterior solicitud de un identificador que no es enviado de forma automática por el navegador Web.

Enfoques automáticos: Hoy en día pocos analizadores automáticos tienen la capacidad de detectar vulnerabilidades CSRF, aunque la detección sea posible para motores de escaneo suficientemente capaces. Sin embargo, si su escáner detecta una vulnerabilidad de secuencia de comandos en sitios cruzados y no dispone de protecciones anti-CSRF, es muy probable que tenga el riesgo de ataques CSRF.

Enfoques manuales: Las pruebas de intrusión son una manera rápida de verificar si la protección contra ataques CSRF está en su sitio. Para verificar que el mecanismo es fuerte y debidamente implementado, la evaluación del código fuente es la acción más efectiva.

PROTECCIÓN

Las aplicaciones deben asegurarse de que no dependen de aplicaciones o testigos suministrados automáticamente por los navegadores. La única solución es utilizar un testigo a medida que el navegador no pueda "recordar" e y por lo tanto incluir automáticamente en un ataque CSRF.

Las siguientes estrategias deberían ser inherentes en todas las aplicaciones Web:

- **Asegurarse que no existen vulnerabilidades XSS en su aplicación**. (ver A1 – Secuencia de Comandos en Sitios Cruzados)
- **Introducir testigos aleatorios "a la medida" en cada formulario y URL que no sean automáticamente presentados por el navegador**. Por ejemplo,

```
<form action="/transfer.do" method="post">

<input type="hidden" name="8438927730" value="43847384383">

…
</form>
```

y entonces verificar que el testigo suministrado es correcto para el usuario actual. Tales testigos pueden ser únicos para una función particular o página para ese usuario, o simplemente único para la sesión global. Cuanto más enfocado esté un testigo a una función particular y/o a un conjunto particular de datos, más fuerte será la protección, pero también más complicada de construir y mantener.

- **Para datos delicados o transacciones de gran valor, re-autenticar o utilizar firmado de transacción** para asegurar que la petición es verdadera. Configure mecanismos externos como e-mail o contacto telefónico para verificar las peticiones o notificar al usuario de la petición.
- **No utilice peticiones GET (URLs) para datos delicados o realizar transacciones de valor.** Utilice únicamente métodos POST cuando procese datos delicados del usuario. Sin embargo, la URL puede contener el testigo aleatorio que crea una única URL, la cual hace que el CSRF sea casi imposible de realizar.
- **El método POST aislado es una protección insuficiente**. Debe también combinarlo con testigos aleatorios, autenticación externa o re-autenticación para proteger apropiadamente contra CSRF.
- En ASP.NET, configura un ViewStateUserKey. (Ver referencias). Esto proporciona un tipo similar de chequeo que un testigo aleatorio tal y como se describió arriba.

Mientras estas sugerencias disminuirán su exposición considerablemente, ataques CSRF avanzados pueden evitar muchas de esas restricciones. La técnica más efectiva es el uso de testigos únicos, y eliminar todas las vulnerabilidades XSS de la aplicación.

EJEMPLOS

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0192
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-5116
- MySpace Samy Interview: http://blog.outer-court.com/archive/2005-10-14-n81.html
- An attack which uses Quicktime to perform CSRF attacks http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9005607&intsrc=hm_list

REFERENCIAS

- CWE: CWE-352 (Cross-Site Request Forgery)
- WASC Threat Classification: No direct mapping, but the following is a close match: http://www.webappsec.org/projects/threat/classes/abuse_of_functionality.shtml
- OWASP CSRF, http://www.owasp.org/index.php/Cross-Site_Request_Forgery
- OWASP Testing Guide, https://www.owasp.org/index.php/Testing_for_CSRF
- OWASP CSRF Guard, http://www.owasp.org/index.php/CSRF_Guard
- OWASP PHP CSRF Guard, http://www.owasp.org/index.php/PHP_CSRF_Guard
- RSnake, "What is CSRF?", http://ha.ckers.org/blog/20061030/what-is-csrf/
- Jeremiah Grossman, slides and demos of “Hacking Intranet sites from the outside” http://www.whitehatsec.com/presentations/whitehat_bh_pres_08032006.tar.gz
- Microsoft, ViewStateUserKey details, http://msdn2.microsoft.com/en-us/library/ms972969.aspx#securitybarriers_topic2

A6 – REVELACIÓN DE INFORMACIÓN Y GESTIÓN INCORRECTA DE ERRORES

Las aplicaciones pueden revelar, involuntariamente, información sobre su configuración, su funcionamiento interno, o pueden violar la privacidad a través de una variedad de problemas. También pueden revelar su estado interno dependiendo de cuánto tardan en procesar ciertas operaciones u ofreciendo diferentes respuestas a diferentes entradas, como, por ejemplo, mostrando el mismo mensaje de error con distintos números de error. Las aplicaciones Web suelen revelar información sobre su estado interno a través de mensajes de error detallados o de depuración. Esta información, a menudo, puede ser utilizada para lanzar, o incluso automatizar, ataques muy potentes.

ENTORNOS AFECTADOS

Todos los entornos de aplicaciones Web son vulnerables a la revelación de información y la gestión de errores incorrecta.

VULNERABILIDAD

Las aplicaciones generan, frecuentemente, mensajes de error y los muestran a los usuarios. Muchas veces estos mensajes de error son bastante útiles para los atacantes, ya que revelan detalles de implementación o información que es útil para explotar una vulnerabilidad. Existen algunos ejemplos frecuentes:

- Gestión de errores detallada, donde la inducción de un error muestra demasiada información, como trazas de la pila, sentencias SQL erróneas u otra información de depuración.
- Funciones que producen diferentes resultados a partir de diferentes entradas. Por ejemplo, introducir el mismo usuario con distintas contraseñas a una función de *conexión* debería producir el mismo texto de usuario inexistente y contraseña errónea. Sin embargo, muchos sistemas producen códigos de error diferentes.

VERIFICANDO LA SEGURIDAD

El objetivo consiste en verificar que la aplicación no revela información a través de los mensajes de error o por otros medios.

Enfoques automáticos: las herramientas de rastreo de vulnerabilidades suelen provocar la generación de mensajes de error. Herramientas de análisis estadístico pueden buscar la utilización de APIs que revelen información, pero no son capaces de verificar el significado de tales mensajes.

Enfoques manuales: una revisión del código puede localizar gestiones de error incorrectas y otros patrones que revelan información, pero requiere bastante tiempo. Las pruebas permiten generar mensajes de error, pero determinar qué tipos de error se han validado no es trivial.

PROTECCIÓN

Los desarrolladores deberían usar herramientas como WebScarab, de OWASP, para intentar hacer que sus aplicaciones generen errores. Las aplicaciones que no han sido probadas de esta manera generarán salidas de

errores inesperadas con toda probabilidad. Las aplicaciones deberían, también, incluir una arquitectura de gestión de errores estándar para prevenir revelar información no deseada a los atacantes.

Prevenir la revelación de información requiere disciplina. Las siguientes prácticas han demostrado su eficacia:

- **Garantizar que todo el equipo de desarrollo de software comparte un enfoque común a la gestión de errores**
- **Deshabilitar o limitar la gestión de errores detallada**. En concreto, no mostrar información de depuración a los usuarios finales, trazas de la pila o información sobre rutas
- **Garantizar que los caminos seguros que tienen múltiples resultados devuelvan mensajes de error idénticos o similares** en, prácticamente, el mismo tiempo. Si esto no es posible, considerar imponer un tiempo de espera aleatorio para todas las transacciones para ocultar este detalle al atacante.
- Varios niveles pueden devolver mensajes de excepciones o errores graves, tales como la capa de la base de datos, el servidor Web subyacente (IIS, Apache, etc.). Es vital que **los errores de todos los niveles estén controlados y configurados convenientemente para evitar que los intrusos exploten los mensajes de error.**
- Hay que tener en cuenta que los entornos más comunes devuelven diferentes códigos de error HTTP, dependiendo de si el error se produce en el código personalizado o en el código del *marco de trabajo*. **Vale la pena crear un gestor de errores predeterminado que devuelva un mensaje de error 'esterilizado' apropiadamente para la mayoría de usuarios en producción y para todos los diferentes tipos de error.**
- Aunque es seguridad a través de la oscuridad, decidir anular el gestor de errores predeterminado para que siempre devuelva pantallas de error "200" (OK) reduce la habilidad de las herramientas automáticas de rastreo para determinar cuándo ocurre un error serio. Se trata de "seguridad a través de la oscuridad", pero puede proporcionar una capa extra de defensa.
- Algunas organizaciones grandes han decidido incluir códigos de error aleatorios / únicos entre todas sus aplicaciones. Esto puede ayudar al CAU (Centro de Atención al Usuario) a encontrar la solución correcta a un error particular, pero también puede ayudar a los atacantes a determinar exactamente qué ruta de la aplicación falla.

EJEMPLOS

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-4899
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3389
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0580

REFERENCIAS

- CWE: CWE-200 (Information Leak), CWE-203 (Discrepancy Information Leak), CWE-215 (Information Leak Through Debug Information), CWE-209 (Error Message Information Leak), others.
- WASC Threat Classification: http://www.webappsec.org/projects/threat/classes/information_leakage.shtml
- OWASP, http://www.owasp.org/index.php/Error_Handling
- OWASP, http://www.owasp.org/index.php/Category:Sensitive_Data_Protection_Vulnerability

A7 –AUTENTICACIÓN Y GESTIÓN DE SESIONES DISFUNCIONAL

La autenticación adecuada y la gestión de sesiones son críticas en la seguridad de las aplicaciones Web. Deficiencias en estas áreas, con mucha frecuencia implican fallas en la protección de credenciales y testigos (tokens) de sesión a través de su ciclo de vida. Estos defectos pueden conducir a un robo de usuarios ó cuentas de administración, sabotaje de los controles de autorización y registro, causando violaciones a la privacidad.

ENTORNOS AFECTADOS

Todos los entornos de trabajo Web son vulnerables a fallas de autenticación y gestión de sesiones.

VULNERABILIDAD

Las deficiencias en el mecanismo principal de autenticación no son raras, pero con mayor frecuencia se presentan a través de las funciones auxiliares de autenticación como el cierre de sesión, gestión de contraseñas, expiración de sesión, recuérdame, pregunta secreta y actualización de cuenta.

VERIFICANDO LA SEGURIDAD

El objetivo es verificar que la aplicación autentica correctamente al usuario y protege adecuadamente las identidades y sus credenciales asociadas.

Enfoques automatizados: Con herramientas de escaneo de vulnerabilidades es muy difícil detectar vulnerabilidades en esquemas personalizados de autenticación y gestión de sesiones. No es probable que herramientas de análisis estático detecten problemas de autenticación y gestión de sesiones en código personalizado.

Enfoques manuales: El testeo y la verificación de código, sobre todo en combinación, son bastante eficaces para verificar que la autenticación, gestión de sesiones y las funciones auxiliares hayan sido implementadas correctamente.

PROTECCIÓN

La autenticación se basa en la comunicación segura y el almacenamiento de las credenciales. En primer lugar se debe asegurar que SSL es la única opción para todas las partes autenticadas de la aplicación (véase A9 – Comunicaciones inseguras) y que todas las credenciales se almacenen en "hashes" o de forma cifrada (Véase A8 – Almacenamiento criptográfico inseguro).

La prevención de defectos de autenticación lleva una planificación cuidadosa. Entre las consideraciones más importantes están las siguientes:

- Solo use los mecanismos de gestión de sesiones incorporadas. No escriba o use manejadores de sesión secundarios bajo ninguna circunstancia.
- No acepte identificadores de sesiones nuevas, preestablecidas ó no validas de la URL o en la petición. Esto es llamado “Ataque de fijación de sesión”.
- Limitar o eliminar el código en cookies personalizadas para propósitos de autenticación y gestión de sesiones, tales como la funcionalidad “recuérdame” o autenticación única hecha a la medida. Esto no aplica a soluciones robustas, bien demostrados SSO o soluciones de autenticación unificados.

- Usar un solo mecanismo de autenticación con la fuerza y número de factores apropiados. Asegúrese de que este mecanismo no es fácilmente objeto de suplantación o ataques de reenvío. No haga este mecanismo tan complejo, que pueda ser propenso a un ataque por eso mismo.
- No permita que el proceso de autenticación inicie desde una página no cifrada. Siempre deberá inicia el proceso de acceso desde una segunda página cifrada, con un nuevo testigo (token) para prevenir el robo de credenciales o sesión, ataques de "phishing" y fijación de sesión.
- Considere generar una nueva sesión al término de una autenticación exitosa o cambio en el nivel de privilegios.
- Asegúrese de que cada página tiene un enlace para cerrar la sesión. Al cerrar la sesión se deben destruir todas las sesiones del lado del servidor y las cookies del lado del cliente. Considere el factor humano: No pedir confirmación a los usuarios, el resultado final será que el usuario cierre las pestañas o ventana en lugar de que la sesión sea cerrada con éxito.
- Utilice un periodo de tiempo que automáticamente cierre la sesión inactiva, dependiendo del valor de los datos protegidos (mientras más corto es mejor).
- Utilice únicamente funciones auxiliares de autenticación fuerte (preguntas y respuestas, restablecimiento de contraseñas) ya que estas son credenciales de la misma forma que el nombre de usuario, contraseñas o testigos también son credenciales. Aplique funciones de un solo sentido a las respuestas, para prevenir ataques de divulgación.
- No exponga ningún identificador de sesión ó cualquier porción de credenciales validas en URLs o bitácoras (no sobrescribir las sesiones ó almacenar las contraseñas de los usuarios en archivos de bitácoras).
- Compruebe la contraseña anterior cuando el usuario cambie a una nueva contraseña.
- No confíe en credenciales falsificables como la única forma de autenticación, tales como direcciones IP o máscaras con rangos de direcciones, DNS o resolución inversa de DNS, cabeceras de referenciao similares.
- Tenga cuidado de enviar secretos a direcciones de correo electrónico registradas (ver RSNAKE01 en las referencias) como mecanismo para restablecer contraseñas. Use números aleatorios limitados en tiempo para restaurar el acceso y enviar seguimientos por correo electrónico tan pronto como la contraseña haya sido restaurada. Tenga cuidado de permitir a los usuarios registrados cambiar su dirección de correo electrónico – enviar un mensaje a la cuenta de correo electrónico previa, antes de realizar el cambio.

EJEMPLOS

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6145
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6229
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6528

REFERENCIAS

- CWE: CWE-287 (Authentication Issues), CWE-522 (Insufficiently Protected Credentials), CWE-311 (Reflection attack in an authentication protocol), others.
- WASC Threat Classification:
  http://www.webappsec.org/projects/threat/classes/insufficient_authentication.shtml
  http://www.webappsec.org/projects/threat/classes/credential_session_prediction.shtml
  http://www.webappsec.org/projects/threat/classes/session_fixation.shtml
- OWASP Guide, http://www.owasp.org/index.php/Guide_to_Authentication
- OWASP Code Review Guide, http://www.owasp.org/index.php/Reviewing_Code_for_Authentication
- OWASP Testing Guide, http://www.owasp.org/index.php/Testing_for_authentication
- RSNAKE01 - http://ha.ckers.org/blog/20070122/ip-trust-relationships-xss-and-you

A8 – ALMACENAMIENTO CRIPTOGRÁFICO INSEGURO

Proteger datos delicados con criptografía se ha convertido una parte clave de la mayoría de las aplicaciones Web. Simplemente no cifrar datos delicados está muy extendido. Aplicaciones que sí cifran, frecuentemente contienen criptografía mal diseñada, ya sea usando sistemas de cifrado no apropiados o cometiendo errores serios al usar algoritmos de cifrado sólidos. Estos defectos pueden conducir a la revelación de datos delicados y violaciones de cumplimiento de estándares.

ENTORNOS AFECTADOS

Todos los entornos de aplicaciones Web son vulnerables al almacenamiento criptográfico inseguro.

VULNERABILIDAD

La prevención de fallas criptográficas conlleva una cuidadosa planeación. Los problemas más comunes son:

- No cifrar datos delicados.
- Usar algoritmos creados a la medida.
- Uso inseguro de algoritmos sólidos.
- Continuar utilizando algoritmos débiles (MD5, SHA-1, RC3, RC4, etc…)
- Incrustar llaves en el código fuente y almacenar las llaves en forma insegura.

VERIFICANDO LA SEGURIDAD

El objetivo es verificar que la aplicación cifra adecuadamente información sensible en almacenamiento.

Enfoques automáticos: Herramientas de exploración de vulnerabilidades no pueden comprobar almacenamiento criptográfico en absoluto. Herramientas de exploración de código pueden detectar el uso de la API criptográfica conocida, pero no pueden detectar si esta siendo utilizado debidamente o si el cifrado se realiza en un componente externo.

Enfoques manuales: Al igual que la exploración, las pruebas no pueden verificar el almacenamiento criptográfico. La revisión de código es la mejor manera de comprobar que una aplicación codifica datos delicados y tiene implementado adecuadamente el mecanismo y la gestión de llaves. En algunos casos esto puede involucrar el examinar la configuración de sistemas externos.

PROTECCIÓN

El aspecto más importante es asegurar que todo lo que debería ser cifrado sea en realidad cifrado. Entonces debe asegurarse que la criptografía se aplica correctamente. Como hay tantas maneras de usar la criptografía incorrectamente, las siguientes recomendaciones deberían ser tomadas como parte de su régimen de pruebas para ayudar a garantizar la manipulación segura de materiales criptográficos.

- **No cree algoritmos criptográficos. Solo use algoritmos públicos aprobados como AES, criptografía de llave pública RSA, y SHA-256 o mejor para funciones de cifrado de una vía..**

- **No use algoritmos débiles como MD5 / SHA1. Favorezca alternativas más seguras como SHA-256 o superior.**
- **Genere las llaves fuera de línea y almacene llaves privadas con extremo cuidado. Nunca transmita llaves privadas sobre canales inseguros.**
- **Asegúrese de que las credenciales de infraestructura como credenciales de bases de datos o detalles de acceso a colas de MQ son aseguradas adecuadamente (por medio de estrictos controles y permisos del sistema de archivos), o cifrados con seguridad y no fácilmente descifrados por usuarios locales o remotos.**
- **Asegúrese de que los datos cifrados almacenados en disco no sean fáciles de descifrar. Por ejemplo, la codificación de la base de datos no tiene valor si la cola de conexiones provee acceso sin cifrar.**
- **En virtud del requisito 3 del Estándar de Seguridad de Datos de la Industria de las Tarjetas de Pago (PCI DSS por sus siglas en inglés), debe proteger los datos de los titulares de tarjetas. La conformidad con el PCI DSS es obligatoria para el 2008 para los comerciantes y cualquier otra persona que trate con tarjetas de crédito.**

Una buena práctica es nunca almacenar datos innecesarios, como la información de la banda magnética o el número de cuenta primario (PAN por sus siglas en inglés, también conocido como el número de tarjeta de crédito). Si almacena el PAN, el cumplimiento de los requisitos del DSS es importante. Por ejemplo, NUNCA se permite almacenar el número CVV (el número de tres dígitos en la parte posterior de la tarjeta) bajo ninguna circunstancia. Para mayor información, vea por favor las directrices del PCI DSS e implemente controles según sea necesario.

EJEMPLOS

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6145
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-1664
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1101 (True of most Java EE servlet containers, too)

REFERENCIAS

- CWE: CWE-311 (Failure to encrypt data), CWE-326 (Weak Encryption), CWE-321 (Use of hard-coded cryptographic key), CWE-325 (Missing Required Cryptographic Step), others.
- WASC Threat Classification: No explicit mapping
- OWASP, http://www.owasp.org/index.php/Cryptography
- OWASP Guide, http://www.owasp.org/index.php/Guide_to_Cryptography
- OWASP, http://www.owasp.org/index.php/Insecure_Storage
- OWASP, http://www.owasp.org/index.php/How_to_protect_sensitive_data_in_URL's
- PCI Data Security Standard v1.1, https://www.pcisecuritystandards.org/pdfs/pci_dss_v1-1.pdf
- Bruce Schneier, http://www.schneier.com/
- CryptoAPI Next Generation, http://msdn2.microsoft.com/en-us/library/aa376210.aspx

A9 – COMUNICACIONES INSEGURAS

Con frecuencia las aplicaciones fallan en el cifrado de tráfico de redes cuando es necesario proteger comunicaciones importantes. El cifrado (normalmente SSL) debería ser usado para todas las conexiones autenticadas, especialmente páginas accesibles desde Internet, aunque también para conexiones de bases de datos.

En caso contrario, la aplicación puede dejar expuestos los testigos de autenticación y de sesión.

Además, el cifrado debe ser usado siempre que se transmitan datos delicados, tales como información de tarjetas de crédito o información de salud.

El estándar de PCI exige que toda la información de tarjetas de crédito transmitida a través de Internet esté cifrada.

ENTORNOS AFECTADOS

Todos los entornos de aplicaciones Web son vulnerables a comunicaciones inseguras.

VULNERABILIDAD

Un fallo cifrando comunicaciones delicados significa que un atacante que pueda husmear en el tráfico de una red será capaz de acceder a la conversación, incluso a cualquier credencial o información sensible transmitida. Se debe considerar que hay diferentes redes que serán más o menos susceptibles al husmeo. Sin embargo, es importante comprender que algunas veces un servidor puede ser comprometido en casi cualquier segmento de red, y los atacantes rápidamente instalarán un “husmeador” para capturar las credenciales de otros sistemas. Usar SSL para las comunicaciones con usuarios finales es crítico, ya que estos probablemente usarán redes inseguras para acceder a las aplicaciones.

Debido a que HTTP incluye credenciales de autenticación o testigos de sesión en cada petición, todo el tráfico autenticado necesita ir sobre SSL, no solamente la petición de conexión. El cifrado de comunicaciones con servidores de bases de datos también es importante. Aunque es probable que estas redes sean más seguras, la información y credenciales que ellas portan son más delicadas y más extensas. Por lo que el uso de SSL en el segmento de bases de datos es muy importante. El cifrado de datos delicados, tales como datos de tarjetas de crédito y números de seguridad social se ha convertido en una regulación de privacidad y financiera para muchas organizaciones. El no usar SSL en conexiones que manipulen este tipo de datos crea riesgos de cumplimiento de la regulación.

VERIFICANDO LA SEGURIDAD

El objetivo es verificar que la aplicación cifra correctamente todas las comunicaciones delicadas y autenticadas.

Enfoques automatizados: Herramientas para analizar vulnerabilidades pueden verificar que SSL es usado en la interfaz y detectar muchas fallas relacionadas con SSL. Sin embargo, estas herramientas no tienen acceso a conexiones a bases de datos y no pueden verificar si son seguras. Las herramientas de análisis estático podrían ayudar con análisis de algunas llamadas a sistemas de bases de datos, pero probablemente no serían capaces de entender la lógica particular requerida para todos los tipos de sistemas.

Enfoques manuales: Las pruebas manuales pueden verificar el uso de SSL y detectar muchas fallas relacionadas con SSL en el segmento de interfaz, pero los enfoques automatizados son probablemente más eficientes. La revisión de código es muy eficiente para la verificación de un uso correcto de SSL en las conexiones de bases de datos.

PROTECCIÓN

La protección más importante es usar SSL en cualquier conexión autenticada o siempre que se transmitan datos delicados. Hay una serie de detalles relacionados con la configuración correcta de SSL para aplicaciones Web, por lo que es importante la comprensión y análisis de su entorno. Por ejemplo, IE 7.0 tiene una barra verde para los certificados SSL de "alta confianza", pero este no es un control idóneo para probar el uso seguro de la criptografía solamente.

- Use SSL para todas las conexiones que sean autenticadas, o transmitan datos delicados o de valor, tales como credenciales, detalles de tarjetas de crédito, información de salud u otros datos privados.
- Asegúrese que las comunicaciones entre los elementos de infraestructura, tales como las que hay entre servidores Web y servidores de bases de datos, estén debidamente protegidas con el uso de una capa de transporte segura o un nivel de protocolo de cifrado para credenciales y datos de valor.
- Según el requerimiento 4 de PCI Data Security Standard, se deben proteger los datos en tránsito de los titulares de tarjetas de crédito. En general, los accesos online de clientes, socios, personal y administrativos a los sistemas deben ser cifrados usando SSL o similares. Para más información, por favor, mire "PCI DSS Guidelines" e implemente los controles necesarios.

EJEMPLOS

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-6430
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2005-4704
- http://www.schneier.com/blog/archives/2005/10/scandinavian_at_1.html

REFERENCIAS

- CWE: CWE-311 (Failure to encrypt data), CWE-326 (Weak Encryption), CWE-321 (Use of hard-coded cryptographic key), CWE-325 (Missing Required Cryptographic Step), others.
- WASC Threat Classification: No explicit mapping
- OWASP *Testing Guide*, Testing for SSL / TLS, https://www.owasp.org/index.php/Testing_for_SSL-TLS
- OWASP Guide, http://www.owasp.org/index.php/Guide_to_Cryptography
- Foundstone - SSL Digger, http://www.foundstone.com/index.htm?subnav=services/navigation.htm&subcontent=/services/overview_s3i_des.htm
- NIST, SP 800-52 Guidelines for the selection and use of transport layer security (TLS) Implementations, http://csrc.nist.gov/publications/nistpubs/800-52/SP800-52.pdf
- NIST SP 800-95 Guide to secure Web services, http://csrc.nist.gov/publications/drafts.html#sp800-95

A10 – FALLA DE RESTRICCIÓN DE ACCESO A URL

Comúnmente, la única protección de una URL, es que el enlace a esa página no se muestre a usuarios no autorizados. Sin embargo, un atacante motivado, habilidoso o simplemente con suerte, puede ser capaz de encontrar el acceso a estas páginas, invocar funciones, y visualizar información. La seguridad por oscuridad no es suficiente para proteger la funcionalidad e información en una aplicación. La revisión de control de acceso debe ser realizada antes de que una petición a una función sensible se conceda, lo cual asegura que el usuario está autorizado para acceder a esa función.

ENTORNOS AFECTADOS

Todos los entornos de aplicaciones Web son vulnerables a las fallas de restricciones de acceso a URL.

VULNERABILIDAD

El método primario de ataque para esta vulnerabilidad es llamado “navegación forzada”, que abarca adivinado de enlaces y técnicas de fuerza bruta para encontrar paginas desprotegidas. Las aplicaciones permiten con frecuencia que el código de control de acceso se desarrolle y se separe a través de código fuente, dando como resultado un modelo complejo que sea difícil de entender para los desarrolladores y para los especialistas de la seguridad también. Esta complejidad hace que sea probable que se produzcan errores de páginas perdidas, dejándolas expuestas

Algunos ejemplos comunes de estos defectos incluyen:

- URL “ocultas” o “especiales”, suministradas sólo a administradores o usuarios con privilegios, pero accesibles a todos los usuarios si ellos saben que existen, tales como /administrar/agregarusuario.php o /aprobarTransferencia.do. Esto es particularmente predominante con código de menú.
- Las aplicaciones a menudo permiten el acceso a archivos “ocultos”, tales como XML estáticos o reportes generados por el sistema, confiando en la seguridad por oscuridad para ocultarlos.
- Código que implementa una política de control de acceso pero no esta actualizado o es insuficiente. Por ejemplo, imagine /aprobarTransferencia.do que alguna vez estuvo disponible para todos los usuarios, pero desde que los controles de SOX fueron traídos, se supone que solo está disponible a los ‘aprobadores’. Una solución podría haber sido la de no presentarlo a usuarios no autorizados, pero actualmente no hay un control de acceso implementado, cuando se solicite esa página.
- Código que evalúa privilegios en el cliente pero no en el servidor, como en este ataque en MacWorld 2007 que permite la aprobación de pases “Platinum” por valor de $1700 a través de JavaScript en el navegador en lugar de en el servidor.

VERIFICANDO LA SEGURIDAD

El objetivo es verificar que el control de acceso se aplique de forma consistente en la capa de presentación y la lógica de negocio en todas las URL en la aplicación.

Enfoques automatizados: Los escáneres de vulnerabilidades y los motores de análisis estático tienen dificultades con la verificación de control de acceso a URLs, pero por razones diferentes. Los escáneres de vulnerabilidades tienen dificultades para adivinar las páginas ocultas y determinar qué páginas se deben permitir para cada usuario, mientras que los motores de análisis estático batallan para identificar los controles de acceso personalizado en el código y vincular la capa de presentación con la lógica de negocio.

Enfoques manuales: El método más exacto y eficiente consiste en utilizar una combinación de revisión de código y pruebas de seguridad para verificar el mecanismo de control de acceso. Si el mecanismo es centralizado, la verificación puede ser bastante eficaz. Si el mecanismo esta distribuido a través de todo el código fuente, la verificación puede llevar más tiempo. Si el mecanismo se aplica de manera externa, la configuración debe ser examinada y probada.

PROTECCIÓN

Tomarse el tiempo para planificar la autorización creando una matriz para definir los roles y las funciones de la aplicación es un paso clave en el logro de la protección contra las fallas de restricción de acceso a URL. Las aplicaciones Web deben hacer cumplir el control de acceso en cada URL y en las funciones de negocio. No basta con poner el control de acceso en la capa de presentación y dejar la lógica de negocio sin protección. Tampoco es suficiente revisar una sola vez durante el proceso para garantizar que el usuario esta autorizado, y no volver a comprobar en las etapas subsiguientes. De lo contrario, un atacante simplemente evitará el paso donde la autorización es revisada, y falsificara los valores de los parámetros indicados para continuar en el siguiente paso.

Habilitar control de acceso a URLs requiere una planificación cuidadosa. Entre las consideraciones más importantes se encuentran las siguientes:

- Garantizar la matriz de control de acceso como parte del negocio, la arquitectura y el diseño de la aplicación.
- Garantizar que todas las URLs y las funciones de negocio estén protegidas por un control de acceso eficaz que verifique el rol y los derechos del usuario antes de ejecutar cualquier proceso. Asegúrese de que este se hace durante cada paso, y no sólo una vez al principio de un proceso con etapas intermedias.
- Realice una prueba de intrusión antes de la implantación o entrega del código para asegurar que la aplicación no puede ser usada de manera malintencionada por un atacante experto motivado.
- Preste mucha atención al incluir archivos de librerías, especialmente si tienen una extensión ejecutable tal como .php. Cuando sea posible, deben mantenerse fuera del directorio raíz de la Web. Se debe verificar que no están siendo accedidas directamente, por ejemplo, revisando una constante que solo puede ser creada por el llamante de la biblioteca.
- No asuma que pasarán desapercibidas para los usuarios las URLs ocultas o las APIs. Siempre asegúrese de que las acciones administrativas o de altos privilegios están protegidas.
- Bloquear el acceso a todos los tipos de archivo que su aplicación no deberá servir nunca. Lo ideal sería que este filtro siga el método “aceptar buenos conocidos” y sólo permita tipos de archivos que usted tiene la intención de servir, por ejemplo .html, .pdf, .php. Esto bloqueara cualquier intento de acceder archivos de bitácora, archivo XML, etc., que nunca tuvo la intención de servir directamente.
- Manténgase al día con la protección antivirus y parches a los componentes, tales como procesadores de XML, los procesadores de texto, procesadores de imagen, etc., que manejan la información de usuario suministrada.

EJEMPLOS

- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0147
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-0131
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-1227

REFERENCIAS

- CWE: CWE-325 (Direct Request), CWE-288 (Authentication Bypass by Alternate Path), CWE-285 (Missing or Inconsistent Access Control)
- WASC Threat Classification: http://www.webappsec.org/projects/threat/classes/predictable_resource_location.shtml
- OWASP, http://www.owasp.org/index.php/Forced_browsing
- OWASP Guide, http://www.owasp.org/index.php/Guide_to_Authorization

¿DÓNDE SIGO AHORA?

El Top 10 de OWASP es sólo el comienzo de tu viaje hacia la seguridad en aplicaciones Web.

> *Los seis billones de personas que habitan en este mundo se pueden dividir en dos grupos: grupo uno, que sabe por qué todas las buenas compañías de software producen productos con defectos conocidos; y el grupo dos, que no lo sabe. Aquellos en grupo 1 tienden a olvidarse lo que era la vida en nuestro optimismo juvenil antes de ser arruinada por la realidad. Algunas veces nos encontramos con una persona del grupo 2… quien se encuentra sorprendido que las compañías de software entregan productos sin antes haber corregido todos los defectos.*
>
> *Eric Sink, Guardian May 25, 2006*

La mayoría de sus usuarios y clientes se encuentran en el grupo dos. Cómo resolver este problema es una oportunidad para mejorar su código fuente y el estado de la seguridad de las aplicaciones Web en general. Billones de dólares se pierden cada año, y muchos millones de personas sufren robo de identidad y fraude debido a las vulnerabilidades discutidas en este documento.

PARA ARQUITECTOS Y DISEÑADORES

Para asegurar apropiadamente sus aplicaciones, primero usted debe saber qué es lo que esta asegurando, conocer las amenazas y riesgos, y tratar estos en una manera estructurada.

Diseñar cualquier aplicación no trivial requiere de una buena dosis de seguridad.

- **Asegúrese que aplica solo la "seguridad necesaria" basada en un modelo de riesgo y clasificación de activos.** Sin embargo, como el cumplimiento de las leyes (SOX, HIPAA, Basel, etc) está tomando un papel preponderante, sería apropiado invertir mayor tiempo y recursos que satisfagan el mínimo indispensable, particularmente si las "mejores practicas" son bien conocidas y considerablemente más estrictas que el mínimo requerido.
- **Haga preguntas sobre requerimientos de negocio,** particularmente requerimientos no funcionales que falten.
- Trabaje sobre un Anexo de Contrato en Código Seguro OWASP con su cliente.
- **Fomente el diseño mas seguro –** incluyendo la defensa a profundidad y construcciones más simples usando modelado de amenazas (ver [HOW1] en las referencias de libros)
- **Asegúrese de que ha considerado confidencialidad, integridad, disponibilidad y no repudio**
- **Asegúrese de que sus diseños son compatibles con las políticas de seguridad y normas, tales como COBIT o PCI DSS 1.1**

PARA DESARROLLADORES

Muchos desarrolladores ya tienen un buen manejo básico de seguridad en aplicaciones Web. Para garantizar un manejo efectivo dentro del dominio de seguridad en aplicaciones Web es necesaria la práctica. Cualquiera puede

destruir (ej. Realizar pruebas de penetración) – pero solo un experto puede construir software seguro. Intente convertirse en dicho experto.

- Considere unirse a OWASP y participar de reuniones de capítulos en su área
- **Pregunte por entrenamiento en desarrollo seguro de aplicaciones si dispone de un presupuesto para formación.** En caso que no disponga de un presupuesto para formación, pídalo.
- **Diseñe sus funcionalidades de forma segura** – considere defensa a profundidad y la simplicidad en el diseño.
- **Adopte estándares de programación que fomenten construcciones de código seguro.**
- **Rediseñe código existente para utilizar construcciones más seguras, tales como consultas parametrizadas.**
- **Revise la Guía OWASP y comience a aplicar controles en su código**. A diferencia de la mayoría de estas guías, está diseñada para ayudarlo a construir código seguro, no romperlo.
- **Teste su código en busca de defectos de seguridad y haga de esta práctica un régimen de testeo en su área.**
- **Revise los libros de referencia**, y observe si existe alguno aplicable a su entorno de trabajo.

PARA PROYECTOS DE CÓDIGO ABIERTO

El código abierto es un particular desafió para la seguridad en aplicaciones Web. Existen millones de proyectos de código abierto, desde unipersonales hasta proyectos gigantescos tales como Apache, Tomcat, y aplicaciones Web de larga escala, tales como PostNuke.

- Considere unirse a OWASP y participar de reuniones de capítulos en su área
- Si su proyecto tiene más de 4 desarrolladores, **considere convertir al menos un desarrollador en un experto en seguridad.**
- **Diseñe sus funcionalidades de forma segura** – considere defensa a profundidad y la simplicidad en el diseño.
- **Adopte estándares de programación que fomenten construcciones de código seguro.**
- **Adopte una política de divulgación responsable para asegurar que los defectos de seguridad son manejados apropiadamente.**
- **Revise los libros de referencia**, y observe si existe alguno aplicable a su entorno de trabajo.

PARA DUEÑOS DE APLICACIONES

Los dueños de aplicaciones en ambientes comerciales se encuentran frecuentemente limitados en tiempo y recursos.

Los dueños de aplicaciones deberían:

- Trabajar sobre un Anexo de Contrato en Código Seguro OWASP con los productores del software.
- **Asegúrarse que los requerimientos de negocio incluyan requerimientos no comerciales (NFRs) tales como requerimientos de seguridad.**
- **Alentar diseños que incluyan seguridad por defecto, defensa en profundidad y simplicidad en diseño.**
- **Contratar (o formar) desarrolladores que tengan experiencia en seguridad.**
- **Teste su código en busca de defectos de seguridad en todas las etapas del proyecto**: diseño, construcción, testeo, e implementación
- **Asigne recursos, presupuesto y tiempo en el plan del proyecto para remediar problemas de seguridad.**

PARA EJECUTIVOS DE NIVEL-C

Su organización tiene que disponer de un ciclo de vida de desarrollo seguro (SDLC) que se adecue a su organización. Las vulnerabilidades son menos costosas si se arreglan durante el desarrollo en lugar que después de la implementación del producto. Un SDLC razonable no solo incluye testeo para el Top 10, también incluye:

- Para productos empaquetados, asegúrese que las políticas de compra y los contratos incluyan los requerimientos de seguridad.
- Para código a la medida, **adopte principios de codificación segura en sus políticas y estándares.**
- **Forme a sus desarrolladores en técnicas codificación segura y asegúrese de que mantengan estas habilidades al día.**
- **Incluya herramientas de análisis de código seguro dentro de su presupuesto.**
- **Notifique a sus productores de software la importancia de la seguridad para su organización.**
- **Forme a sus arquitectos, diseñadores, y gente de negocios en los fundamentos de seguridad en aplicaciones Web.**
- **Considere utilizar auditores externos**, que pueden proporcionar una evaluación independiente.
- **Adopte prácticas responsables de divulgación y construya un proceso para responder adecuadamente a reportes de vulnerabilidades en sus productos.**

REFERENCIAS

PROYECTOS OWASP

OWASP es el sitio por excelencia de seguridad en aplicaciones Web. El sitio OWASP contiene muchos proyectos, foros, blogs, presentaciones, herramientas y documentos de investigación. OWASP mantiene dos conferencias de seguridad en aplicaciones Web por año, y tiene más de 80 sedes locales.

Seguramente encontrara de su interés los siguientes proyectos de OWASPinterés (Algunos en Español):

- OWASP Guide to Building Secure Web Applications
- OWASP Testing Guide
- OWASP Code Review Project (en desarrollo)
- OWASP PHP Project (en desarrollo)
- OWASP Java Project
- OWASP .NET Project

LIBROS

Esta no es una lista exhaustiva. Utilice estas referencias para encontrar el área apropiada en su librería y seleccione algunos títulos relacionados que se adecuen a sus requisitos:

- [ALS1] Alshanetsky, I. "*php|architect's Guide to PHP Security*", ISBN 0973862106
- [BAI1] Baier, D., "*Developing more secure ASP.NET 2.0 Applications*", ISBN 978-0-7356-2331-6
- [GAL1] Gallagher T., Landauer L., Jeffries B., "*Hunting Security Bugs*", Microsoft Press, ISBN 073562187X
- [GRO1] Fogie, Grossman, Hansen, Rager, "*Cross Site Scripting Attacks: XSS Exploits and Defense*", ISBN 1597491543
- [HOW1] Howard M., Lipner S., "*The Security Development Lifecycle*", Microsoft Press, ISBN 0735622140
- [SCH1] Schneier B., "*Practical Cryptography*", Wiley, ISBN 047122894X
- [SHI1] Shiflett, C., "*Essential PHP Security*", ISBN 059600656X
- [WYS1] Wysopal et al, *The Art of Software Security Testing: Identifying Software Security Flaws*, ISBN 0321304861

SITIOS WEB

- OWASP, http://www.owasp.org
- MITRE, Common Weakness Enumeration – Vulnerability Trends, http://cwe.mitre.org/documents/vuln-trends.html
- Web Application Security Consortium, http://www.webappsec.org/
- SANS Top 20, http://www.sans.org/top20/

- PCI Security Standards Council, publishers of the PCI standards, relevant to all organizations processing or holding credit card data, https://www.pcisecuritystandards.org/
- PCI DSS v1.1, https://www.pcisecuritystandards.org/pdfs/pci_dss_v1-1.pdf
- Build Security In, US CERT, https://buildsecurityin.us-cert.gov/daisy/bsi/home.html