OWASP
Open Web Application
Security Project

FOR DEVELOPERS

2018 v 3.0

10 Critical Security Areas That Software Developers Must Be Aware Of

PROJECT LEADERS

KATY ANTON
JIM MANICO
JIM BIRD

Sobre OWASP

El *Open Web Application Security Project* (OWASP) es una caridad educacional sin fines de lucro (501c3) dedicada a habilitar a organizaciones a diseñar, desarrollar, adquirir, operar, y mantener software seguro. Todas las herramientas, documentos, foros, y capítulos de OWASP son gratis y abiertos a cualquier interesado en mejorar la seguridad de las aplicaciones. Nos pueden encontrar en www.owasp.org.

OWASP es un nuevo tipo de organización. Nuestra libertad de presiones comerciales nos permite proveer información imparcial, práctica y económica sobre seguridad de las aplicaciones.

OWASP no está afiliado a ninguna compañía de tecnología. Similar a muchos proyectos de software open source, OWASP produce muchos tipos de material en forma colaborativa y abierta. La Fundación OWASP es una entidad sin fines de lucro que se asegura del éxito a largo plazo del proyecto.

PREFACIO

El software inseguro está socavando nuestra infraestructura global financiera, de cuidado de salud, de defensa, de energía y otras infraestructuras críticas. A medida que nuestra infraestructura global digital se vuelve cada vez más compleja e interconectada, la dificultad de lograr la seguridad de las aplicaciones aumenta exponencialmente. We can no longer afford to tolerate relatively simple security problems.

OBJETIVOS

El objetivo de *OWASP Top 10 Proactive Controls project (OPC)* es crear conciencia sobre la seguridad de las aplicaciones describiendo las partes más importantes de preocupación que los desarrolladores de software deben tener en cuenta. Alentamos que uses OWASP Proactive Controls para que tus desarrolladores entren al mundo de la seguridad de las aplicaciones. Los desarrolladores pueden aprender de los errores de otras organizaciones. Esperamos que OWASP Proactive Controls sea útil para tus esfuerzos en construir software seguro.

CONTACTO

Por favor no duce en entrar en contacto con el proyecto de OWASP de Control Proactivo con sus preguntas, comentarios, e ideas, tanto públicamente en nuestra lista de mails como en forma privada a jim@owasp.org.

COPYRIGHT Y LICENCIAS

LÍDERES DE PROYECTO

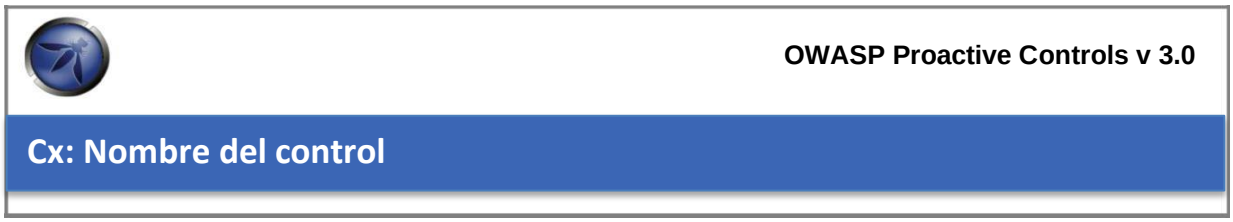
Katy Anton Jim Bird Jim Manico

CONTRIBUIDORES

Chris Romeo | Dan Anderson | David Cybuck
Dave Ferguson | Josh Grossman | Osama Elnaggar
Colin Watson | Rick Mitchell | And many more…

ESTRUCTURA DEL DOCUMENTO

Este documento está estructurado como una lista de controles de seguridad. Cada control se describe de la siguiente manera:



Descripción

Una descripción detallada del control que incluye algunas mejores prácticas a considerar.

Implementación

Mejores practices de implementación y ejemplos para ilustrar cómo implementar cada control.

Vulnerabilidades prevenidas

- _____
- _____

Lista de vulnerabilidades prevenidas o riesgos abordados (OWASP TOP 10 Risk, CWE, etc.)

Referencias

- _____
- _____

Lista de referencias para studio adicional (OWASP Cheat sheet, Security Hardening Guidelines, etc.)

Herramientas

- _____
- _____

Conjunto de herramientas/proyectos para fácilmente introducir/integrar controles de seguridad en tu software.

INTRODUCCIÓN

OWASP Diez Mejores Controles Proactivos 2018 es una lista de técnicas de seguridad que deberían considerarse para todo proyecto de desarrollo de software. Este documento está escrito para que desarrolladores asistan a principiantes en el desarrollo seguro.

Uno de los objetivos principales de este documento es proveer una guía práctica y concreta que ayude a los desarrolladores a construir software seguro. Estas técnicas deberían ser aplicadas de forma proactive en las primeras etapas de Desarrollo de software para asegurar su máxima efectividad.

Los 10 Mejores Controles Proactivos

Esta lista está ordenada por importancia, siendo el punto número 1 el más importante:

C1: Definir Requisitos de Seguridad

C2: Aprovechar Bibliotecas y Frameworks de Seguridad

C3: Securizar el Acceso a la Base de Datos

C4: Codificar y Escapar Datos

C5: Validar Todas las Entradas

C6: Implementar Identidad Digital

C7: Cumplir los Controles de Acceso

C8: Proteger los Datos en Todas Partes

C9: Implementar Registro y Monitoreo de Seguridad

C10: Manejar Todos los Errores y Excepciones

Cómo se creó esta lista

Esta lista fue creada originalmente por los actuales líderes de proyecto con contribuciones de varios voluntarios. El documento fue luego compartido globalmente para que incluso

sugerencias anónimas puedan ser consideradas. Se aceptaron cientos de cambios de este proceso comunitario abierto.

Audiencia Objetivo

Este documenta esta principalmente escrito para desarrolladores. Sin embargo, gerentes de desarrollo, dueños de producto, profesionales QA, gerentes de proyecto, y cualquiera relacionado en la construcción de software puede beneficiarse con este documento.

Como Usar Este Documento

Este documento pretende proveer concientización inicial sobre la construcción de software seguro. Este documento también proveerá una buena base de temas para impulsar entrenamiento introductorio sobre seguridad de software a desarrolladores. Estos controles deben ser usados de manera consistente sobre todas las aplicaciones. Sin embargo, este documento debe verse como un punto de partida y no como un conjunto de técnicas y prácticas exhaustivo. Un proceso de Desarrollo Seguro debe incluir requerimientos integrales de un estándar como OWASP ASVS además de incluir un agama de actividades de Desarrollo de software descriptos en modelos de madurez como OWASP SAMM y BSIMM.

Enlace al Proyecto OWASP Top 10

El OWASP Diez Mejores Controles Proactivos es similar al OWASP Top 10 pero está enfocado en técnicas defensivas y controles a diferencia de los riesgos. Cada técnica o control en este documento se asignará a uno más de los puntos del OWASP Top 10, *basado en riesgos*. Esta asignación está incluida al final de la descripción de cada control.

OWASP Proactive Controls v 3.0

C1: Definir Requisitos de Seguridad

Descripción

Un requisito de seguridad es una declaración de la funcionalidad de seguridad necesaria que garantiza que se satisfaga una de muchas propiedades de seguridad del software. Los requisitos de seguridad se derivan de estándares de la industria, leyes aplicables y un historial de vulnerabilidades pasadas. Los requisitos de seguridad definen nuevas características o adiciones a las características existentes para resolver un problema de seguridad específico o eliminar una vulnerabilidad potencial.

Los requisitos de seguridad proporcionan una base de funcionalidad de seguridad examinada para una aplicación. En lugar de crear un enfoque personalizado de seguridad para cada aplicación, los requisitos de seguridad estándar permiten a los desarrolladores reutilizar la definición de controles de seguridad y mejores prácticas. Esos mismos requisitos de seguridad examinados brindan soluciones para problemas de seguridad que ocurrieron en el pasado. Los requisitos existen para evitar la repetición de fallas de seguridad pasadas.

El OWASP ASVS

El OWASP Application Security Verification Standard (ASVS) es un catálogo de requisitos de seguridad disponibles y criterios de verificación. OWASP ASVS puede ser una fuente de requisitos detallados de seguridad para equipos de desarrollo.

Los requisitos de seguridad se clasifican en diferentes categorías en base a una función de seguridad de orden superior compartida. Por ejemplo, el ASVS contiene categorías como autenticación, control de acceso, manejo de errores / registro, y servicios web. Cada categoría contiene una colección de requisitos que representan las mejores prácticas para esa categoría redactadas como declaraciones verificables.

Aumentando Requisitos con Historias de Usuario y Casos de Uso Indebido

Los requisitos del ASVS son declaraciones verificables básicas que pueden expandirse con historias de usuario y casos de uso indebido. Las ventajas de una historia de usuario o caso de uso indebido es que ata a la aplicación exactamente a lo que el usuario o atacante hace al Sistema, contra describir que ofrece el sistema ofrece al usuario.

Aquí hay un ejemplo de cómo expandir un requisito ASVS 3.0.1. De la sección “Requisitos de verificación de autenticación” de ASVS 3.0.1, el requisito 2.19 se enfoca en contraseñas por defecto.

> 2.19 Verificar que no se utilizan contraseñas por defecto en la aplicación o cualquiera de los componentes utilizados por la misma (como "admin/password").

Este requisito contiene tanto una acción para verificar que no existen contraseñas por defecto, como también lleva la guía de que no deben usarse contraseñas por defecto dentro de la aplicación.

Una historia de usuario se enfoca en la perspectiva del usuario, administrador o atacante del sistema, y describe la funcionalidad en base a lo que el usuario quiere que el sistema haga por él. Una historia de usuario tiene la forma de “como usuario, puedo hacer x, y, y z”.

```
Como usuario, yo puedo ingresar mi usuario y contraseña para acceder a la aplicación.

Como usuario, yo puedo ingresar una contraseña larga que tiene un máximo de 1023 caracteres.
```

Cuando la historia se enfoca en el atacante y sus acciones, se refiere a ella como caso de uso indebido.

```
Como atacante, yo puedo ingresar un nombre de usuario y contraseña predeterminados para obtener acceso.
```

Esta historia contiene el mismo mensaje que el requisito tradicional de ASVS, con detalles adicionales de usuario o atacante para ayudar a hacer el requisito más comprobable.

Implementación

El uso exitoso de requisitos de seguridad implica cuatro pasos. El proceso incluye descubrir / seleccionar, documentar, implementar y luego confirmar la correcta implementación de nuevas funcionalidades de implementación dentro de una aplicación.

Descubrimiento y Selección

El proceso comienza con el descubrimiento y selección de requisitos de seguridad. En esta fase, el desarrollador comprende los requisitos de seguridad de una fuente estándar como el ASVS y elige qué requisitos incluye para una versión determinada de una aplicación. El punto del descubrimiento y selección es elegir un número manejable de requisitos de seguridad

para la presente versión o sprint y luego continuar iterando, añadiendo más funcionalidades de seguridad con el tiempo.

Investigación y Documentación

Durante la investigación y documentación, el desarrollador revisa la aplicación existente contra el nuevo conjunto de requisitos de seguridad para determinar si la aplicación los cumple o si es necesario algún desarrollo. Esta investigación culmina con la documentación de los resultados de la revisión.

Implementación y Pruebas

Una vez determinada la necesidad del desarrollo, el desarrollador debe modificar la aplicación de alguna manera para agregar la nueva funcionalidad o eliminar una opción insegura. En esta fase, el desarrollador primero determina el diseño requerido para atender el requisito y luego completa los cambios necesarios en el código. Se deben crear casos de prueba para confirmar la existencia de la nueva funcionalidad o para refutar la existencia de una opción insegura.

Vulnerabilidades Prevenidas

Los requisitos de seguridad definen la funcionalidad de seguridad de una aplicación. Una mejor seguridad incorporada desde el comienzo del ciclo de vida de una aplicación resulta en la prevención de muchos tipos de vulnerabilidades.

Referencias

- OWASP Application Security Verification Standard (ASVS)
- OWASP Mobile Application Security Verification Standard (MASVS)
- OWASP Top Ten

OWASP Proactive Controls v 3.0

C2: Aprovechar Bibliotecas y Componentes de Seguridad

Descripción

Las bibliotecas de codificación seguras y los componentes de software con seguridad embebida ayudan a los desarrolladores de software a protegerse contra fallas de implementación y diseño relacionadas con la seguridad. Un desarrollador construyendo una aplicación desde cero puede no tener suficiente conocimiento, tiempo, o presupuesto para implementar o mantener funcionalidades de seguridad apropiadamente. Aprovechar los componentes de seguridad ayuda a lograr objetivos de seguridad más eficientemente y de forma más precisa.

Buenas Prácticas de Implementación

Al incorporar bibliotecas o componentes de terceros en tu software, es importante considerar las siguientes buenas prácticas:

1. Use bibliotecas y componentes de fuentes confiables, activamente mantenidas y ampliamente utilizadas por muchas aplicaciones.
2. Cree y mantenga un catálogo de inventario de todas las bibliotecas de terceros.
3. Mantenga actualizadas de forma proactive las bibliotecas y componentes. Use una herramienta como OWASP Dependency Check y Retire.JS para identificar dependencias de proyecto y verificar si hay vulnerabilidades conocidas y divulgadas públicamente en el código de terceros.
4. Reduzca la superficie de ataque encapsulando la biblioteca y exponga sólo el comportamiento requerido en su software.

Vulnerabilidades Prevenidas

Los componentes y bibliotecas seguras pueden ayudar a prevenir un gran rango de vulnerabilidades web. Es crítico mantener esos componentes y bibliotecas actualizadas como se describe en el riesgo de Top Ten 2017, uso de componentes con vulnerabilidades conocidas.

Herramientas

- OWASP Dependency Check – identifica dependencias de proyecto y busca vulnerabilidades divulgadas públicamente
- Retire.JS escáner para bibliotecas de JavaScript

OWASP Proactive Controls v 3.0

C3: Securizar el Acceso a Base de Datos

Descripción

Esta sección describe el acceso seguro a todos los almacenamientos de datos, incluyendo bases de datos relacionales y bases de datos NoSQL. Algunas áreas que considerar:

1. Consultas seguras
2. Configuración segura
3. Autenticación segura
4. Comunicación segura

Consultas Seguras

La inyección SQL ocurre cuando una entrada no confiable de usuario es añadida dinámicamente a una consulta SQL de manera insegura, comúnmente a través de manipulación básica de cadenas. La inyección SQL es uno de los riesgos de seguridad de la aplicación más peligrosos. La inyección SQL es fácil de explotar y podría provocar el robo, borrado o modificación de toda la base de datos. La aplicación puede incluso ser usada para ejecutar comandos peligrosos contra el sistema operativo que aloja la base de datos, dando al atacante un punto de soporte en tu red.

Para mitigar la inyección SQL, las entradas que no sean de confianza no deben ser interpretadas como parte de un comando SQL. La mejor manera de hacer esto es con una técnica de programación conocida como “Parametrización de Consultas”. Esta defensa debe ser aplicada a SQL, OQL, y cualquier uso de procedimientos almacenados (stored procedures).

Una buena lista de ejemplos de parametrización de consultas en ASP, ColdFusion, C#, Delphi, .NET, Go, Java, Perl, PHP, PL/SQL, PostgreSQL, Python, R, Ruby y Scheme puede encontrarse en http://bobby-tables.com y la OWASP Cheat Sheet on Query Parameterization.

Precaución en la Parametrización de Consultas

Ciertas partes de una consulta de base de datos no son parametrizables. Estas partes son diferentes para cada proveedor de base de datos. Asegúrese de realizar con mucho cuidado validación de coincidencia exacta o un escape manual al enfrentarse parámetros de una consulta de base de datos que no puedan vincularse a una consulta parametrizada. Además, aunque el uso de consultas parametrizadas tiene en gran medida un impacto positivo en el rendimiento, ciertas consultas parametrizadas en implementaciones de bases de datos

específicas afectarán negativamente el rendimiento. Asegúrese de probar el rendimiento de las consultas; especialmente consultas complejas con extenso uso de la cláusula LIKE o que realice búsquedas de texto.

Configuración Segura

Desafortunadamente, los sistemas de administración de bases de datos no siempre se instalan con una configuración "segura por defecto". Se debe tener cuidado para garantizar que los controles de seguridad disponibles del Sistema de Administración de Base de Datos (DBMS) y de la plataforma que la aloja estén habilitados y configurados correctamente. Hay estándares, guías y referencias para los DBMS más comunes.

Autenticación Segura

Todo acceso a la base de datos debe estar debidamente autenticado. La autenticación al DBMS debe realizarse de manera segura. La autenticación debe realizarse solo a través de un canal seguro. Las credenciales deben estar debidamente securizadas y disponibles para su uso.

Comunicación Segura

La mayoría de los DBMS admiten una variedad de métodos de comunicación (servicios, API, etc.) - seguros (autenticados, encriptados) e inseguros (no autenticados o no encriptados). Es una buena práctica usar solo las opciones de comunicaciones seguras acorde al control *Proteger los datos en todas partes*.

Vulnerabilidades Prevenidas

- OWASP Top 10 2017- A1: Injection
- OWASP Mobile Top 10 2014-M1 Weak Server Side Controls

Referencias

- OWASP Cheat Sheet: Query Parameterization
- Bobby Tables: A guide to preventing SQL injection
- CIS Database Hardening Standards

OWASP Proactive Controls v 3.0

C4: Codificar y Escapar Datos

Descripción

La codificación y el escape de datos son técnicas defensivas destinadas a evitar ataques de inyección.

Codificación implica la traducción de caracteres especiales a una forma diferente pero equivalente que deje de ser peligrosa para el intérprete de destino, por ejemplo, traducir el carácter "<" al carácter "%lt;" al escribir en una página HTML.

Escape implica agregar un carácter especial antes del carácter para evitar que se malinterprete, por ejemplo, agregar un "\" antes de un "" (comilla doble) para que se interprete como texto y no como el final de una cadena.

La codificación o escape de salida es mejor aplicada justo antes de que el contenido se pase al intérprete de destino. Si esta defensa se hace demasiado temprano en el procesamiento de una petición, podría interferir con el uso del contenido en otras partes del programa. Por ejemplo, si HTML escapara el contenido antes de almacenar los datos en la base de datos y la interfaz de usuario lo codifica automáticamente una segunda vez, el contenido no se mostrará correctamente debido al doble escape.

Codificación Contextual de las Salidas

La codificación contextual de las salidas es una técnica de programación de seguridad crucial necesaria para detener el XSS. Esta defensa se realiza en la salida, cuando se está creando una interfaz de usuario, en el último momento antes de que los datos que no son de confianza se agreguen dinámicamente al HTML. El tipo de codificación dependerá de la ubicación (o contexto) en el documento donde los datos serán mostrados o almacenados. Los diferentes tipos de codificación que se utilizarían para construir interfaces de usuario seguras incluyen Codificación de Entidades HTML, Codificación de Atributos HTML, Codificación de JavaScript y Codificación de URL.

Ejemplos de Codificación en Java

Para ejemplos del codificador de OWASP en Jaca proveyendo codificación contextual de salida ver: OWASP Java Encoder Project Examples.

Ejemplos de Codificación en .NET

Comenzando con .NET 4.5, la biblioteca anti-XSS es parte del componente, pero no está habilitada por defecto. Puedes especificar el uso de AntiXssEncoder de esta biblioteca como el codificador por defecto para tu aplicación entera usando la configuración de web.conf. Cuando se aplica, es importante codificar contextualmente su salida, lo que significa usar la función correcta de la biblioteca AntiXssEncoder para la ubicación apropiada de los datos en el documento.

Ejemplos de Codificación en PHP

Zend Framework 2

En Zend Framework 2 (ZF2), Zend\Escaper puede ser utilizado para la codificación de la salida. Para ejemplos de codificación contextual ver Context-specific escaping with zend-escaper.

Otros Tipos de Defensa en Codificación e Inyección

La codificación/escape puede utilizarse para neutralizar contenido contra otras formas de inyección. Por ejemplo, es posible neutralizar ciertos meta-caracteres especiales al agregar una entrada a un comando del sistema operativo. Esto se llama “escape de comando del sistema operativo”, “escape de shell”, o similar. Esta defensa se puede utilizar para detener vulnerabilidades de “Inyección de comandos”.

Hay otras formas de escape que pueden ser usadas para detener inyección, como el escape de atributo XML que detiene varias formas de inyección XML e inyección de ruta XML, así como escape de nombre distinguido LDAP que puede usarse para detener varias formas de inyección LDAP.

Codificación y Canonicalización de Caracteres

La codificación Unicode es un método para almacenar caracteres con múltiples bytes. Dondequiera que se permitan datos de entrada, los datos pueden ingresar usando Unicode para disfrazar código malicioso y permitir una variedad de ataques. RFC 2279 hace referencia a muchas formas en que se puede codificar texto.

La canonicalización es un método en que los sistemas convierten datos a una forma simple o estándar. Las aplicaciones web comúnmente usan canonicalización de caracteres para garantizar que todo el contenido tenga el mismo tipo de carácter cuando se almacena o muestra.

Para estar Seguro contra los ataques relacionados con la canonicalización, una aplicación debe estar segura cuando se ingresan caracteres Unicode o con otras representaciones de caracteres con formato incorrecto.

Vulnerabilidades Prevenidas

- OWASP Top 10 2017 - A1: Injection
- OWASP Top 10 2017 - A7: Cross Site Scripting (XSS)
- OWASP Mobile_Top_10_2014-M7 Client Side Injection

Referencias

- XSS – Información general
- OWASP Cheat Sheet: XSS Prevention – Detener XSS en su aplicación web
- OWASP Cheat Sheet: DOM based XSS Prevention
- OWASP Cheat Sheet: Injection Prevention

Herramientas

- OWASP Java Encoder Project
- AntiXSSEncoder
- Zend\Escaper – ejemplos de codificación contextual

OWASP Proactive Controls v 3.0

C5: Validar Todas las Entradas

Descripción

La validación de entrada es una técnica de programación que garantiza que sólo los datos formateados correctamente puedan ingresar a un componente del sistema de software.

Validez Sintáctica y Semántica

Una aplicación debe verificar los datos sean válidos tanto *sintáctica* como *semánticamente* (en ese orden) antes de que sean usados de cualquier manera (incluso mostrárselo al usuario).

La ***validez sintáctica*** significa que los datos estén en la forma esperada. Por ejemplo, una aplicación puede permitir que un usuario seleccione un "ID de cuenta" de cuatro dígitos para realizar algún tipo de operación. La aplicación debe suponer que el usuario está ingresando una carga de inyección SQL y debe verificar que los datos ingresados por el usuario sean exactamente cuatro dígitos en longitud y que sean sólo dígitos (además de la utilizar parametrización de consulta adecuada).

La ***validez semántica*** significa solo aceptar entradas que estén dentro de un rango aceptable para la funcionalidad dada. Por ejemplo, una fecha de inicio debe ser anterior a una fecha de fin al elegir intervalos de fechas.

Lista de Permitidos vs Lista de Prohibidos

Existen dos enfoques generales para realizar la validación de la sintaxis de entrada, comúnmente conocidos como lista de permitidos y lista de prohibidos:

- Las *listas de prohibidos* intentan verificar que los datos dados no contengan contenido "malo conocido". Por ejemplo, una aplicación web puede bloquear entradas que contengan el texto exacto "<SCRIPT>" para ayudar a prevenir XSS. Sin embargo, esta defensa puede evadirse usando minúsculas o una mezcla de minúsculas y mayúsculas.
- Las *listas de permitidos* intentan verificar que los datos dados coincidan con contenido "bueno conocido". Por ejemplo, una validación para un Estado de EE. UU. sería que la entrada coincida con un código de dos letras de uno de los Estados de EE. UU. válidos.

Al crear software seguro, la lista de permitidos es el enfoque mínimo recomendado. La inclusión en la lista de prohibidos es propensa a errores, se puede eludir con diferentes técnicas de evasión y puede ser peligrosa si sólo se depende de ella. Aunque la lista de prohibidos a menudo se puede evadir, puede ser útil para ayudar a detectar ataques obvios. Entonces, mientras que la lista de permitidos ayuda a limitar la superficie de ataque garantizando que los datos tengan validez sintáctica y semántica, la lista de prohibidos ayuda a detectar y potencialmente detener ataques obvios.

Validación en el Cliente y Validación en el Servidor

La validación de entrada siempre debe realizarse en el lado del servidor por seguridad. Si bien la validación del lado del cliente puede ser útil tanto para fines funcionales como de seguridad, a menudo se puede evadir fácilmente. Esto hace que la validación del lado del servidor sea aún más fundamental para la seguridad. Por ejemplo, la validación de JavaScript puede alertar al usuario de que un campo en particular debe constar de números, pero la aplicación del lado del servidor debe validar que los datos enviados solo constan de números en el rango numérico apropiado para esa característica.

Expresiones Regulares

Las expresiones regulares ofrecen una forma de verificar si los datos coinciden con un patrón específico. Comencemos con un ejemplo básico.

La siguiente expresión regular se usa para definir una regla de lista de permitidos para validar nombres de usuario.

```
^[a-z0-9_]{3,16}$
```

Esta expresión regular solo permite letras minúsculas, números y el carácter de guion bajo. El nombre de usuario también está restringido a una longitud entre 3 y 16 caracteres.

Precaución: Denegación de Servicio potencial

Se debe tener cuidado al crear expresiones regulares. Las expresiones mal diseñadas pueden dar lugar a posibles condiciones de denegación de servicio (también conocido como ReDoS). Varias herramientas pueden probar para verificar que las expresiones regulares no son vulnerables a ReDoS.

Precaución: Complejidad

Las expresiones regulares son solo una forma de validar. Las expresiones regulares suelen ser difíciles de mantener o entender para algunos desarrolladores. Otras alternativas de validación implican escribir métodos de validación mediante programación, lo que puede ser más fácil de mantener para algunos desarrolladores.

Límites de la Validación de Entradas

La validación de entrada no siempre hace que los datos sean "seguros", ya que ciertas formas de entrada compleja pueden ser "válidas" pero aún peligrosas. Por ejemplo, una dirección de correo electrónico puede contener una inyección SQL o una URL válida puede contener un ataque de Cross Site Scripting. Siempre se deben aplicar defensas adicionales además de la validación de entradas como la parametrización de las consultas o el escape.

Desafíos de la Validación de Datos Serializados

Algunas formas de entrada son tan complejas que la validación puede proteger la aplicación de forma mínima. Por ejemplo, es peligroso deserializar datos que no son de confianza o datos que pueden ser manipulados por un atacante. El único patrón arquitectónico seguro es no aceptar objetos serializados de fuentes no confiables o sólo deserializar en capacidad limitada para tipos de datos simples. Debe evitar el procesar formatos de datos serializados y usar formatos más fáciles de defender, como JSON, cuando sea posible.

Si no es posible, considere una serie de defensas de validación al procesar datos serializados.

- Implementar controles de integridad o cifrado de los objetos serializados para evitar la creación de objetos hostiles o manipulación de datos.
- Aplicar restricciones estrictas durante la deserialización antes de la creación del objeto; normalmente el código espera un conjunto definido de clases. Evasiones a esta técnica han sido demostradas.
- Aísle el código que se deserializa, de modo que se ejecute en entornos de privilegios bajo, como contenedores temporales.
- Registrar excepciones y errores de deserialización, como cuando el tipo esperado o la deserialización genera excepciones.
- Restringir o monitorear la conectividad entrante y saliente desde contenedores o servidores que deserializan
- Monitorear la deserialización, alertando si un usuario deserializa constantemente.

Entradas de Usuario Inesperadas (Asignación Masiva)

Algunos componentes soportan el enlace automático de parámetros de peticiones HTTP a objetos del lado del servidor usados por la aplicación. Esta función de vinculación automática puede permitir que un atacante actualice objetos del lado del servidor que no deben modificarse. El atacante posiblemente puede modificar su control de acceso o evadir la lógica de negocio prevista de la aplicación con esta función.

Este ataque tiene un número de nombres incluidos: asignación masiva, enlace automático e inyección de objetos.

Como ejemplo simple, si el objeto usuario tiene un privilegio de campo que especifica el nivel de privilegio del usuario en la aplicación, un usuario malintencionado puede buscar páginas donde se modifiquen los datos del usuario y agregar “privilege=admin” a los parámetros HTTP enviados. Si el enlace automático está habilitado de manera no segura, el objeto del lado del servidor que representa al usuario se modificará en consecuencia.

Se pueden usar dos enfoques para manejar esto:

- Evite vincular la entrada directamente y utilice Objetos de Transferencia de Datos (DTOs) en su lugar.
- Habilite el enlace automático, pero configure reglas de lista de permitidos para cada página o función para definir qué campos se pueden enlazar automáticamente.

Más ejemplos disponibles en OWASP Mass Assignment Cheat Sheet.

Validar y Sanitizar HTML

Considere una aplicación que necesita aceptar HTML de los usuarios (a través de un editor WYSIWYG que representa el contenido como HTML o funciones que aceptan HTML directamente como entrada). En esta situación, la validación o el escape no ayudarán.

- Las expresiones regulares no son lo suficientemente expresivas para comprender la complejidad de HTML5.
- La codificación o el escape de HTML no ayudará, ya que hará que el HTML no se represente correctamente.

Por lo tanto, necesita una biblioteca que pueda analizar y limpiar texto con formato HTML. Por favor vea XSS Prevention Cheat Sheet on HTML Sanitization para más información sobre sanitización de HTML.

Funcionalidad de Validación en Bibliotecas y Componentes

Todos los lenguajes y la mayoría de los componentes proporcionan bibliotecas o funciones de validación que deben aprovecharse para validar datos. Las bibliotecas de validación generalmente cubren tipos de datos comunes, requisitos de longitud, rangos de números enteros, verificaciones "es nulo" y más. Muchas bibliotecas de validación y componentes le permiten definir su propia expresión regular o lógica para una validación personalizada de una manera que le permite al programador aprovechar esa funcionalidad en toda su aplicación. Ejemplos de funcionalidad de validación incluyen las funciones de filtro de PHP o el Hibernate Validator de Java. Ejemplos de sanitizadores de HTML incluyen el método sanitize de Ruby on Rails, OWASP Java HTML Sanitizer o DOMPurify.

Vulnerabilidades Prevenidas

- La validación de entrada reduce la superficie de ataque de las aplicaciones y puede, en ocasiones, dificultar los ataques contra una aplicación.
- La validación de entrada es una técnica que brinda seguridad a ciertas formas de datos, específica para ciertos ataques y no puede aplicarse de manera confiable como regla de seguridad general.
- La validación de entrada no debe usarse como método principal para prevenir XSS, SQL Injection y otros ataques.

Referencias

- OWASP Cheat Sheet: Input Validation
- OWASP Cheat Sheet: iOS - Security Decisions via Untrusted Inputs
- OWASP Testing Guide: Testing for Input Validation

Herramientas

- OWASP Java HTML Sanitizer Project
- Java JSR-303/JSR-349 Bean Validation
- Java Hibernate Validator
- JEP-290 Filter Incoming Serialization Data
- Apache Commons Validator
- Funciones de filtro de PHP

OWASP Proactive Controls v 3.0

C6: Implementar Identidad Digital

Descripción

La Identidad Digital es la representación única de un usuario (u otro sujeto) al participar en una transacción en línea. La autenticación es el proceso de verificar que una o entidad es quien dice ser. La administración de sesiones es un proceso mediante el cual un servidor mantiene el estado de autenticación de los usuarios para que el usuario pueda continuar usando el sistema sin volver a autenticarse. La publicación especial 800-63B del NIST: Digital Identity Guidelines (Authentication and Lifecycle Management) proporciona una guía sólida sobre la implementación de controles de gestión de sesión, autenticación e identidad digital.

A continuación, se presentan algunas recomendaciones para una implementación segura.

Niveles de Autenticación

NIST 800-63b describe tres niveles de garantía de autenticación denominados nivel de garantía de autenticación (AAL). El nivel 1 de AAL está reservado para aplicaciones de menor riesgo que no contienen PII u otros datos privados. En el nivel 1 sólo se requiere la autenticación de un solo factor, normalmente mediante el uso de contraseña.

Nivel 1: Contraseñas

Las contraseñas son realmente muy importantes. Necesitamos una política, necesitamos almacenarlas de forma segura, a veces necesitamos permitir que los usuarios las restablezcan.

Requisitos de Contraseñas

Las contraseñas deben cumplir como mínimo con los siguientes requisitos:

- tener al menos 8 caracteres de longitud si se utiliza múltiple factor de autenticación (MFA) y otros controles. Si MFA no es possible, debe aumentarse a 10 caracteres
- todos los caracteres ASCII que se pueden imprimir en pantalla, tanto como el carácter de espacio deben ser aceptables en los secretos memorizados
- fomentar el uso de contraseñas y frases de contraseñas largas
- remover requisitos de complejidad, ya que se ha determinado que estos requisitos tienen una eficacia limitada. En su lugar, se recomienda la adopción de MFA o de contraseñas más largas.

- asegurarse de que las contraseñas utilizadas no sean contraseñas de uso común que ya se hayan filtrado en un compromiso anterior. Puede elegir bloquear las 1000 o 10000 contraseñas más comunes que cumplan con los requisitos de longitud anteriores y se encuentren en listas de contraseñas comprometidas. El siguiente enlace contiene las contraseñas más comunes: https://github.com/danielmiessler/SecLists/tree/master/Passwords

Implementar un Mecanismo Seguro de Recuperación de Contraseña

Es común que una aplicación tenga un mecanismo para que un usuario obtenga acceso a su cuenta en caso de olvidar su contraseña. Un buen diseño de flujo de trabajo para una función de recuperación de contraseña utilizará elementos de MFA. Por ejemplo, puede hacer una pregunta de seguridad, algo que saben, y luego enviar un token generado a un dispositivo, algo que les pertenece.

Por favor vea Forgot_Password_Cheat_Sheet y Choosing_and_Using_Security_Questions_Cheat_Sheet para más detalles.

Implementar Almacenamiento Seguro de Contraseñas

Para proporcionar controles de autenticación sólidos, una aplicación debe almacenar de forma segura las credenciales de los usuarios. Además, se deben implementar controles criptográficos de modo que si una credencial (por ejemplo, una contraseña) se ve comprometida, el atacante no tenga acceso inmediato a esta información.

Ejemplo de PHP de Almacenamiento de Contraseñas

A continuación se muestra un ejemplo de un hash de contraseña seguro en PHP usando la función password_hash() (disponible desde 5.5.0) que utiliza por defecto el algoritmo bcrypt. El ejemplo usa un factor de trabajo de 15.

```
<?php
  $cost = 15;
  $password_hash = password_hash("secret_password", PASSWORD_DEFAULT, ["cost" =>
$cost] );
?>
```

Por favor vea OWASP Password Storage Cheat Sheet para más detalles.

Nivel 2: Autenticación de Factor Múltiple

El nivel 2 de AAL de NIST 800-63b está reservado para aplicaciones de mayor riesgo que contienen "PII auto firmada u otra información personal disponible en línea." En el nivel 2 de AAL se requiere autenticación de factor múltiple, incluyendo OTP u otra forma de implementación de factor múltiple.

La autenticación de factor múltiple (MFA) garantiza que los usuarios sean quienes dicen ser al exigirles que se identifiquen con una combinación de:

- Algo que sepan - contraseña o PIN
- Algo que posean - token o teléfono
- Algo que sean - datos biométricos, como una huella dactilar

El uso de contraseñas como único factor proporciona una seguridad débil. Las soluciones de factor múltiple proporcionan una solución más robusta al requerir que un atacante adquiera más de un elemento para autenticarse con el servicio.

Vale la pena señalar que los datos biométricos, cuando se emplea como factor único de autenticación, no se considera un secreto aceptable para la autenticación digital. Se pueden obtener en línea o tomando una foto de alguien con una cámara de teléfono (por ejemplo, imágenes faciales) con o sin su conocimiento, extrayéndolas de objetos que alguien toque (por ejemplo, huellas dactilares latentes) o capturadas con imágenes de alta resolución (por ejemplo, patrones de iris). Los datos biométricos deben usarse solo como partes de la autenticación de factores múltiples con un autenticador físico (algo que posean). Por ejemplo, acceder a un dispositivo de contraseña de un solo uso (OTP) que generará una contraseña de un solo uso que el usuario ingresa manualmente para el verificador.

Nivel 3: Autenticación Basada en Criptografía

El NIST 800-63b garantía de autenticación nivel 3 (AAL3) se requiere cuando el impacto de los sistemas comprometidos podría provocar daños personales, pérdidas financieras significativas, dañar el interés público o involucrar violaciones civiles o penales. AAL 3 requiere una autenticación "basada en prueba de posesión de una clave a través de protocolo criptográfico." Este tipo de autenticación se usa para lograr el nivel más alto de garantía de autenticación. Esto normalmente se hace a través de módulos criptográficos de hardware.

Gestión de Sesiones

Una vez que se ha llevado a cabo la autenticación inicial exitosa del usuario, una aplicación puede optar por rastrear y mantener este estado de autenticación durante un período de tiempo limitado. Esto permitirá que el usuario continue usando la aplicación sin tener que

volver a autenticarse con cada solicitud. El seguimiento de este estado de usuario se denomina Gestión de Sesión.

Generación y Expiración de Sesiones

El estado de usuario se rastrea en una sesión. Esta sesión generalmente se almacena en el servidor para la administración tradicional de sesiones basada en la web. Luego le se proporciona un identificador de sesión al usuario para que pueda identificar qué sesión del lado del servidor contiene los datos de usuario correctos. El cliente solo necesita mantener este identificador de sesión, que también mantiene los datos confidenciales de la sesión del lado del servidor fuera del cliente.

Estos son algunos controles que se deben tener en cuenta al crear o implementar soluciones de administración de sesiones:

- Asegúrese que el ID de la sesión sea larga, única y aleatoria.
- La aplicación debe generar una nueva sesión o al menos rotar el ID de sesión durante la autenticación y reautenticación.
- La aplicación debe implementar un tiempo de inactividad después de un período de inactividad y una vida útil máxima absoluta para cada sesión, luego de lo cual los usuarios deben volver a autenticarse. La duración de los tiempos de espera debe ser inversamente proporcional al valor de los datos protegidos.

Por favor vea Session Management Cheat Sheet para más detalles. ASVS Sección 3 cubre requisitos adicionales de administración de sesiones.

Cookies del Navegador

Las cookies del navegador son un método común para que las aplicaciones web almacenen identificadores de sesión para aplicaciones que implementan técnicas estándar de administración de sesiones. Aquí algunas defensas a tener en cuenta al usar cookies del navegador.

- Cuando las cookies del navegador se utilizan como mecanismo para rastrear la sesión de un usuario autenticado, deben ser accesibles para un conjunto mínimo de dominios y rutas y deben etiquetarse para que caduquen junto al período de validez de la sesión o poco después.
- El indicador 'secure' debe establecerse para garantizar que la transferencia se realice solo a través de un canal seguro (TLS).

- El indicador HttpOnly debe ser configurado para evitar que se acceda a la cookie a través de JavaScript.
- Agregar el atributo "samesite" a las cookies evita que algunos navegadores modernos envíen cookies con solicitudes a otros sitios y proporciona protección contra la falsificación de solicitudes entre sitios y ataques de fuga de información.

Tokens

Las sesiones del lado del servidor pueden ser limitantes para algunas formas de autenticación. Los "servicios stateless (sin estado)" permiten la administración del lado del cliente de los datos de la sesión con fines de rendimiento, por lo que el servidor tiene menos carga para almacenar y verificar la sesión del usuario. Estas aplicaciones "stateless" generan un token de acceso de corta duración que se puede utilizar para autenticar la solicitud de un cliente sin enviar las credenciales del usuario después de la autenticación inicial.

JWT (JSON Web Tokens)

JSON Web Token (JWT) es un estándar abierto (RFC 7519) que define una forma compacta y autónoma de transmitir información de forma segura entre las partes como un objeto JSON. Esta información se puede verificar y confiar porque está firmada digitalmente. Se crea un token JWT durante la autenticación y el servidor (o servidores) lo verifica antes de cualquier procesamiento.

Sin embargo, los JWT a menudo no son guardados por el servidor después de la creación inicial. Los JWT generalmente se crean y luego se entregan a un cliente sin que el servidor los guarde de ninguna manera. La integridad del token se mantiene mediante el uso de firmas digitales para que un servidor pueda verificar más tarde que el JWT sigue siendo válido y no fue manipulado desde su creación.

Este enfoque no tiene estado y es portable en el sentido en que las tecnologías de cliente y servidor pueden ser diferentes y aun así interactuar.

Precaución

La identidad digital, la autenticación y la gestión son temas muy importantes. Apenas estamos rascando la superficie del tema de Identidad Digital. Asegúrese que su talento de ingeniería más capaz es el responsable para mantener la complejidad que implica la mayoría de las soluciones de Identidad.

Vulnerabilidades Prevenidas

- OWASP Top 10 2017 A2- Broken Authentication and Session Management
- OWASP Mobile Top 10 2014-M5- Poor Authorization and Authentication

Referencias

- OWASP Cheat Sheet: Authentication
- OWASP Cheat Sheet: Password Storage
- OWASP Cheat Sheet: Forgot Password
- OWASP Cheat Sheet: Choosing and Using Security Questions
- OWASP Cheat Sheet: Session Management
- OWASP Cheat Sheet: IOS Developer
- OWASP Testing Guide: Testing for Authentication
- NIST Special Publication 800-63 Revision 3 - Digital Identity Guidelines

Herramientas

- Daniel Miessler: Most commonly found passwords

OWASP Proactive Controls v 3.0

C7: Cumplir los Controles de Acceso

Descripción

El Control de Acceso (o Autorización) es el proceso de otorgar o denegar *peticiones específicas* de un usuario, programa o proceso. El acceso de control también implica el acto de *otorgar y revocar esos privilegios*.

Cabe señalar que la autorización (verificación del acceso a funciones o recursos específicos) no es equivalente a la autenticación (verificación de la identidad).

La funcionalidad de Control de Acceso a menudo abarca muchas áreas de software dependiendo de la complejidad del sistema de control de acceso. Por ejemplo, la gestión de metadatos de control de acceso o la creación de almacenamiento caché con fines de escalabilidad suelen ser componentes adicionales en un sistema de control de acceso que deben crearse o gestionarse.

Hay varios tipos diferentes de diseño de control de acceso que se deben considerar.

- El Control de Acceso Discrecional (DAC) es un medio para restringir el acceso a objetos (p.e. archivos, entidades de datos) en función de la identidad y la necesidad de saber de los sujetos (p.e. usuarios, procesos) y/o grupos a los que pertenece el objeto
- El Control de Acceso Obligatorio (MAC) es un medio para restringir el acceso a los recursos del sistema en función de la sensibilidad (representada por una etiqueta) de la información contenida en el recurso del sistema y la autorización formal (es decir, un permiso) de los usuarios para acceder a la información de tal sensibilidad.
- El Control de Acceso Basado en Roles (RBAC) es un modelo para controlar el acceso a los recursos donde las acciones permitidas en los recursos se identifican con roles en lugar de identidades de sujetos individuales.
- El Control de Acceso Basado en Atributos (ABAC) otorgará o rechazará las solicitudes de los usuarios en función de los atributos arbitrarios del usuario y los atributos arbitrarios del objeto, y las condiciones del ambiente que podrían reconocerse globalmente y ser más relevantes para las políticas en cuestión.

Principios de Diseño de Control de Acceso

Los siguientes requisitos de diseño de control de acceso "positivos" deben ser considerados en las etapas iniciales del desarrollo de la aplicación.

1) Diseñe el Control de Acceso Completo desde el Principio

Una vez que haya elegido un patrón de diseño de control de acceso específico, a menudo es difícil y lleva mucho tiempo rediseñar el control de acceso en su aplicación con un nuevo patrón. El Control de Accesos es una de las principales áreas de diseño de seguridad de aplicaciones que debe ser diseñarse minuciosamente desde el principio, especialmente cuando se abordan requisitos como tenencia múltiple y control de acceso horizontal (dependiente de datos).

El diseño de Acceso de Control puede comenzar de manera simple, pero a menudo crece hacia un control de seguridad complejo y con muchas funcionalidades. Al evaluar la capacidad de control de acceso de componentes de software, asegúrese de que su funcionalidad de control de acceso permita la personalización para su necesidad específica de funciones de control de acceso.

2) Fuerce a Todas las Peticiones a que Pasen por Chequeos de Control de Acceso

Asegúrese de que todas las peticiones pasen por algún tipo de capa de verificación de control de acceso. Las tecnologías como los filtros de Java u otros mecanismos de procesamiento automático de peticiones son artefactos de programación ideales que ayudarán a garantizar que todas las peticiones pasen por algún tipo de verificación de control de acceso.

3) Negar por Defecto

Denegar por defecto es el principio de que, si una petición no está específicamente permitida, se deniega. Hay muchas formas en que esta regla se manifestará en el código de la aplicación. Algunos ejemplos son:

1. El código de aplicación puede arrojar un error o excepción al procesar peticiones de control de acceso. En estos casos, siempre se debe denegar el control de acceso.
2. Cuando un administrador crea un nuevo usuario o un usuario se registra para una nueva cuenta, esa cuenta debe tener un acceso mínimo o nulo por defecto hasta que se configure ese acceso.

3. Cuando se agrega una nueva función a una aplicación, se debe denegar su uso hasta que esté configurado correctamente

4) Principio de Privilegio Mínimo

Asegúrese de que todos los usuarios, programas o procesos tienen el mínimo acceso necesario posible. Tenga cuidado con los sistemas que no brindan capacidades de configuración de control de acceso granular.

5) No Codificar roles

Muchos componentes de aplicaciones utilizan el control de acceso basado en roles por defecto. Es común encontrar código de aplicación que está lleno de chequeos de esta naturaleza.

```
if (user.hasRole("ADMIN")) || (user.hasRole("MANAGER")) {
      deleteAccount();
}
```

Tenga cuidado con este tipo de programación basada en roles en el código. Tiene las siguientes limitaciones o peligros.

- La programación basada en roles de esta naturaleza es frágil. Es fácil crear comprobaciones de roles incorrectas o faltantes en el código.
- La programación basada en roles no permite tenencia múltiple. Se requerirán medidas extremas como chequeos adicionales para cada cliente o diferentes versiones del código para permitir que los sistemas basados en roles tengan diferentes reglas para diferentes clientes.
- La programación basada en roles no permite reglas de control de acceso horizontales o específicas a datos.
- Bases de código grandes con muchas verificaciones de control de acceso pueden ser difíciles de auditar o verificar la política general de control de acceso de la aplicación.

En su lugar, considere la siguiente metodología de programación de control de acceso:

```
if (user.hasAccess("DELETE_ACCOUNT")) {
      deleteAccount();
}
```

Las comprobaciones de control de acceso basadas en características o atributos de esta naturaleza son el punto de partida para crear sistemas de control de acceso bien diseñados y ricos en funciones. Este tipo de programación también permite una mayor capacidad de personalización del control de acceso a lo largo del tiempo.

6) Registra Todos los Eventos de Control de Acceso

Todas las fallas de control de acceso deben registrarse, ya que pueden ser indicativas de un usuario malintencionado que investiga la aplicación en busca de vulnerabilidades.

Vulnerabilidades Prevenidas

- OWASP Top 10 2017-A5-Broken Access Control
- OWASP Mobile Top 10 2014-M5 Poor Authorization and Authentication

References

- OWASP Cheat Sheet: Access Control
- OWASP Cheat Sheet: iOS Developer - Poor Authorization and Authentication
- OWASP Testing Guide: Testing for Authorization

Tools

- OWASP ZAP junto al add-on opcional Access Control Testing

OWASP Proactive Controls v 3.0

C8: Proteger los Datos en Todas Partes

Descripción

Los datos sensibles, como las contraseñas, números de tarjeta de crédito, registros de salud, información personal y secretos de negocio, requieren protección extra, particularmente si esos datos se encuentran bajo leyes de privacidad (el Reglamento General de Protección de Datos General o GDPR de la UE), reglas de protección de datos financieros como el Estándar de Seguridad de Datos PCI (PCI DSS) u otras regulaciones.

Los atacantes pueden robar datos de aplicaciones web y servicios web de varias formas. Por ejemplo, si se envía información confidencial a través de internet sin seguridad den las comunicaciones, un atacante en una conexión inalámbrica compartida podría ver y robar los datos de otro usuario. Además, un atacante podría usar SQL Injection para robar contraseñas y otras credenciales de una base de datos de una aplicación y exponer esa información al público.

Clasificación de Datos

Es fundamental clasificar los datos en su sistema y determinar qué nivel de sensibilidad asignar a cada dato. Luego, a cada categoría de datos se le son aplicadas a las reglas de protección necesarias para cada nivel de sensibilidad. Por ejemplo, la información pública de marketing, que no es confidencial, puede clasificarse como información pública, por lo que puede se puede colocar en el sitio web público. Los números de tarjetas de crédito pueden clasificarse como datos de usuario privados que pueden necesitar ser encriptados mientras se almacenan o están en tránsito.

Encriptar Datos en Tránsito

Al transmitir datos sensibles a través de cualquier red, la seguridad en la comunicación extremo-a-extremo (o encriptación en tránsito) de algún tipo debe considerarse. TLS es, con mucho, el protocolo criptográfico más común y ampliamente compatible para la seguridad de comunicaciones. Es usado por muchos tipos de aplicaciones (web, servicios web, móviles) para comunicarse a través de una red de forma segura. TLS debe ser correctamente configurado en una variedad de formas para defender adecuadamente las comunicaciones.

El principal beneficio de TLS es la protección de los datos de la aplicación web frente a la divulgación y modificación no autorizada cuando se transmite entre los clientes (navegadores

web) y el servidor de aplicación web, y entre el servidor de aplicación web y el back-end u otros componentes de negocio que no se basan en el navegador.

Encriptar Datos en Almacenamiento

La primera regla de la gestión de datos confidenciales es evitar almacenar datos confidenciales cuando sea posible. Si debe almacenar datos confidenciales, asegúrese de que estén protegidos criptográficamente de alguna manera para evitar la divulgación y modificación no autorizadas.

La criptografía es uno de los temas más avanzados de la seguridad de la información, y uno cuya comprensión requiere la mayor educación y experiencia. Es difícil de hacer bien porque existen muchos enfoques para el cifrado, cada uno con ventajas y desventajas que los arquitectos y desarrolladores de soluciones web deben comprender a fondo. Además, la investigación seria en criptografía generalmente se basa en matemática avanzada y teoría de números, lo que proporciona una seria barrera de entrada.

En lugar de desarrollar la capacidad criptográfica desde cero, se recomienda fuertemente que se utilicen soluciones abiertas y revisadas por pares, como el proyecto Google Tink, Libsodium y la capacidad de almacenamiento seguro integrada en muchos componentes de software y servicios en la nube.

Aplicación Móvil: Almacenamiento Local Seguro

Las aplicaciones móviles corren un riesgo particular de fuga de datos porque los dispositivos móviles se pierden o son robados regularmente, pero contienen datos confidenciales.

Como regla general, solo los datos mínimos requeridos deben almacenarse en el dispositivo móvil. Pero si debe almacenar datos confidenciales en un dispositivo móvil, entonces los datos confidenciales deben almacenarse dentro de cada directorio de almacenamiento de datos específico de los sistemas operativos móviles. En Android, este será el keystore de Android y en iOS, será el keychain de iOS.

Ciclo de Vida de las Claves

Las claves secretas se utilizan en un número de funciones sensibles de las aplicaciones. Por ejemplo, las claves secretas se pueden usar para firmar JWT, proteger tarjetas de crédito, proporcionar varias formas de autenticación y facilitar otras funciones de seguridad confidenciales. En la gestión de claves, se deben seguir una serie de reglas que incluyen:

- Asegúrese de las claves secretas estén protegidas contra el acceso no autorizado
- Almacene las claves una bóveda de secretos adecuada, como se describe a continuación
- Utilice claves independientes cuando se requieran múltiples claves

- Construya soporte para cambiar algoritmos y claves cuando sea necesario
- Construya funcionalidad en la aplicación para manejar una rotación de claves

Manejo de Secretos de Aplicación

Las aplicaciones contienen numerosos "secretos" que son necesarios para las operaciones de seguridad. Estos incluyen certificados, contraseñas de conexión SQL, credenciales de cuentas a servicios de terceros, contraseñas, claves SSH, claves de cifrado y más. La divulgación o modificación no autorizada de estos secretos podría comprometer al sistema completo. Al administrar los secretos de aplicaciones, tenga en cuenta lo siguiente.

- No almacene los secretos en código, archivos de configuración o variables de entorno. Use herramientas como GitRob o TruffleHog para escanear repositorios buscando secretos.
- Mantenga las claves y otros secretos de nivel de aplicación en una bóveda de secretos como KeyWhiz, Vault project de Hashicorp o Amazon KMS para proveer almacenamiento seguro y acceso a los secretos de aplicación en tiempo de ejecución.

Vulnerabilidades Prevenidas

- OWASP Top 10 2017 - A3: Sensitive Data Exposure
- OWASP Mobile Top 10 2014-M2 Insecure Data Storage

Referencias

- OWASP Cheat Sheet: Transport Layer Protection
- Ivan Ristic: SSL/TLS Deployment Best Practices
- OWASP Cheat Sheet: HSTS
- OWASP Cheat Sheet: Cryptographic Storage
- OWASP Cheat Sheet: Password Storage
- OWASP Cheat Sheet: IOS Developer - Insecure Data Storage
- OWASP Testing Guide: Testing for TLS

Herramientas

- SSLyze – herramienta CLI y biblioteca de escaneo de configuración SSL
- SSLLabs – Servicio gratuito de escaneo y chequeo de configuración TLS/SSL
- OWASP O-Saft TLS Tool – Herramienta de testeo de conexión TLS
- GitRob – Herramienta CLI para encontrar información sensible en GitHub
- TruffleHog - Busca secretos accidentalmente agregados en commits
- KeyWhiz – Administrador de secretos
- Hashicorp Vault – Administrador de secretos
- Amazon KMS – Administra claves en Amazon AWS

OWASP Proactive Controls v 3.0

C9: Implementar Registro y Monitoreo de Seguridad

Descripción

El registro es un concepto que la mayoría de los desarrolladores ya utilizan con fines de depuración y diagnóstico. El registro de seguridad es un concepto igualmente básico: registrar información de seguridad durante el tiempo de ejecución de una aplicación. El monitoreo es la revisión en vivo de los registros de aplicación y de seguridad utilizando varias formas de automatización. Las mismas herramientas y patrones se pueden utilizar con fines de depuración, seguridad y operaciones.

Beneficios del Registro de Seguridad

El registro de seguridad puede usarse para:

1) Alimentar sistemas de detección de intrusos
2) Investigaciones y análisis forenses
3) Satisfacer requisitos de cumplimiento normativo

Implementación del Registro de Seguridad

La siguiente es una lista de mejores prácticas de implementación de registros de seguridad.

- Sigue un formato y enfoque de registro común dentro del sistema y a través de todos los sistemas de la aplicación. Un ejemplo de un componente común de registro es Apache Logging Services, que ayuda a proporcionar coherencia de registro entre las aplicaciones Java, PHP, .NET y C++.
- No registre demasiado ni muy poco. Por ejemplo, asegúrese de registrar siempre la marca de tiempo y la información de identificación, incluida IP de origen y la identificación del usuario, pero tenga cuidad de no registrar datos privados o confidenciales.
- Preste mucha atención a la sincronización de tiempo entre nodos para asegurarse de que las marcas de tiempo sean consistentes.

Registrar Para Detección y Respuesta de Intrusión

Use el registro para identificar actividad que indique que un usuario se está comportando de manera maliciosa. La actividad potencialmente maliciosa por registrar incluye:

- Datos enviados que están fuera de un rango esperado.
- Datos enviados que implican cambios en los datos que no deberían ser modificables (lista de selección, casillas de verificación u otros componentes de entrada limitada).
- Peticiones que violan las reglas de control de acceso del lado del servidor.
- Una lista más completa de posibles puntos de detección está disponible aquí.

Cuando su aplicación encuentra dicha actividad, su aplicación debe, al menos, registrar la actividad y marcarla como un problema de alta gravedad. Idealmente, su aplicación también debería responder a un posible ataque identificado, por ejemplo, invalidando la sesión del usuario y bloqueando su cuenta. Los mecanismos de respuesta permiten que el software reaccione en tiempo real ante posibles ataques identificados.

Diseño Seguro de Registro

Las soluciones de registro deben construirse y administrarse de manera segura. El diseño de registro seguro puede incluir los siguientes puntos:

- Codifique y valide cualquier carácter peligroso antes de registrar eventos para prevenir ataques de inyección de registro.
- No registre información confidencial. Por ejemplo, no registre contraseñas, ID de sesión, tarjetas de crédito, o números de seguridad social.
- Proteja la integridad del registro. Un atacante puede intentar alterar los registros. Por lo tanto, se debe considerar el permiso de los archivos de registro y la auditoría de los cambios de registro.
- Reenvíe registros desde sistemas distribuidos a un servicio de registro seguro y central. Esto asegurará que los datos de registro no se pierdan si un nodo se ve comprometido. Esto también permite el monitoreo centralizado.

Referencias

- OWASP AppSensor Detection Points – Puntos de detección usados para identificar un usuario malicioso que está buscando vulnerabilidades o debilidades en la aplicación.
- OWASP Log injection
- OWASP Log forging
- OWASP Cheat Sheet: Logging Cómo implementar apropiadamente el registro en una aplicación
- OWASP Development Guide: Logging
- OWASP Code Review Guide: Reviewing Code for Logging Issues

Herramientas

- OWASP Security Logging Project
- Apache Logging Services

OWASP Proactive Controls v 3.0

C10: Manejar Todos los Errores y Excepciones

Descripción

El manejo de excepciones es un concepto de programación que permite que una aplicación responda a diferentes estados de error (como una red inactiva o una falla en la conexión a la base de datos, etc.) en varias maneras. El manejo correcto de excepciones y errores es fundamental para que su código sea confiable y seguro.

El manejo de errores y excepciones ocurre en todas las áreas de una aplicación, incluida la lógica comercial crítica, así como las características de seguridad y el código de componentes.

El manejo de errores también es importante desde la perspectiva de detección de intrusiones. Ciertos ataques contra su aplicación pueden desencadenar en errores que pueden ayudar a detectar ataques en curso.

Errores en el Manejo de Errores

Investigadores en la Universidad de Toronto han descubierto que incluso los errores más pequeños en el manejo de errores o el olvido de manejar los errores pueden provocar fallas catastróficas en sistemas distribuidos.

Errores en el manejo de errores pueden generar diferentes tipos de vulnerabilidades de seguridad:

- Fuga de información: la fuga de información confidencial en los mensajes de error puede ayudar a los atacantes sin querer. Por ejemplo, un error que devuelve un seguimiento de la pila u otros detalles de errores internos puede proporcionar a un atacante información sobre su entorno. Incluso pequeñas diferencias en el manejo de diferentes condiciones de error (p.e., devolver “usuario no válido” o “contraseña no válida” en los errores de autenticación) pueden proporcionar pistas valiosas a los atacantes. Como se describió anteriormente, asegúrese de registrar los detalles del error con fines forenses y de depuración, pero no exponga esa información, especialmente a un cliente externo.
- Evasión de TLS: El ”error de falla” de goto de Apple fue un error de flujo de control en el código de manejo de errores que condujo a un compromiso total de las conexiones TLS en los sistemas Apple.

- DOS: La falta de manejo básico de errores puede provocar el apagado del sistema. Esta suele ser una vulnerabilidad bastante fácil de explotar para los atacantes. Otros problemas de manejo de errores podrían conducir a un mayor uso de CPU o el disco de formas que podrían degradar el sistema.

Consejo Positivo

- Gestione las excepciones de forma centralizada para evitar la duplicación de bloques try/catch en el código. Asegúrese de que todo comportamiento inesperado se maneje correctamente dentro de la petición.
- Asegúrese de que los mensajes de error que se muestran a los usuarios no filtren datos críticos, pero aún así sean suficientemente detallados para permitirla respuesta adecuada del usuario.
- Asegúrese que las excepciones se registren de manera que brinden suficiente información para que soporte, QA, análisis forense, o respuesta a incidentes puedan comprender el problema.
- Pruebe y verifique cuidadosamente el código de manejo de errores.

Referencias

- OWASP Code Review Guide: Error Handling
- OWASP Testing Guide: Testing for Error Handling
- OWASP Improper Error Handling
- CWE 209: Information Exposure Through an Error Message
- CWE 391: Unchecked Error Condition

Herramientas

- Error Prone – Una herramienta de análisis estático de Google para detectar errores comunes en el manejo de errores para desarrolladores Java
- Una de las herramientas automatizadas más famosas para encontrar errores en tiempo de ejecución es Chaos Monkey de Netflix, que deshabilita intencionalmente las instancias del sistema para garantizar que el servicio general se recupere correctamente.

Últimas Palabras

Este documento debe verse como un punto de partida y no como un conjunto de técnicas y prácticas exhaustivo. Queremos enfatizar nuevamente que este documento intenta proveer conocimiento inicial sobre la creación de software seguro.

Próximos pasos principales para ayudar a construir un programa de seguridad de aplicación incluyen:

1) Para entender algunos de los riesgos de seguridad en aplicaciones web, por favor revise el OWASP Top Ten y el OWASP Mobile Top Ten.
2) Según el Control Proactivo #1, un programa de desarrollo seguro debe incluir una *exhaustiva lista de requerimientos de seguridad* basada en un estándar como por ejemplo el OWASP (Web) ASVS o el OWASP (Mobile) MASVS.
3) Para entender los bloques de construcción principales de un programa de software seguro desde un punto de vista macro, por favor revise el OWASP OpenSAMM project.

Si tienes cualquier pregunta para el equipo de liderazgo del Proyecto, por favor regístrese en nuestra lista de mails en https://lists.owasp.org/mailman/listinfo/owasp_proactive_controls.

OWASP
Open Web Application
Security Project

FOR DEVELOPERS