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:

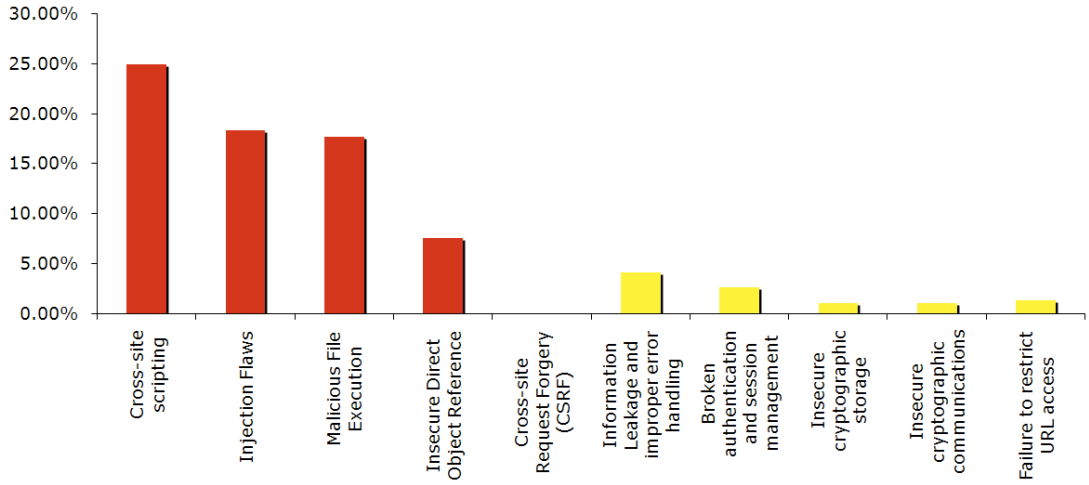

Figura 2: Datos de MITRE sobre las Top 10 vulnerabilidades de aplicaciones Web en 2006

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**	**5**

OWASP Top 10 2007	OWASP Top 10 2004	MITRE 2006 Ranking
	T10)	
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

OWASP

ANEXO PARA CONTRATO DE SOFTWARE SEGURO DE OWASP

Tabla de contenidos

Sobre esta versión

Esta versión del documento 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:

- Juan Carlos Calderón

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

ADVERTENCIA: ESTE DOCUMENTO DEBE SER CONSIDERADO COMO UNA GUÍA SOLAMENTE. OWASP RECOMIENDA VEHEMENTEMENTE QUE CONSULTE UN ABOGADO CALIFICADO PARA AYUDARLE A NEGOCIAR UN CONTRATO DE SOFTWARE

INTRODUCCIÓN

Este anexo de contrato pretende ayudar a los proveedores de software y sus clientes a negociar y capturar importantes términos y condiciones contractuales relacionadas a la seguridad en el Software a ser desarrollado o entregado. La razon para este proyecto es que la mayoría de los contratos no mencionan estos temas, y las partes frecuentamente tienen puntos de vista dramaticamente diferentes a lo que en ralidad fue acordado. Creemos que articular claramente estos términos es la mejor manera de asegurarse que ambas partes pueden hacer desiciones informadas sobre como proceder.

"La seguridad del software comercial mejorará cuando el mercado demande mejor seguridad.
Al menos, cada petición para propuestas de software debería pedir a los vendedores detallar como van ellos
a probar sus productos para encotrar vulnerabilidades de seguridad. Este paso empezará convenciendo a los vendedores
de software empaquetado y proveedores externos que las compañias valoran la seguridad."
-- John Pescatore, Director de investiagacion con Gartner

Pedimos a los clientes y proveedores que usen este documento como un marco para discutir espectativas y negociar responsabilidades. Este anexo se creó para ser añadido a un contrato de desarrollo de software. Estos términos son negociables, significa que ellos pueden y deben ser discutidos por el cliente y el proveedor.

SOBRE EL PROYECTO

Este documento fue creado por la fundación OWASP (Open Web Application Security Project por sus siglas en Inglés), una organización caritativa sin animos de lucro con el objetivo de crear herramientas y documentación, gratuita y abierta para asegurar software. Para facilitar su uso en contratación privada, este documento es de dominio publico. Puede encontrar detalles adicionales sobre este proyecto en http://www.owasp.com/index.php/OWASP_Legal_Project. Son Bienvenidos comentarios de ambos, productores y consumidores de software, asi como de la comunidad legal.

OWASP reconoce gratamente la contribución especial de Aspect Security por su rol en la investigación y preparación de este documento.

OBJECIONES

Las siguientes páginas cubren algunas objeciones que escuchamos frecuentemente sobre usar este lenguaje en contratos de desarrollo de software:

PERO NO TODOS LOS TÉRMINOS SON CORRECTOS PARA NOSOTROS…

Este documento debe ser considerado un punto de inicio para su contrato. Puede que no le gusten todas las actividades, o puede querer proponer mas. Puede querer asignar responsabilidades de manera diferente. Este document no pretende capturar exactamente las necesidades de todos los clientes y proveedores de software. Pretende proveer un marco para discutir temas clave, importantes para asegurarse que el software se cree seguro. Despues de que tengan una discusión sobre seguridad y alcancen un acuerdo. Puede ajustar el convenio para que concuerde con el contrato.

PERO, ¿QUIEN DEBE PAGAR POR ESTAS ACTIVIDADES?…

Este contrato no es sobre poner mas peso al desarollo de software. La pregunta no es sobre si hay costos asociados con seguridad (por supuesto que hay). Sino, la pregunta correcta es que actividades deberían hacerse por ambas partes para minimizar costos, y cuando deberían ocurrir.

Este anexo no menciona (intensionalmente) sobre el detalle de quien debería pagar por la actividades descritas. Mientras muchas de estas actividades deben estar pasando ya, y son esperadas por muchos clientes, ellas no estan practicadas regularmente en la industria del software. La cuestion de quien paga (y cuando) debe ser parte de la negociación.

Calcular el costo de la seguridad es muy difícil. Auque hay costos asociados con realizar actividades de seguridad, hay también costos importantes asociados con ignorar estas actividades. Estamos convencidos el mejor costo-beneficio para desarrollar software es reducir la probabilidad de que fallas de seguridad sean introducidas y minimzar el tiempo entre introducir una falla y arreglarla.

Una distición importante a hacer cuando calculamos costos es entre construir mecanismos de seguridad y las actividades para asegurarse que esos mecanismos sean correctos y efectivos. Intentar agregar mecanismoa al final de el ciclo de desarrollo puede causar serios problemas de diseño e incrementará el costo dramáticamente. Este convenio promieve decisiones sobre mecanismos para minimizar estos costos. Similarmente, esperas hasta justo antes de la publicación para hacer actividades de aseguramiento, tales como revisión de código y pruebas de intrudión, tambien incrementaría los costos dramaticamente. Creemos que el mejor costo-beneficio para ganar certeza es poner un nivel constante de esfuerzo en el aseguramiento a través del ciclo de desarrollo.

PERO EL NIVEL DE RIGOR ESTA INCORRECTO….

Este convenio asume que el software que esta siendo desarrollado es rasonablemente importante para una empresa grande o agencia de gobierno. Hemos seleccionado un

"nivel de rigor" para el convenio que es alcanzable por la mayoria de las organizaciones proveedoras de software, e identificaría y manejaría los riesgos más comunes.

Sin embargo, para el software que va a ser usado en aplicaciones críticas, como aplicaciones médicas, financieras o relacionadas con el ejército. Puede querer incrementar el nivel de rigor. Puede querer agregar revisiones, documentación y actividades adicionales de prueba. Puede querer mejorar el proceso de encontrar, diagnosticar y remediar vulnerabilidades. Para aplicaciones menos sensibles, puede desear reducir o remover actividades.

PERO NOSOTROS NO PODEMOS ACEPTAR TANTO RIESGO…

Este convenio pretende facilitar la discusión sobre quien tomaría el riesgo por las vulnerabilidades de seguridad en el software. Por un lado del espectro, el cliente podria tomar todo el riesgo y el proveedor podria entregar código con muchas vulnerabilidades. Por el otro extremo, el proveedor podría tomar todo el riesgo y asumir la responsabilidad de entregar código perfecto. Ambos extremos son irrealizables.

Actualmente, en este convenio, el proveedor toma el riesgo por problemas que sean descubiertos en los requerimientos o deben ser cubiertos por esfuerzos razonables en pruebas. Pero la remediación de "nuevos" problemas de seguridad tendría que ser pagado por el cliente. Creemos que este es un equilibrio funcional, ya que limita el riesgo adquirido por el proveedor y promueve obtener requerimientos de seguridad por anticipado. Pero hay muchas otras posibles soluciones a este problema. Por favor, dejenos saber si tiene sugerencias alternas, podriamos incluirlas en versiones futuras de este documento.

Note que la discusión aqui solo cubre el riesgo asociado con arreglar las vulnerabilidades de seguridad en el código, y no incluye los costos asociados con recuperarse de incidentes de seguridad relacionados con explotar cualquiera de esas vulnerabilidades. Estamos interesados en mejores prácticas en esta area.

PERO COMO PODEMOS ASEGURAR CÓDIGO DE TERCEROS…

Casi todos los proyectos de sofware tienen una cantidad importante de código de terceros como librarieas, frameworks y productos. Desde la perspectiva de seguridad, este código es tan importante como el código desarrollado a la medida para su proyecto. Creemos que la responsabilidad de asegurar la seguridad de este código es llevada mejor por el proveedor, aunque ellos podrían no tener toda la capacidad para garantizar la seguridad por ellos mismos. Sin embargo, La seguridad puede ser parte de el la desición de "construir o comprar", y esta parece ser la mejor manera de promoverla.

El proveedor, por supuesto, tiene la opción de pasar la responsabilidad a el proveedor del software de terceros. El desarrolador puede tambien analizar el código de terceros por él mismo o contratar expertos en seguridad para analizarlo por él.

PERO PORQUE DEBO PASAR POR TODO ESTE PROBLEMA…

En última instancia, creemos que no hay alternativa para hacer de la seguridad una parte de el proceso de contratación de software. Actualmente, creemos que hay serios malentendidos sobre la seguridad del código que esta siendo desarrollado bajo muchos contratos de desarrollo de software. Esto puede solamente llevar a una cara litigación y a desiciones hechas por individuos con poca experiencia y entendimiento sobre software. Lea el Caso de estudio hipotético sobre contratación de software seguro para una discusión completa de este problema.

Hay muchos beneficios al trabajar con este convenio. El principal es que pondrá claras las expectativas entre las partes involucradas. En algunos casos ayudará a evitar demandas cuando problemas de seguridad difíciles aparecen en el software. Tambien, estan son las mismas actividades requeridas por muchas razónes legales y de conformidad con estándares/leyes.

PERO ES MUY DIFÍCIL PRODUCIR TODA ESTA DOCUMENTACIÓN…

OWASP no promueve la documentacion por la simpre documentacion. Este convenio esta enfocado en promover la calidad, no la cantidad. Creemos que puede ser posible (en algunos proyectos) cumplir con este contrato con una pequeña evaluacion de riesgo, algunas páginas de requerimientos, un pequeño documento de diseño de seguridad, un plan de pruebas, y algunos resultados de pruebas.

El objetivo de esta documentación es simplemente asegurar, en cada etapa del desarrollo de software, que se ha puesto la atención apropiada en la seguridad. Un beneficio adiciona es que esta documentacion puede ser recolectada en conjunto para formar un "paquete de certificación" que básicamente esbose el argumento de porque deberiamos confiar que el software hará lo que dice.

ANEXO DE CONTRATO

1. INTRODUCCIÓN

Esta anexo esta hecho para _____ ("convenio") entre el Cliente y el proveedor. Tanto el Cliente como el proveedor acuerdan maximisar la seguridad del software de acuerdo a los siguientes términos.

2. FILOSOFÍA

Esta anexo pretende aclarar los derechos y obligaciones relacionados a seguridad de todas las partes en una relacion de desarrollo de software. Al mayor nivel, las partes acuerdan que:

(a) Las desiciones de seguridad estarán basadas en el riesgo
    Las desciciones sobre la seguridad seran tomadas en conunto por el cliente y el proveedor basandose en un firme entendimiento de el riesgo involucrado.
(b) Las actividades de seguridad estarán balanceadas
    El esfurzo de seguridad será distribuido equitativamente a través de todo el ciclo de desarollo de software.
(c) Las actividades de seguridad estarán integradas
    Todas las actividades y documentación discutidas aqui pueden y deben ser integradas dentro del ciclo de desarrollo de software del proveedor y no mantenerse separadas de el resto del proyecto. Nada en este anexo implica algún processo de desarrollo de software en particular.
(d) Se esperan vulnerabilidades
    Todo software contiene defectos, y algunos de estos crearan problemas de seguridad. Ambos, cliente y proveedor se esforzará para identificar las vulnerabilidades tan pronto como sea posible en el ciclo de desarrollo.
(e) La información sobre seguridad será comunicada por completo
    Toda la información relevante para la seguridad se compartirá entre el cliente y el proveedor inmediatamente y completamente.
(f) Se requerirá solo documentacion de seguridad necesaria
    La documentación de seguridad no necesita ser demasiado amplia para describir claramente el diseño de seguridad, análisis de riesgo y problemas.

3. ACTIVIDADES EN EL CICLO DE DESARROLLO

(a) Entendimiento del Riesgo
    El proveedor y el cliente acuerdan trabajar juntos para entender y documentar los riesgos que enfrenta la aplicacion. Este esfuerzo debe identificar los riesgos clave para los activos y funciones importantes que provee la aplicación. Cada uno de los temas listados en la sección de requerimientos debe ser considerado.
(b) Requerimientos
    Basado en los riesgos, El proveedor y cliente acuerda trabajar juntos para crear requerimientos de seguridad detallados como parte de la especificacion de el software a ser desarrollado. Cada uno de los temas listados en la sección de requerimientos de este anexo deben ser discutidos y evaluados por ambos, cliente y proveedor. Estos requerimientos pueden ser satisfechos por software creado a la medida, software de terceros, o la plataforma.
(c) Diseño

el desarollador acuerda proveer documentation que explique claramente el diseño para adquirir cada uno de los requerimientos de seguridad. En muchos de los casos, esta documentacion describirá los mecanismos de seguridad, donde se incluiran dentro de la arquitectura, y todos los patrones de diseño relevantes para asegurar su uso adecuado. El diseño debe especificar claramente si tales mecanismos vienen de software a la medida, software de terceros o de la plataforma.

(d) Implementación

El proveedor acuerda entregar y seguir un conjunto de lineamientos de codificación segura. Estos lineamientos indicarán como sele de debe dar formato, estructurar y comentar el código fuente. Todo código relacionado a seguridad debe ser comentado profundamente. Guías sobre como evitar vulnerabilidades se seguridad comunes debe ser incluida. Tambien, todo el código debe ser revisado por al menos otro desarollador basandose en los requerimientos de seguridad y los lineamientos de codificación antes de ser conciderado listo para pruebas unitarias.

(e) Pruebas y análisis de seguridad

El proveedor acuerda entregar y seguir un plan de pruebas de seguridad que defina como se realizarán las pruebas o bien establecer como cada uno de los requerimientos de seguridad va a ser complido. El nivel de rigor de esta actividad debe ser considerado y detallado en un plan. El proveedor ejecutará el plan de prueba y proveera los resultados a el cliente.

(f) Publicación segura

El proveedor acuerda entregar lineamientos de configuración segura que describan completamente todas las opciones de configuracion relacionadas con seguridad y sus implicaciones para la seguridad del software en su conjunto. Los lineamientos deben incluir una descripción completa de las dependencias en la plataforma y como deben ser configuradas de manera segura, incluidas las del sistema operativo, servidor Web y servidor de aplicacion. La configuracion predeterminada del software debe ser segura.

4. ÁREAS CON REQUERIMIENTOS DE SEGURIDAD

Los siguientes temas/áreas deben ser considerados durante las actividades de entendimiento de riesgo y definición de requerimientos. Este esfuerzo debe producir un conjunto de requerimientos ajustados, especificos y probables. Ambos, proveedor y cliente deben ser involucrados en este proceso y deben ponerse de acuerdo sobre el conjunto final de requerimientos.

(a) Validación de entradas y condificación

Los requerimientos deben especificar las reglas para canonicalización, validación y codificación de cada dato de entrada a la aplicación, ya sea de usuarios, sistemas de archivos, bases de datos o sistemas externos. La regla predeterminada debe ser que todas las entradas sean validadas a menos que cumplan con una especificacion detallada de que esta permitido. Además, los requerimientos deben especificar las acciones a tomar cuando se reciven entradas no válidas. Especificamente, la aplicación no debe ser susceptible a inyecciones, desbordamientos, manipulación y otros ataques con entradas de usuario corruptas.

(b) Autentificación y manejo de sessiones

Los requerimientos deben especificar como se protegeran las credenciales para autentificación y los identificadores de sesión a través del ciclo de desarrollo de

software. Los requerimientos para todas las funciones relacionadas deben ser agregadas incluyendo recuperar contraseñas, cambio de contraseñas, recordar contraseñas, desconexión y conexión multiple.

(c) Control de Acceso

Los requerimientos deben incluir una descripcion detallada de todos los roles (grupos, privilegios, autorizaciones) usadas en la aplicación. Los requerimientos deben indicar todos los activos y funciones que provee la aplicacion. Los requerimientos deben especificar detallada y exactamente los derechos de acceso para cualquier activo y función de cada rol. Se sugiere utilizar un formato de matriz de control de acceso para documentar estas reglas.

(d) Manejo de Errores

Los requerimientos deben detallar como se van a manejar los errores que ocurran dentro del procesamiento. Algunas aplicaciones deberían hacer lo mejor posible en caso de un error, mientras que otras deberían terminar su procesamiento inmediatamente.

(e) Historial

Los requerimientos deben especificar que eventos son relevantes para la seguridad y necesitan ser registrados, como ataques detectados, intentos de conexión fallidos e intentos de exeder la autorización. Los requerimientos deben especificar tambien que information registrar con cada evento, incluyendo hora y fecha, descripción del evento, detalles de aplicación, y otra información útil en esfuerzos forences.

(f) Conexiones a sistemas externos

Los requerimientos deben especifica como la autentificación y cifrado será manejado para todos los systemas externos, tales como bases de datos, directorios y servicios Web. Todas las credenciales requeridas para la comunicación con sistemas externos deben almacenarce cifradas y fuera de el código dentro de archivos de configuración.

(g) Cifrado

Los requerimientos deben especificar que datos deben ser cifrados, como serán cifrados y como todos los certificados y otras credenciales deben ser manejadas. Las aplicaciones deben usar algoritmos estándar implementados en una librerias de cifrado que hayan sido usadas y probadas ampliamente.

(h) Disponibilidad

Los requerimientos deben especificar como protegerse de ataques de negación de servicio. Todos los posibles ataques en la aplicacion deben ser considerados, incluyendo bloqueos de auntentificación, agotamiento de conexiones y otros ataques de agotamiento de recursos.

(i) Configuración segura

Los requerimientos deben especificar que los valores predeterminados para todas las configuraciones relacionadas a seguridad deben ser seguras. Para propósitos de auditoría, el software deberia ser capás de producir un reporte sencillo de leer que muestre los detalles de todas las configuraciones relacionadas con seguridad.

(j) Vulnerabilidades especificas

Los requerimientos deben incluir un conjunto de vulnerabilidades especificas que no deben estar presentes en el software. A menos que sea especificado de otra manera, el software no debe incluir ninguna de las fallas descritas en el la "Lista de OWASP sobre las 10 vulnerabilidades mas críticas en aplicaciones Web".

5. PERSONAL Y ORGANIZACIÓN

(a) Arquitecto en seguridad

El desarollador asignará la responsabilidad por la seguridad a un solo recurso técnico experimentado, para ser conocido como el arquitecto de seguridad del proyecto. El arquitecto de seguridad certificará la seguridad de cada entregable.

(b) Entrenamiento en Seguridad

el proveedor será responsable de verificar que todos los miembros del equipo de desarrollo han sido entrenados en tecnicas de programación segura.

(c) Desarrolladores dignos de confianza

El proveedor acuerda realizar las investigaciones de antecendentes apropiadas a todos los miembros del equipo de desarrollo.

6. AMBIENTE DE DESARROLLO

(a) Codificación segura

El proveedor debe mencionar que herramientas se usarán en el ambiente de desarrollo de software para promover la codificación segura.

(b) Adminstración de configuración

El proveedor debe usar un sistema de control de versiones que autentifique y registre el miembro del equipo asociado con todos los cambios en el código base, todos los archivos de configuración y compilación.

(c) Distribución

El proveedor debe usar un proceso de compilación que construya confiablemente un archivo de distribución completo desde el código fuente. Este proceso debe incluir un método para verificar la integridad de el software entregado al cliente.

7. LIBRERIAS, MARCOS DE DESARROLLO Y PRODUCTOS

(a) Apertura

El proveedor debe mencionar todas los aplicaciones de terceros usadas en el software, incluyendo todas las librerias, marcos de desarrollo, componentes y otros productos, ya sean comerciales, gratuitos, de código abierto o código cerrado.

(b) Evaluación

El proveedor debe hacer esfuerzos rasonables para asegurar que el sofware de terceros cumple con todos los términos de este contrato y es tan seguro como el código a la medida desarrollado bajo este contrato.

8. REVISIONES DE SEGURIDAD

(a) Derecho de revisión

El cliente tiene el derecho mandar revisar el software para buscar fallas de seguridad, esto a sus expesas y en cualquier momento dentro de los 60 dias despues de la entrega. El desarrollador acepta proveer apoyo rasonable al equipo de revisión proveyendo el codigo fuente y acceso a los ambientes de pruebas.

(b) Covertura de la revisión

Las revisiones de seguridad deben cubrir todos los aspectos de el software entregado, incluyendo código a la medida, componentes, productos y configuración de sistema.

(c) Alcance de la revisión

Al menos, la revisión debe cubrir todos los requerimientos de seguridad y debería buscar otras vulnerabilidades comunes. La revisión puede incluir una combinación de busqueda automatizada de vulnerabilidades, pruebas de intrusión, análisis estático de código fuente y revisión de código por expertos.

(d) Problemas encontrados

Los problemas de seguridad descubiertos serán reportados a el cliente y proveedor por igual. A todos los problemas se les dará seguimiento y serán

reparados como se especifique en la sección de seguimiento de problemas de seguridad de este anexo.

9. ADMINISTRACIÓN DE PROBLEMAS DE SEGURIDAD

(a) Identificación

El proveedor dara seguimiento a todos los problemas de seguridad descubiertos en todo el ciclo de desarrollo, ya sea un problema en los requerimientos, diseño, implementación, pruebas, publicación u operación. El riesgo asociado con cada problema de seguridad será evaluado, documentado y reportado a el cliente tan pronto como sea posible.

(b) Protección

El proveedor protegerá apropiadamente la información relacionada a problemas de seguridad y la documentación relacionada a ellos, esto, para ayudar a disminuir la probabilidad de las vulnerabilidades en el software operacioneal del cliente sean expuestas.

(c) Reparación

Los problemas de seguridad que sean indentificados antes de la entrega deben ser reparados por el proveedor. Los problemas de seguridad descubiertos despues de la entrega deben ser manejados en la misma manera que otros problemas funcionales según lo especificado en este contrato.

10. CERTEZA

(a) Certeza

El proveedor proveera un "paquete de certificación" consistente en la documentación de seguridad creada a traves del proceso de desarrollo. El paquete deberá establecer que los requerimientos de seguridad, diseño, implementación y resultados de pruebas fueron completados apropiadamente y todos los problemas de seguridad fueron resueltos apropiadamente.

(b) Auto-Certificación

El aquitecto de seguridad certificará que el software cumple con los requerimientos de seguridad, que todas las actividades de seguridad se realizaron, y que todos los problemas de seguridad identificados han sido documentados y resueltos. Cualquier excepción a el estado de la certificacion debe estar totalmente documentada al momento de la entrega.

(c) No código malicioso

El proveedor garantize que el sofware no debe contener ningun código que no se alinee a algun requerimiento del software y debilite la seguridad de la aplicación, incluyendo viruses, gusanos, bombas de tiempo, puertas traseras, caballos de troya, huevos de pascua y cualquier otra forma de código malicioso.

11. CONCENTIMIENTO Y MANTENIMIENTO DE LA SEGURIDAD

(a) Consentimiento

El software no debe ser considerado como aceptado hasta que el paquete de certificación este completo y todos los problemas de seguridad han sido resueltos.

(b) Investigación de problemas de seguridad

Despues del concentimiento, si se sospecha (razonablemente) o descubren problemas de seguridad, el proveedor deberá asisitir al cliente en realizar una investigación para determinar la naturaleza del problema. El problema debe ser

considerado "nuevo" si no es cubierto por los requerientos de seguridad y esta fuera de el alcance (razonable) de las pruebas de seguridad.

(c) Problemas de seguridad nuevos

El proveedor y el cliente consientes en medir el alcance y esfuerzo requerido para resolver problemas de seguridad nuevos, y negociad de buena fe para alcanzar un acuerdo y realizar el trabajor requerido para solucionarlos.

(d) Otros problemas de seguridad

El proveedor debe usar todos los esfuersos comercialmente rasonables consistentes con practicas de desarrollo comunes en la industria, tomando en cuenta la severidad de el riesgo, para resolver todos los problemas de seguridad considerados nuevos tan pronto como sea posible.

Obtenido de
http://www.owasp.org/index.php/Anexo_para_Contrato_de_Software_Seguro_de_OWASP

OWASP

PREGUNTAS FRECUENTES SOBRE SEGURIDAD EN APLICACIONES WEB

(OWASP FAQ)

VERSION 2007 EN ESPAÑOL

TABLA DE CONTENIDO

INTRODUCCIÓN

¿Sobre qué tratan estas preguntas frecuentes?

Esta lista de preguntas frecuentes da respuesta a algunas de las cuestiones que los desarrolladores suelen realizarse acerca de la seguridad en aplicaciones Web. Esta lista no es especifica para una plataforma o lenguaje en particular, discute los problemas y soluciones que son aplicables a cualquier plataforma.

¿Cuáles son las amenazas más usuales en las aplicaciones Web?

A la hora de desarrollar una aplicación, generalmente nos centramos más en la funcionalidad que en la seguridad. Los atacantes se aprovechan de ello explotando la aplicación de diferentes maneras. Las amenazas más comunes en las aplicaciones Web son la inyección de código SQL, el ejecución inter-sitio (Cross Site Scripting), la manipulación de variables y la explotación de funcionalidad importante como el recordatorio de contraseñas, etcétera. Se han creado secciones separadas en esta lista para cada una de estas amenazas.

¿Quién Desarrolló esta lista de preguntas frecuentes?

Esta lista es una documento que evoluciona constantemente con contribuciones de la comunidad de seguridad. Sangita Pakala y su equipo de Paladio Networks desarrollaron la primera versión de esta lista y mantienen esta pagina.

¿Cómo puedo contribuir a esta lista?

Necesitamos sus opiniones y contribuciones para mejorar la lista. Nos encanta escuchar sobre:

- Nuevas preguntas para agregar a la lista
- Mejores respuestas a las preguntas actuales
- Nuevas ligas a documentos y herramientas
- Sugerencias para mejorar esta lista.

Puede enviar sus contribuciones a appsecfaq@owasp.org o mailto:owasp-spanish@lists.sourceforge.net

SOBRE INICIO DE SESIÓN (LOGIN)

¿Qué aspectos debo recordar a la hora de diseñar las páginas de inicio de sesión (login)?

- Desde la página de Inicio de sesión, el usuario deberá ser dirigido a una página de autenticación. Una vez autenticado, deberá enviarse al usuario a la siguiente página. Este tema se trata en la siguiente pregunta.
- La contraseña nunca debe enviarse en claro (sin cifrar) ya que podría ser robada con un rastreador(sniffer). Guardar la contraseña en claro en la base de datos también es peligroso. El mejor método para cifrar y enviar contraseñas es la técnica de cifrado MD5 con sal (Salted MD5).
- La mejor forma de gestionar sesiones es manejar una ficha (token) de sesión con dos valores, uno para antes de la autenticación y otro para después.

¿Es realmente necesario redirigir al usuario a una nueva página después del inicio de sesión?

Consideremos que la aplicación cuenta con una página de entrada que envía al servidor el nombre de usuario y la contraseña mediante una petición POST. Si un usuario actualiza o refresca la segunda página (la que sigue a la de entrada), se enviará de nuevo la misma petición (incluyendo el nombre de usuario y contraseña). Ahora supongamos que un usuario válido entra en la aplicación navega a través de ella y después cierra la sesión pero no cierra la ventana. Un atacante podrá pulsar el botón "Atrás" en el navegador hasta llegar a la segunda página. Bastaría con que actualizase la página para que el nombre de usuario y la contraseña se reenviasen y validasen, completándose el proceso de inicio de sesión con las credenciales del usuario anterior.

Ahora supongamos que página de entrada de la aplicación envía el nombre de usuario y contraseña a una página intermedia de autenticación. Una vez autenticado, el usuario es redirigido a la siguiente página (la que antes era la segunda) con una ficha de sesión. En este caso, aunque el atacante consiga llegar ésta página y pulse en actualizar, el nombre de usuario y contraseña no serán reenviados. Esto ocurre

porque la petición que se reenviará en este caso será la correspondiente a la segunda página, que no contiene el nombre de usuario y contraseña. Por tanto, siempre es mejor redirigir al usuario.

¿Cómo funciona la técnica MD5 con sal (salted-MD5)?

Esta técnica funciona de la siguiente forma: la base de datos almacena el hash MD5 de la contraseña (MD5 es una técnica de encripción de una vía que permite que el valor original no pueda recuperarse). Cuando el cliente hace una petición de la página de entrada, el servidor genera un número aleatorio, la sal, y lo envía al cliente junto con la página. Un código JavaScript de cliente se encarga de aplicar la función hash MD5 a la contraseña proporcionada por el usuario. El mismo código concatena la sal al resultado anterior y vuelve a aplicar MD5 al conjunto. El resultado se envía al servidor. El servidor toma el hash de la contraseña de la base de datos, le concatena la sal y le aplica de nuevo la función hash MD5. Si el usuario ha introducido correctamente la contraseña, los dos resultados deberían coincidir y, en tal caso, el usuario será autenticado.

¿Como puede ser explotada mi función de recordatorio de contraseña?

La función para recordar contraseñas olvidadas puede ser implementada de muchas formas. Una forma muy común es hacer al usuario una pregunta "pista" a la cual el usuario ha enviado la respuesta en el proceso de registro. Las preguntas suelen ser del tipo: ¿Cuál es tu color favorito?, ¿Cuál es tu pasatiempo favorito?, etc. Una vez contestada la pregunta, el sistema bien muestra la contraseña correcta o bien ofrece otra temporal . En este método un atacante trata de robar la contraseña de un usuario puede ser capaz de adivinar la respuesta correcta a la pregunta e incluso reestablecer la contraseña.

En la función de recordar contraseña, ¿Es mejor mostrar la contraseña o permitir al usuario reestablecerla?

Si se muestra la contraseña antigua en la pantalla, esta podría ser vista por otros usuarios. Por tanto, es buena idea no mostrar en la pantalla la contraseña y permitir cambiar a una nueva. mas aún, para poder mostrar la contraseña, ésta deberá almacenarse de forma que se pueda recuperar, lo que no es aconsejable. Si la contraseña se guarda mediante un función hash de una vía, la única forma de implementar una función de este tipo es permitiendo cambiar la contraseña antigua. (un hash es el resultado obtenido cuando pasamos una cadena a una funcion hash de una vía. Es resultado es tal que es imposible regresar al valor original desde él. Las contraseñas son almacenadas mejor en la base de datos como hashes no recuperables)

¿Existe riesgo en enviar la nueva contraseña al correo electrónico autorizado por el usuario?

Enviar la contraseña en claro puede ser arriesgado, ya que un atacante podría obtenerla con un rastreador (sniffer). Además, el correo electrónico que contiene la contraseña puede pasar mucho tiempo en el buzón del usuario y ser visto por un atacante.

¿Cuál es la mejor forma de diseñar la función de recordatorio de contraseñas?

En primer lugar deberemos pedir al usuario que proporcione detalles personales o que responda a un pregunta ”pista”. A continuación deberá enviarse un correo electrónico al usuario incluyendo un enlace a una página en la que podrá reestablecer su contraseña. Este enlace deberá estar activo durante un corto periodo de tiempo, y deberá tener SSL implementado. De esta forma la contraseña real no podrá verse. Las ventajas son: la contraseña no se envía por correo electrónico; dado que el enlace está activo durante poco tiempo, no importa que el correo electrónico se almacene durante un tiempo prolongado en el buzón del usuario.

¿Cómo me protejo ante ataques automatizados para adivinar contraseñas? ¿Debo simplemente bloquear el usuario después de 5 intentos fallidos?

La obtención automatizada de contraseñas resulta un importante problema ya que existe un gran número de herramientas disponibles para este propósito. estas herramientas, básicamente, intentan diferentes contraseñas hasta que una concuerda. Bloquear la cuenta de usuario después de 5 intentos fallidos es una buena defensa en contra de estas herramientas. Sin embargo, lo crucial será determinar cuánto tiempo se mantendrá bloqueada dicha cuenta. Si se bloquea durante demasiado tiempo, un atacante podrá denegar el servicio a usuarios válidos mediante el continuo bloqueo de su cuenta. Si el tiempo es demasiado corto (por ejemplo 1 o 2 minutos), la herramienta de ataque podrá comenzar de nuevo tras ese periodo. El mejor método es insistir en la intervención humana después de un número de intentos fallidos. Un método muy utilizado consiste en mostrar al usuario una palabra aleatoria dentro de una imagen que deberá interpretar e introducir. Dado que esta interpretación no puede ser realizada por una herramienta, podemos frustrar ataques automatizados para adivinar la contraseña.

A continuación se incluyen dos enlaces a herramientas para adivinar contraseñas:

Brutus - http://www.hoobie.net/brutus/

WebCracker - http://www.securityfocus.com/tools/706

¿Y si el atacante a plantado un registrador de tecleos (Keystroke logger) en la computadora cliente? ¿Puedo contrarestar esto?

Los registrador de tecleos infiltrados en las máquinas cliente pueden arruinar todos los esfuerzos por transmitir y almacenar contraseñas de forma segura. Los usuarios por sí mismos pueden no enterarse de que tienen instalado un programa que registra todas las teclas que se pulsan. Dado que el mayor riesgo concierne a la contraseña, si somos capaces de autenticar al usuario sin que tengan que usar el teclado, o sin que tengan que introducir la contraseña completa, el problema estará solucionado. Las formas de hacerlo son:

- Mostrando un teclado gráfico en el que los usuarios tengan que introducir los caracteres seleccionándolos con el ratón. Este resulta especialmente útil para PINs numéricos.

- Pidiendo al usuario que introduzca sólo una parte de la contraseña. Por ejemplo se le podría mostrar el mensaje "Por favor, introduce el primero, tercero y sexto caracteres de tu contraseña", haciendo esta regla aleatoria cada vez.

Mi sitio podrá usarse desde máquinas compartidas públicamente como en bibliotecas. ¿Qué precauciones debo tomar?

Si la aplicación va a ser usada desde lugares públicos como bibliotecas, se deberán tomar las siguiente precauciones:

- Deberemos asegurarnos de que las páginas no quedan en la memoria rápida (cache) de los navegadotes mediante las directivas de control de adecuadas.
- Deberá evitarse incluir información relevante o sensible en la URL, ya que ésta quedará guardada en el historial del navegador.
- Tener un teclado gráfico para escribir la contraseña o pedir al usuario que introduzca diferentes partes de la contraseña cada vez. Esto la protegerá de los registradores de tecleos (Keystroke loggers).
- Para evitar ataques con rastreadores (sniffers), debería usarse SSL o MD5 "con sal" para las contraseñas. Deberá borrarse la contraseña en claro de la memoria una vez aplicada la función MD5.

INYECCIÓN DE SQL

¿Qué es la inyección de SQL?

Se trata de una técnica con la cual un atacante puede ejecutar sentencias SQL en la base de datos mediante la manipulación de la entrada proporcionada a la aplicación. Entendamos esto con un ejemplo de página de entrada de una aplicación Web en la que la base de datos se encuentra en un servidor SQL Server. El usuario necesita introducir su nombre de usuario y contraseña en la página Login.asp. Supongamos que el usuario introduce lo siguiente: nombre de usuario 'Obelix' y contraseña 'Dogmatix'.

Estos datos de entrada son usados para construir dinámicamente la sentencia SQL que tendrá el siguiente aspecto:

SELECT * FROM Users WHERE username= 'Obelix' and password='Dogmatix'

Esta consulta devolvería el registro de la tabla "User" que cumple las condiciones dadas. El usuario se considerará autenticado en el caso de que la consulta devuelva uno o más registros. Supongamos que una atacante teclea lo siguiente en la página de entrada:

Nombre de Usuario: ` **or 1=1--**

La sentencia que se construirá será la siguiente:

SELECT * FROM Users WHERE username='' or 1=1-- and password=''

-- indica a SQL Server que lo que viene a continuación son comentarios. De forma que lo que ejecutará SQL Server será:

SELECT * FROM Users WHERE username='' or 1=1

Esta sentencia lo que hará es buscar en la tabla aquellos registros cuyo nombre de usuario es nulo o cuando se cumpla que 1 es igual a 1 (lo que se cumplirá siempre). Por tanto la consulta devolverá todos los registros de la tabla y el usuario se habrá autenticado sin usuario ni contraseña.

En la siguiente dirección se puede consultar más información al respecto:
http://www.securiteam.com/securityreviews/5DP0N1P76E.html

¿Sólo son ASP y SQL Server vulnerables a este tipo de ataques?

Todas las plataformas son vulnerables a la inyección de SQL. Una inadecuada validación de la entrada y la creación dinámica de las sentencias SQL son las condiciones que permiten este tipo de ataques. La sintaxis a usar para la inyección de SQL dependerá del gestor de base de datos usado. A lo largo de nuestras auditorias hemos encontrado muchas aplicaciones vulnerables corriendo con otros gestores de bases de datos. El ejemplo anterior podría funcionar en SQL Server, Oracle y MySQL, lo que demuestra que el origen del problema no es el tipo de base de datos sino una inadecuada validación de la entrada de usuario junto con la generación dinámica de sentencias SQL.

Además del usuario y contraseña, ¿qué otras variables son candidatas para la inyección de SQL en nuestras aplicaciones?

Cualquier campo de entrada que participa en la creación de la cláusula WHERE de una consulta a una base de datos. Por ejemplo: números de cuentas, números de tarjetas de crédito, etc, en el caso de banca en línea. Además de los campos de un formulario, un atacante puede usar campos ocultos y parámetros del URL para realizar la inyección de comandos

¿Cómo evitamos ataques de inyección de SQL?

Es sencillo proteger una aplicación de la inyección de SQL durante su desarrollo. Deberá chequearse toda la información que provenga del cliente antes de construir una sentencia SQL. El mejor método es eliminar toda entrada no deseada y aceptar tan sólo la esperada.

Aunque la validación de la entrada en el servidor es el método más efectivo, otro método de prevención es evitar el uso de SQL dinámicas. Esto se puede lograr mediante el uso de procedimientos almacenados y parámetros. Para aplicaciones escritas en Java, se pueden usar las sentencias del tipo CallableStatements y PreparedStatements. Para aplicaciones ASP, se puede usar el objeto Command de ADO.

En el siguiente artículo se da más información sobre la inyección SQL en Oracle: http://www.integrigy.com/info/IntegrigyIntrotoSQLInjectionAttacks.pdf

Estoy usando validación de la entrada mediante JavaScripts de Cliente, ¿no es esto suficiente?

No. A pesar de que la validación de la entrada en el cliente impide que un atacante introduzca código malicioso directamente sobre los campos de entrada, solo esto no es suficiente para evitar ataques de Inyección de SQL. Los Scripts de cliente sólo validan la entrada en el navegador. Pero esto no es garantía de que la información será la misma cuando llegue al servidor. Existen herramientas que pueden capturar la información que viaja desde el navegador al servidor y que pueden manipular dicha información antes de ser enviada al servidor. El atacante podría además introducir comandos en las variables de URL, ya que estas no son verificadas por los Scripts de cliente.

Estoy usando procedimientos almacenados para la autenticación, ¿soy vulnerable?

No. El uso de procedimientos almacenados evita la inyección de comandos SQL ya que la entrada del usuario ya no es usada para la construcción dinámica de una sentencia SQL. Dado que un procedimiento almacenado es una colección de sentencias SQL precompiladas y que acepta la entrada como parámetros, se evita la generación dinámica de la consulta. Aunque la entrada es puesta en la consulta precompilada tal cual, dado que la consulta en sí está en un formato diferente. No tiene el efecto de cambiar la consulta como se esperaría. Usando procedimientos almacenados estamos permitiendo a la base de datos que manipule la ejecución de la consulta en ves de pedirle que ejecute una consulta ya construida.

¿Los Servlets de Java son vulnerables a la inyección de SQL?

Sí, lo son si la entrada de usuario no es revisada adecuadamente y las sentencias SQL se construyen dinámicamente. Los Servlets de Java también cuentan con características que evitan la inyección de SQL, como CallableStatements y PreparedStatements. Igual que los procedimientos almacenados y las variables enlazadas, evitan la generación dinámica de las sentencias SQL.

MANIPULACIÓN DE VARIABLES

¿Por qué no puedo fiarme de lo que proviene del navegador?

Hay ocasiones en las que la información es modificada antes de que llegue al servidor. Los atacantes que naveguen por la aplicación pueden modificar la información de una petición POST o GET. Existe un gran número de herramientas como "Aquiles" que son capaces de interceptar la información y que permiten al atacante manipularla. Además, la información que el usuario envía o ve a través de una página Web, tiene que viajar a través de Internet antes de llegar al servidor. Aunque el cliente y el servidor puedan ser confiables, no podemos asegurar que la información no ha sido modificada tras ser enviada por el navegador. Los atacantes podrían capturar la información en tránsito y modificarla.

¿Qué información puede ser manipulada por un atacante?

La manipulación de las variables en las URLs es muy sencilla. Pero los atacantes también pueden modificar la información contenida en campos de formularios y campos ocultos.

¿Cómo manipulan los atacantes la información? ¿Qué herramientas usan?

Para manipular la información, incluyendo campos de formularios, campos ocultos y galletas (cookies), los atacantes usan herramientas conocidas como proxys HTTP. Una vez que el navegador está configurado para funcionar a través del proxy HTTP, la herramienta puede ver toda la información que fluye entre el cliente y el servidor, permitiendo incluso modificar cualquier información antes de ser enviada. Algunas de estas herramientas son:

WebScarab, puede bajarse en www.owasp.org

Achilles. Puede bajarse en http://www.mavensecurity.com/achilles

Odysseus, puede encontrarse en:
http://www.wastelands.gen.nz/odysseus/index.php

Estoy usando SSL, ¿pueden los atacantes modificar la información?

SSL proporciona seguridad en dos aspectos. Primero cuando el cliente se conecta al servidor, puede estar seguro que esta comunicándose con el servidor correcto verificando el certificado que el servidor envía. Segundo, SSL asegura la confidencialidad de los datos, dado que el cliente y el servidor intercambian mensajes encriptados que no pueden ser entendidos por nadie mas.

Aquí esta como funciona SSL: cuando el cliente pide una pagina SSL, el servidor envía un certificado que es obtenido de una autoridad certificadora confiable. El certificado contienen la llave pública del servidor. Después de asegurarse que el certificado es correcto y que el servidor es genuino, el cliente genera un número aleatorio, la llave de sesión. La llave es encriptada con la llave publica del servidor y enviada. El servidor desencripta el mensaje con su llave privada. Ahora ambos lados tienen una llave de sesión conocida solamente por ellos dos. Toda comunicación desde y hacia ellos es encriptada y desencriptada con la llave de sesión.

A pesar de que SSL proporciona un buen nivel de seguridad, no es suficiente. Todas las herramientas mencionadas anteriormente pueden modificar información incluso cuando el sitio esta corriendo sobre SSL. Veamos cómo trabaja Achiles con SSL. Achiles cuenta con un certificado falso con un par de llaves generadas por él mismo. Cuando el cliente realizar una petición con SSL, Achiles la envía tal cual al servidor. Entonces el servidor envía como respuesta su certificado con su llave pública . Achiles lo intercepta, genera una llave de sesión y la manda al servidor cifrada con la llave pública del servidor. De esta forma consigue establecer una conexión SSL con el servidor. Entonces, envía a la parte cliente su propio certificado y llave pública. El navegador cliente mostrará un mensaje diciendo que el certificado no es de confianza y preguntará al usuario si quiere aceptarlo, pero ya que se trata del navegador del atacante, éste acepta el certificado. Entonces el cliente genera un llave de sesión, la cifra con la llave pública de Achiles y la envía. De esta forma ahora Achiles ha establecido dos conexiones SSL – una con el servidor y otra con el cliente. Descifra la información que proviene del servidor y se la muestra en texto plano al atacante y luego la cifra de nuevo con la llave del cliente y la envía . Para el tráfico en la otra dirección utiliza el mismo método.

¿Hay alguna forma de evitar que este tipo de herramienta proxy modifiquen los datos enviados a la aplicación?

La mayor amenaza que representan este tipo de herramientas es la manipulación de la información que viaja del cliente al servidor. Una forma de protegerse es firmando los mensajes enviados desde el cliente mediante un Applet Java cargado en la máquina cliente. Dado que este applet será el encargado de validar el certificado y no el navegador, una herramienta proxy no será capaz de intervenir con un certificado falso. La llave publica del certificado puede entonces ser usada para firmar digitalmente cada mensaje enviado entre el cliente y el servidor. Un atacante tendría que reemplazar el certificado en el applet con un certificado falso, esto incrementa la dificultad para el atacante.

MEMORIA RÁPIDA (CACHE) DEL NAVEGADOR

¿Cómo puede usarse la memoria rápida (cache) del navegador para realizar ataques?

Los navegadores tienen la capacidad de almacenar temporalmente las páginas visitadas. Estos archivos se almacenan en una carpeta (la carpeta Archivos Temporales de Internet en el caso de Internet Explorer). Cuando solicitamos estas páginas de nuevo, el navegador las obtiene de dicha carpeta, lo que resulta mucho más rápido que volver a descargarlas del servidor. Consideremos un escenario en el que un usuario ha realizado el inicio de sesión en una aplicación Web mediante un nombre de usuario y contraseña; el usuario visita diferentes páginas que contienen información sensible, el usuario cierra la sesión. Supongamos el navegador guarda en memoria rápida un página con información de la tarjeta de crédito del usuario. Supongamos que un atacante tiene acceso a la misma máquina y busca entre los archivo temporales de Internet entonces accedería a el número de tarjeta de crédito. El atacante no necesita saber el nombre de usuario y contraseña para acceder a la información de la tarjeta de crédito del usuario.

¿Cómo puede asegurarme de que las páginas con información sensible no se guardan en memoria rápida?

La encabezados en la respuesta enviada por el servidor incluye algunas directivas de control de memoria rápida que pueden ser configuradas mediante código. Esta directivas controlan los contenidos que el navegador del cliente guarda en memoria rápida. Las directivas a configurar son cache-control : no-cache ó cache-control : no-store.

¿Qué diferencia hay entre las directivas de control de memoria rápida "no-cache" y "no-store"?

La directiva "no-cache" indica que la respuesta del servidor no puede ser usada para servir a una nueva petición, es decir, la memoria rápida no mostrará una respuesta que incluya esta directiva y deberá ser el servidor quien la proporcione. La directiva "no-cache" puede incluir algunos nombres de campos, en este caso la respuesta podrá ser mostrada desde la memoria rápida exceptuando aquellos campos que deberán ser proporcionados por el servidor.

La directiva "no-store" se aplica al mensaje completo e indica que no se podrá almacenar en memoria rápida ninguna parte de la respuesta o petición que la generó.

¿Estoy totalmente seguro con estas directivas?

Estas directivas resuelven el problema hasta cierto punto, ya que no están soportadas por las caches HTTP 1.0.

¿Dónde puedo encontrar más información sobre caching?

Algunos enlaces útiles son:

Caching Tutorial for Web Authors and Webmasters por Mark Nottingham en http://www.mnot.net/cache_docs/

RFC de HTTP en:
http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html - sec14.9.1

EJECUCIÓN INTER-SITIO (CROSS SITE SCRIPTING O XSS[1])

¿Qué es el XSS?

La ejecución inter-sitio es un tipo de ataque que puede llevarse a cabo para robar información sensible perteneciente a los usuarios de un sitio Web. Esta confía en que el servidor reflejará la entrada del usuario sin verificar si existe JavaScript incrustado. Esto puede ser usado para robar galletas (Cookies) e identificadores de sesión. Veamos cómo funciona, a menudo nos encontramos con la siguiente situación: escribimos una URL en el navegador, supongamos que es www.abcd.com/mypage.asp, y recibimos una página de error que nos dice que la página "www.abcd.com/mypage.asp" no existe. En otras palabras, la página muestra la información tecleada por el usuario en el navegador. Este tipo de páginas pueden ser explotadas mediante XSS.

Pensemos en qué ocurriría si la entrada contiene un script. En este caso, al devolver dicha entrada al navegador, éste la interpretará no como HTML normal sino como lo que es, un script, ejecutando por tanto los comandos que incluya, pudiendo incluir código maligno.

Un atacante podría enviar a un usuario un enlace que contenga un script como parte de la URL. Cuando el usuario haga clic en el enlace, el script se ejecuta en el navegador del usuario. Dicho script puede hacer sido escrito para enviar al atacante información importante del usuario.

El artículo "Ejecución inter-sitio, ¿son vulnerables tus aplicaciones?" es una buena fuente de información:
http://www.spidynamics.com/whitepapers/SPIcross-sitescripting.pdf

¿Qué información puede robar un atacante mediante XSS?

Un atacante puede robar el identificador de sesión de un usuario válido mediante *XSS*. El identificador de sesión es muy valioso ya que es un elemento secreto que el usuario presenta a una aplicación tras hacer un *inicio de sesión* y hasta que la cierra.

[1] Se usan las siglas XSS para evitar confusiones con las hojas de estilo CSS

Si el identificador de sesión se guarda en una *galleta*, el atacante puede escribir un *script* que se ejecutará en el navegador del usuario y que consultará el valor de la *galleta* para luego enviárselo. El atacante podrá entonces usar el identificador de sesión válido para entrar en la aplicación sin pasar por el proceso de *inicio de sesión*. El *script* podría además recoger otra información de la página, incluyendo todo su contenido.

Aparte de las enlaces a páginas de error enviadas por correo electrónico, ¿hay otros métodos para realizar ataques de XSS?

Sí, hay otros métodos. Pongamos el caso de una aplicación de tablero de anuncios (bulletin board) que contiene páginas en las que lo escrito por un usuario puede ser consultado por el resto, el atacante introduce un *script* en esta página, cuando un usuario válido intenta ver la página, el *script* se ejecuta en el navegador del usuario y podrá enviar información al atacante.

¿Cómo puedo evitar ataques XSS?

Este tipo de ataques se pueden evitar durante el desarrollo (codificación) de la aplicación. Deberá validarse toda la información que entre y salga de y desde la aplicación y "escapar" todos los caracteres especiales que puedan ser usados en un script. Si el código reemplaza los caracteres especiales antes de mostrarlos (según la tabla siguiente), la aplicación estará en gran medida protegida.

<	<
>	>
(	(
)	)
#	#
&	&

¿Me puedo proteger de ataques XSS sin modificar el código de la aplicación?

Existe un método que requiere un codificación mínima comparado con implementar la validación completa de la entrada y salida, y que evita los robos de galletas mediante XSS. El navegador Internet Explorer 6 cuenta con un atributo llamado

HTTP Only que se puede configurar para las galletas. El uso de este atributo asegurar que las galletas no puedan ser accedidas por scripts.
http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dhtml/httponly_cookies.asp
Mozilla también planea implementar una función similar.

Los investigadores han encontrado un método para saltarse esta función, es el método conocido como rastreo inter-sitio (Cross Site Tracing).

¿Qué es el rastreo inter-sitio (Cross Site Tracing ó XST)?¿Cómo puedo evitarlo?

Mediante XST, los atacantes pueden saltarse el atributo *HTTP Only* para robar la información de una *galleta*. *TRACE* es un método de *HTTP* que puede ser enviado al servidor. El servidor envía de vuelta al navegador cualquier información incluida en la petición *TRACE*. En un sitio que usa *galletas*, la información de la *galleta* es enviada al servidor en cada petición. Si enviamos una petición *TRACE* a través en una *URL*, el servidor devolverá toda la información de la *galleta* al navegador. Supongamos ahora una situación similar a la descrita para el caso de *XSS* pero en la que el sitio Web está usando ahora *galletas* de tipo *HTTP Only*. El atacante construye un enlace válido en el que ha incluido un *script* que llama al método *TRACE*. Cuando el usuario hace clic en el enlace, envía al servidor además de la petición *TRACE*, la información de la *galleta*. El servidor entonces devuelve la información de la *galleta* al *script* del navegador. Supongamos que el script contiene además código para enviar la información a los atacantes. Los atacantes han conseguido de esta forma robar la información de galletas HTTP Only. En resumen, HTTP Only evita el acceso directo a las galletas de los JavaScripts, lo que el atacante puede evadir mediante este método de acceso indirecto.

XST puede evitarse deshabilitando el método TRACE en el servidor Web.

El siguiente artículo de Jeremiah Grossman discute en detalle los ataques XST: http://www.cgisecurity.com/whitehat-mirror/WhitePaper_screen.pdf

IDENTIFICACIÓN DEL SERVIDOR WEB

¿Cómo identifican los atacantes qué servidor Web estoy usando?

Identificar la aplicación que está ejecutándose en el servidor Web se conoce como "*Server Fingerprinting*". La forma más sencilla de hacerlo es enviando una petición al servidor y observando el *banner* (encabezado) enviado como respuesta. Los *banners* generalmente incluyen el nombre del servidor y la versión. Podemos evitar este problema configurando el servidor de forma que no muestra el *banner* o cambiándolo para que muestre otra cosa diferente.

¿Cómo puedo falsificar los banners o rescribir las cabeceras desde mi servidor Web?

Existe una serie de herramientas que ayudan a falsificar banners:

URLScan permite cambiar el banner de un servidor Web IIS
http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/tools/URLScan.asp

mod_security incluye una función que permite cambiar la identidad de un servidor Apache. Puede ser encontrada en http://www.modsecurity.org/

Servermask permite falsificar banners de IIS. Puede ser encontrada en http://www.servermask.com/

Una vez falsificado el banner,¿puede mi servidor Web ser identificado?

Sí, desafortunadamente existen herramientas que identifican al servidor sin depender de los banners. Los diferentes servidor Web pueden implementar de forma diferente funciones no especificadas en las HTTP RFCs. Supongamos que hacemos una base de datos con esas peticiones (o funciones) especiales y las respuestas que da cada tipo de servidor. Entonces podremos enviar peticiones al servidor que queremos identificar y compara sus respuestas con las de la base de datos. Esta es la técnica que utilizan herramientas como Fire & Water, que puede encontrarse en la siguiente dirección: http://www.ntobjectives.com/products/firewater/

El siguiente artículo de Saumil Shah analiza la herramienta httpprint:
http://net-square.com/httprint/httprint_paper.html

La herramienta httpprint puede encontrarse en:
http://net-square.com/httprint/

Un amigo me ha contado que es más seguro hacer correr mi servidor bajo un puerto no estándar, como 5001. ¿Es esto verdad?

Un servidor Web normalmente necesita ser accedido por un número grande de usuarios. Dado que los servidores Web corren normalmente bajo el puerto 80, los navegadotes están configurados para acceder a los servidores Web a través del puerto 80. Si se cambia el puerto en el servidor, se obliga a los usuarios que especifiquen el puerto junto con el dominio. Sin embargo, puede ser una buena idea para una aplicación de tipo Intranet en la que todos los usuario saben dónde conectarse. Esto es más seguro ya que así el Servidor Web no será encontrado por ataques automatizados (como gusanos) que verifican el puerto 80 y otros puertos estándar.

¿Debe realmente preocuparme que se identifique mi servidor Web?

sí, se debe proteger al servidor contra su identificación ya que éste puede ser el primer paso de un ataque peligroso. Si un atacante averigua que un sitio Web se basa en IIS 5, por ejemplo, entonces buscará las vulnerabilidades conocidas de este servidor Web. Si el servidor Web no está parchado para todas las vulnerabilidades o el atacante encuentra una nueva para la que aún no existe parche, entonces nada podrá impedir su ataque. Una vez comprometido el servidor Web se puede hacer un gran daño. También se puede despistar a algunas herramientas y gusanos manipulando la información de la versión del servidor. algunos atacantes determinados y persistentes pueden recurrir a medidas adicionales para identificar el servidor, mas los problemas que ahora tienen que sobrepasar son mayores cuando es mas difícil identificar el nombre y versión del servidor.

PRUEBAS (TESTING)

Quiero encadenar un software de tipo proxy con mi servidor proxy, ¿existen herramientas que permitan hacerlo?

Las herramientas que permiten el encadenamiento de proxys. son:

WebScarab - http://www.owasp.org/development/webscarab
Exodus - http://home.intekom.com/rdawes/exodus.html
Odysseus - http://www.wastelands.gen.nz/odysseus/index.php

¿No pueden ser automatizadas las pruebas? ¿Existen herramientas que pueda correr contra mi aplicación?

Existen herramientas que revisan aplicaciones en busca de fallos de seguridad. Pero estas herramientas sólo pueden buscar un número limitado de vulnerabilidades y pueden no encontrar todos los problemas en la aplicación. Añadido a esto, muchos ataques requieren de la comprensión de las reglas de negocio de la aplicación para decidir, por ejemplo, que variables manipular en una petición determinada, cosa que una herramienta es incapaz de realizar.

En la siguiente presentación Jeremiah Grossman analiza las limitaciones de la revisión automática:
http://www.blackhat.com/presentations/bh-federal-03/bh-fed-03-grossman-up.pdf

Algunas herramientas de este estilo son:
SpikeProxy, herramienta open-source y gratuita disponible en:
http://www.immunitysec.com/spikeproxy.html
AppScan, disponible en:
http://www.sanctuminc.com/trials/index.cfm?type=appscanau
WebInspect, puede encontrarse en:
http://www.spidynamics.com/productline/WE_over.html

¿Dónde puedo realizar mis pruebas? ¿Existe una aplicación Web con la que pueda practicar?

OWASP provee una aplicación de ejemplo que puede ser usada para este propósito. WebGoat, como el sitio lo menciona, el objetivo del proyecto en enseñar seguridad en Web en un ambiente interactivo. Hay lecciones sobre la mayoría de las vulnerabilidades comunes .
http://www.owasp.org/development/webgoat

Una aplicación similar para aprender es WebMaven, disponible en http://sourceforge.net/projects/webmaven

Otro sitio interesante es http://www.hackingzone.org/sql/index.php que cuenta con un juego sobre la inyección de SQL.

¿Existen herramientas para el revisión de código fuente que permitan predecir vulnerabilidades para lenguajes como .NET, java, php, etc?

Rough Auditing Tool for Security (*RATS*) es una herramienta que revisa código fuente en busca de fallos de seguridad para los lenguajes *C*, *C++*, *Python*, *Perl* y *PHP*. Pude encontrarla en la siguiente dirección:
http://www.securesoftware.com/download_rats.htm

Nos gustaría saber sobre mas herramientas para revisar código fuente. Si sabe de alguna por favor infórmenos y la agregaremos a esta lista.

¿Pueden otros protocolos diferentes de HTTP ser interceptados y usados para realizar ataques?

Sí. *Interactive TCP Replay* (*ITR*) es una herramienta que actúa como *proxy* para aplicaciones que no se basen en *HTTP* y que además puede modificar el tráfico. Permite la edición de los mensajes en un editor hexadecimal. *ITR* además guarda en un *registro* todos los mensajes entre el cliente y el servidor. Puede usar diferentes tipos de codificación de caracteres (como *ASCII* o *EBCDIC*) para la edición o el registro. Más información en:
http://www.webcohort.com/web_application_security/research/tools.html

GESTIÓN DE GALLETAS(COOKIES) Y SESIONES

Qué son las galletas seguras?

Una galleta puede ser marcada como "secure" lo cual asegura que la galleta solo se envíe en sesiones SSL. Si "secure" no es especificado, la galleta será enviada sin encriptar en canales no SSL. Galletas sensibles como las fichas de sesión deben ser marcadas como seguras si todas las paginas en el sitio Web que requiere fichas de sesión son SSL. Una cosa a mantener en mente aquí es que, generalmente, las imágenes no son bajadas sobre SSL. Así que si una ficha de sesión será presentada para bajar una imagen esta será enviada en texto claro a menos que el atributo "secure" sea utilizado.

¿Puede otro sitio Web robar las galletas que mi sitio Web almacena en la máquina de un usuario?

No, no es posible que un sitio Web acceda a las *galletas* de otro sitio Web. Las *galletas* llevan asociado un atributo de dominio. Sólo una petición que provenga del dominio especificado podrá acceder a la *galleta*. Este atributo tiene un único valor.

¿Existe algún riesgo en usar galletas persistentes frente a galletas no persistentes?

Las *galletas* persistentes son datos que el sitio Web guarda en el disco duro (o el equivalente) del usuario para mantener información entre diferentes sesiones del navegador. Estos datos se mantendrán en el sistema del usuario y podrán ser consultados por el sitio Web la próxima vez que el usuario lo visite.

Las *galletas* no persistentes son aquellas que sólo son usadas durante la sesión de navegador en la que fueron creadas. Se almacenan en la memoria y no en el disco duro.

El riesgo de seguridad de las *galletas* persistentes proviene de que generalmente se almacenan en un archivo de texto en el cliente, de forma que un atacante que accediese a la máquina del usuario podría robar la información.

Si uso un identificador de sesión calculado en función de la IP del cliente, ¿estoy protegido contra el robo de la sesión?

El robo de sesión aun es posible, suponga que el atacante está en la misma red LAN que el usuario y usa la misma dirección IP de proxy que usa el usuario para acceder el sitio Web. El atacante puede aun robar la sesión si puede rastrear el identificador de sesión. Tampoco sería posible implementar esto si el dirección IP del cliente cambia durante la sesión, invalidándola. Esto puede pasar si el cliente viene de un banco de servidores proxy.

¿Cuál es el mejor método para transmitir identificadores de sesión: en galletas, en la URL o en variables ocultas?

Trasmitir los identificadores de sesión en la *URL* conlleva ciertos riesgos. Otros usuarios de la misma máquina pueden ver el identificador de sesión, si la *URL* se guarda en la *memoria rápida* del cliente, el ID de sesión también se guardará con ella. Además, los IDs de sesión podrían guardarse también en los *registros(logs)* de otros sitios en el atributo *referrer* (que guarda la *URL* del sitio del que proviene el usuario). Los campos ocultos (o variable ocultas) no son siempre prácticas ya que toda petición no tiene por qué ser de tipo *POST*. Las *galletas* son el método más seguro ya que no se guardan en *memoria rápida*, no son visibles en los registros *W3C* o *referrer* y la mayoría de los usuarios las aceptan.

¿Es útil cifrar las galletas que guardan el identificador de sesión y enviarlas al servidor?

Cifrar solamente el ID de sesión sobre una conexión sin SSL no sirve ya que el ID de sesión se cifrará una vez y siempre será enviado el mismo valor (el que resulta del cifrado). Un atacante puede usar este valor cifrado para robar la sesión.

¿En qué se basa el concepto de usar un identificador (ID) de página además del identificador de sesión?

El ID o *ficha* de sesión tiene la misma duración que la sesión y está unido al usuario que realizó el *inicio de sesión*. Un ID o *ficha* de página tiene la misma duración que la página y representa una página enviada por el servidor. Se trata de una *ficha* única que se proporciona cuando una página se descarga y que es presentado en el momento de acceder a la siguiente página. El servidor esperará un determinado valor del usuario para acceder a la siguiente página. Sólo si la ficha enviada coincide con el esperado se servirá la siguiente página. Este mecanismo puede usarse para asegura que un usuario accede a las páginas siguiendo la secuencia determinada por la aplicación. Un usuario no podrá por tanto pegar una dirección *URL* en el navegador y saltarse páginas solo porque tiene un ID de sesión, dado que la ficha de página no estará autorizada a acceder un URL mas profundo directamente.

REGISTROS (LOGS) Y AUDITORIA

¿Qué son los registros W3C?

W3C es un formato de registro usado en los archivos *de registro* de los servidores Web. Estos *registros* guardan los detalles de acceso de cada petición: la fecha y hora, IP de origen, página solicitada, el método usado, versión del protocolo *HTTP*, tipo de navegador, la página de origen (*referrer*), el código de respuesta, etc. Nótese de que se trata de registros de acceso, por lo que existirá un registro para cada petición. Cuando se descarga un página con varios ficheros gif, se guardará una entrada por cada uno de ellos; por tanto, estos *registros* suelen ser voluminosos.

¿Necesito mantener un registro en mi aplicación a pesar de tener activados los registros W3C?

Sí, es importante que el sitio Web mantenga un *registro* a nivel de aplicación. Ya que los *registros W3C* guardan una entrada por cada petición *HTTP*, resulta difícil (y en ocasiones imposible) extraer de ellos información útil de "alto nivel". Por ejemplo, resulta muy engorroso identificar en ellos una sesión específica de un usuario y qué acciones ha realizado. Es mejor que la aplicación lleve un registro de las acciones más importantes, en lugar de tener que extraerlas de los *registros W3C*.

¿Qué debo registrar en el registros de mi aplicación?

Deberá guardarse el rastro de toda aquella acción que podamos necesitar revisar para resolver problemas o realizar análisis de lo ocurrido. Es importante resaltar que no es aconsejable guardar información sensible en estos registros, ya que los administradores tienen acceso a los mismos para solucionar problemas. Las acciones que comúnmente se registran son:

- Entrada y salida de sesión para usuarios
- Transacciones críticas (por ejemplo, traspasos de fondos entre cuentas)
- Intentos de inicio de sesión fallidos
- Bloqueos de cuentas
- Violación de políticas

Los datos que normalmente se guardan son:

- Nombre del usuario
- Fecha y hora
- Dirección IP fuente
- Código de error, si procede
- Prioridad

¿Debo cifrar mis registros? ¿No afecta esto al rendimiento?

Se requiere el cifrado cuando la información debe ser protegida de ser leída por usuario no autorizados. Sí, el cifrado supone una carga que afecta al rendimiento, de forma que si los *registros* no contienen información sensible pueden dejarse sin cifrar. Aún así, se recomienda el uso de firmas digitales para proteger los *registros* de posibles manipulaciones. Las firmas digitales son procesos menos costosos que el cifrado.

¿Puedo fiarme de las direcciones IP registradas en mi registro? ¿Puede un usuario personificas o falsificar su dirección IP?

Un atacante que quiera esconder su dirección IP actual podrá usar un servicio como *anonymizer* o usar "*relays"* de HTTP abiertos (se trata de servidores Web "impropiamente" configurados que están accesibles y que son usados como *proxys HTTP* para conectarse a otros sitios). En tales casos la dirección IP que estará viendo en los archivos de registro será aquella de los servicios o el "*relay"* abierto que está siendo usado. Por tanto la dirección IP guardada en el *registro* no siempre es fiable.

CRIPTOGRAFÍA/SSL

¿Debería usar SSL de 40 o de 128 bits?

Existen dos niveles de SSL: del 40 bits y de 128 bits. Este número hace referencia a la longitud de la llave secreta usada para cifrar la sesión. Esta llave es generada para cada sesión SSL y se mantiene durante el resto de la misma. Cuanto más larga sea la llave más difícil será romper la protección de los datos. Por ello, el cifrado de 128 bits es mucho más segura que la de 40 bits. La mayoría de los navegadotes de hoy soportan el cifrado de 128 bits.

¿Realmente SSL de 40 bits es inseguro?

Realmente no es inseguro. Ocurre que computacionalmente puede "romperse", mientras que para el caso de una llave de 128 bits no es así. Aunque en el caso de 40 bits, se necesita un número 200 o mas ordenadores. Nadie ha intentado hacerlo para descubrir un número de tarjeta de crédito o algo por el estilo. Por tanto, dependiendo del tipo de datos que maneje la aplicación, se deberá adoptar una longitud u otra. Usar 128 bits es definitivamente mas seguro.

Tengo que usar SSL de 40 bits, ¿Como puedo asegurarme que estoy protegido adecuadamente?

Con SSL de 40 bits puede necesitar tomar precauciones adicionales para proteger la información sensible. Un "hash con sal" para transmitir contraseñas es una buena técnica. Esto asegura que la contraseña no puede ser robada incluso si la llave SSL es rota.

¿Qué se cifra cuando uso SSL? ¿La petición de páginas también se cifra?

Una vez finalizada la fase de negociación y la conexión HTTPS se ha establecido, se cifra todo, incluidas las peticiones de páginas. Por tanto cualquier dato enviado a través del URL será también cifrado.

¿Qué algoritmos de cifrado usa SSL?

SSL soporta varios algoritmos criptográficos. Durante la fase inicial (llamada fase de saludo) el algoritmo de llave pública RSA. Para cifrar los datos con la llave de sesión, usa los siguientes algoritmos: RC2, RC4, IDEA, DES, triple-DES, y MD5.

Quiero usar SSL, ¿Por dónde debo empezar? O ¿Cómo compro un certificado SSL?

Existen numerosas entidades de certificación en las que se puede comprar un certificado. Independientemente de la autoridad certificadota (CA) que se use, los pasos a seguir son los siguientes:

1. Crear el par de llaves para el servidor
2. Crear la petición de firma de certificado (CSR). Esto requerirá proporcionar cierto detalles como la localización y el nombre completo válido del servidor.
3. Enviar la petición CSR a la autoridad de certificación (CA).
4. Instalar el certificado enviado por la CA.

VARIOS

¿Qué son los cortafuegos de aplicación? ¿Qué tan buenos son en realidad?

Los cortafuegos (o firewalls) de aplicación analizan las peticiones a nivel de aplicación. Estos cortafuegos se usan para aplicaciones específicas como un servidor Web o un servidor de bases de datos. Los cortafuegos de aplicaciones Web protegen al servidor Web de ataques basados en HTTP. Monitorizan las peticiones para detectar ataques de inyección de SQL, XSS, codificación de URL, etc. Pero no protegen contra ataques que requieran comprender la lógica de negocio de la aplicación – lo que incluye la mayoría de los ataques basados en la manipulación de variables. Algunos cortafuegos de aplicación son:

Appshield. Una versión está disponible en:
http://www.sanctuminc.com/forms/regform.cfm?type=appshield

Netcontinuum. Obtenga mas información en:
https://www.netcontinuum.com/index_flash.html

¿En qué consisten los registros referrer (referrer logs) y las URLs sensibles?

Una cabecera HTTP contiene un campo conocido como Referrer. A la hora de visitar una página Web podemos:

- Escribir su dirección URL completa en la barra de direcciones del navegador
- Hacer clic en un enlace de otra página
- Ser redirigidos desde otra página

En el primer caso, el campo referrer estará vacío, pero en los otros dos casos contendrá la URL de la primera página. Las URL de la primera página se guardará en el registro del servidor Web de la segunda página, cuando el usuario llega a esta segunda desde la primera.

Ahora supongamos que las dos páginas residen en diferentes sitios y que la primera URL contiene información sensible como un ID de sesión. Si la segunda página reside en la máquina a la que pueda acceder un atacante, éste podrá usar esta información consultando los registros.

La información de las URLs será almacenada en los registros de referrer y en el navegador. Por tanto, deberemos tener cuidado de no incluir información sensible en la URL.

¿Quiero usar el lenguaje más seguro?¿Cuál es más recomendable?

Puede ser usado cualquier lenguaje, ya que la seguridad de una aplicación Web reside fundamentalmente en las técnicas o prácticas usadas a la hora de programar. Nuestro consejo es usar cualquier lenguaje en el que uno se sienta cómodo. Al margen de esto, hay algunos lenguajes como Java que incluyen características o funciones especiales como las variables enlazadas (bind) que aportan mayor seguridad. Usted puede usar esas características adicionales si decide programar en ese lenguaje.

¿Qué libros son aconsejables para aprender técnicas o prácticas de programación segura?

La guía de OWASP para construir Aplicaciones Web y Servicios Web Seguros (The OWASP Guide to Building Secure Web Application and Web Services). Es una buena guia para los desarrolladores de aplicaciones Web. http://www.owasp.org/documentation/guide

Escribiendo código seguro (Writing Secure Code) por Michael Howard y David LeBlanc. Tiene un capítulo sobre asegurar servicios basados en Web. puede encontrar mas información sobre este libro en:
http://www.dwheeler.com/secure-programs

Programación segura para Linux y Unix (Secure Programming for Linux and Unix HOWTO) por David Wheeler por talks habla sobre como escribir aplicaciones segura incluyendo aplicaciones Web; también especifica algunas guías para varios lenguajes. El libro puede ser encontrado en:
http://www.dwheeler.com/secure-programs

Hay algún programa de entrenamiento sobre programación segura al cual pueda atender?

Microsoft ofrece programas de entrenamiento para desarrollar aplicaciones Web mejoradas en seguridad (security-enhanced), también, desarrollar y desplegar aplicaciones seguras con el marco de Microsoft .net. Puede encontrar mas información en:
http://www.microsoft.com/traincert/syllabi/2300AFinal.asp y
http://www.microsoft.com/traincert/syllabi/2350BFinal.asp

SPI Dynamics enseña un curso en seguridad en aplicaciones Web. Puede encontrar mas información en:
http://www.spidynamics.com/training/