

Unidad N° 2: Amenazas a la Seguridad

Daño: Definiremos como daño el perjuicio que se produce cuando un sistema informático falla. Dicho perjuicio debe de ser cuantificable.

Riesgo: Definiremos el riesgo como el producto entre la magnitud de un daño, y la probabilidad de que este tenga lugar.

Amenaza: Entendemos por amenaza aquella situación de daño cuyo riesgo de producirse es significativo.

Vulnerabilidad: Una vulnerabilidad es una deficiencia en un sistema susceptible de producir un fallo en el mismo.

Exploit: Llamaremos exploit a cualquier técnica que permita aprovechar una vulnerabilidad de un sistema para producir un daño en el mismo.

Tipos de Vulnerabilidades en seguridad informática

1. **Debidas a la implementación:** Son errores que ocurren en la fase de codificación o programación de un sistema, aplicación o servicio. Se deben a fallos en la forma en que fue implementado el código, las funciones, o cómo interactúan los componentes.

Ejemplos:

SQL Injection (SQLi): Se produce por no validar correctamente las entradas de datos. Cross-Site Scripting (XSS): Al no filtrar bien el contenido introducido por el usuario. Desbordamiento de búfer (Buffer Overflow): Al no controlar el tamaño de entrada que espera una variable.

Resumen: El diseño puede haber sido correcto, pero la implementación es insegura por errores humanos o de programación.

2. **Debidas al diseño:** Son fallas que aparecen en la fase de planificación o arquitectura del sistema, es decir, antes de que se programe una sola línea de código. Suceden cuando el sistema está mal diseñado desde su concepción, exponiendo debilidades estructurales.

Ejemplos:

Un sistema que permite contraseñas sin requerir un mínimo de complejidad.
Una aplicación que confía demasiado en que el cliente valide los datos, en lugar del servidor. Diseñar una red sin segmentación de zonas críticas.

Resumen: Aunque el código sea correcto, el diseño es inseguro y tiene fallos de base.

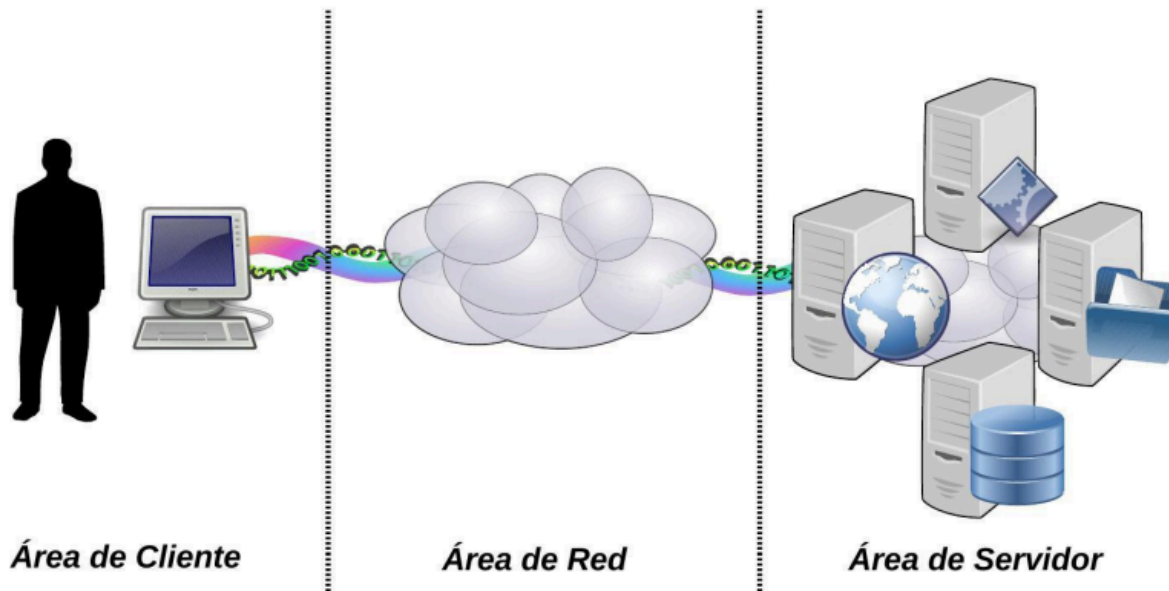
3. Debidas al uso: Surgen por cómo es utilizado el sistema por parte de usuarios o administradores, o por malas configuraciones y prácticas inseguras. No es culpa del diseño ni de la implementación, sino de la operación o mantenimiento posterior.

Ejemplos:

Un administrador deja activada una cuenta por defecto con contraseña conocida. Un usuario usa contraseñas débiles o repetidas. No aplicar actualizaciones de seguridad a tiempo.

Resumen: El sistema puede estar bien diseñado y programado, pero si se usa mal, sigue siendo vulnerable.

Áreas de vulnerabilidades en entornos Web



La nube

Ventajas:

- Facilidad de acceso a la información desde diferentes ubicaciones y dispositivos.
- Infraestructura flexible y escalable basada en servicios.
- Reducción de costos por infraestructura y servicios.
- Centralización de administración y gestión de datos.

Desventajas:

- Desventajas Dependencia en la infraestructura y servicios de terceros.
- Dependencia en servicios de comunicación externos para acceder a los sistemas y datos propios.
- Pérdida del control de la seguridad de la información (delegada al proveedor).

Ataques a aplicaciones conocidas

CVE(Common Vulnerabilities and Exposures).(Vulnerabilidades y amenaza común) :Es un código asignado a una vulnerabilidad que le permite ser identificada de forma unívoca. Este código fue creado por MITRE Corporation y permite a un usuario conocer de una forma más objetiva una vulnerabilidad en un programa o sistema. El código identificador es del modo CVE - año - número.

CWE (Common Weakness Enumeration), (Enumeración de debilidades comunes):Es una lista de tipos de debilidades de software y hardware, las mismas se encuentran clasificadas y con identificadores del tipo CWENúmero.

La NVD(National Vulnerability Database): Del NIST es el repositorio del gobierno de Estados Unidos para la gestión de datos de vulnerabilidades basados en los estándares.

CVSS(Common Vulnerability Scoring System): Es un conjunto de estándares abiertos para asignar un valor o puntaje de severidad a una vulnerabilidad. Este puntaje va desde 0.0 a 10.0, siendo este último el de mayor severidad.

Prevención de vulnerabilidades

- Listas bugtraq: Es una lista pública donde se publican vulnerabilidades y soluciones para ayudar a prevenirlas. Sirve para prevención de vulnerabilidades, detectar ataques y conocer sus métodos, proteger los sistemas reales distrayendo a los atacantes. Por ejemplo, <http://seclists.org/bugtraq/>
- DAST (Dynamic Application Security Testing): Sistemas automáticos de análisis Scanners de vulnerabilidades, por ejemplo: Vega, OpenVAS, Uniscan.
- SAST (Static Application Security Testing): Auditoria automática de código, por ejemplo: PMD, SonarQube, bandit. Lista de herramientas de analisis estatico.
- IAST (Interactive Application Security Testing): Detectan vulnerabilidades tiempo real durante la ejecución de una aplicación, por ejemplo: Contrast Assess, ContrastSecurity, Hdiv security.
- Redes Trampa o honeypot: Es un sistema falso usado como trampa para detectar o estudiar ataques reales. Sirve para detectar ataques y conocer sus métodos, identificar vulnerabilidades que los atacantes están buscando, proteger los sistemas reales distrayendo a los atacantes. Por ejemplo: <http://www.honeynet.org>

CERT / CSIRT

CERT(Computer Emergency Response Team): Es un equipo técnico especializado en responder a emergencias informáticas, especialmente a incidentes de seguridad a gran escala, como ataques masivos, virus nuevos o vulnerabilidades críticas. Es más amplio: puede coordinar incidentes que afectan a todo un país o sector, y tiene alto reconocimiento internacional.

CSIRT (Centro de Respuesta a Incidentes de Seguridad Informática):

Son equipos reconocidos por la dirección de su organización como responsables de gestionar los incidentes de seguridad informática que le competen según su alcance y comunidad. Estos grupos interactúan entre si a fin de facilitar información oportuna para actuar frente a diferentes tipos de incidentes,

determinar su impacto, alcance y naturaleza, comprender las causas, investigar soluciones, coordinar y dar apoyo para la implementación de las estrategias de respuesta con las partes involucradas, difundir información sobre los tipos de incidentes más frecuentes y mitigación de sus efectos, coordinar y colaborar con otros actores, tales como proveedores de Internet (ISP), otros grupos de seguridad, etc.









Funciones:

- Ayudar al público objetivo a atenuar y prevenir incidentes graves de seguridad.
- Ayudar a proteger informaciones valiosas.
- Coordinar de forma centralizada la seguridad de la información.
- Guardar evidencias, por si hubiera que recurrir a pleitos.
- Apoyar y prestar asistencia a usuarios para recuperarse de las consecuencias de los incidentes de seguridad.
- Dirigir de forma centralizada la respuesta a los incidentes de seguridad - Promover confianza, que alguien controla la situación.

Denegación de servicio (DoS)

En seguridad informática, un ataque de denegación de servicio, también llamado ataque DoS (de las siglas en inglés Denial of Service), es un ataque a un sistema de computadoras o red que causa que un servicio o recurso sea inaccesible a los usuarios legítimos. Normalmente provoca la pérdida de la conectividad de la red por el consumo del ancho de banda de la red de la víctima o sobrecarga de los recursos computacionales del sistema de la

víctima.

Característica	DoS (Denial of Service)	XSS (Cross-Site Scripting) 
 Objetivo	Saturar un servicio para que deje de funcionar	Inyectar scripts maliciosos en una página web
 Impacto	Inaccesibilidad del sitio o aplicación	Robo de datos, manipulación del contenido web
 A quién afecta	Al servidor o servicio web	A los usuarios que visitan la web
 Cómo se ejecuta	Enviando miles de peticiones hasta saturar recursos	Inyectando código JavaScript en formularios, URLs o comentarios
 Lenguaje involucrado	Puede usar scripts automáticos, herramientas de red	Usa principalmente JavaScript
 Prevención	Balanceadores, firewalls, limitación de tráfico	Validación y sanitización de entradas (input)
 Ejemplo de ataque	Script que envía 100.000 peticiones al servidor	<code><script>document.cookie</script></code> en un campo de comentario

Tipos:

[Volume-based DDoS attacks \(basados en volumen\):](#) El atacante inunda la red o servidor con tráfico masivo (paquetes, conexiones, peticiones).

Objetivo: Saturar el ancho de banda o los dispositivos de red (routers, switches, etc.).

Ejemplos: UDP flood, ICMP flood (ping flood), Amplification attacks (como DNS amplification). Imaginá una autopista llena de autos falsos: los autos legítimos ya no pueden entrar.

[Application DDoS attacks\(a nivel aplicación\):](#) El atacante hace muchas solicitudes legítimas a una aplicación web, por ejemplo, usando HTTP.

Objetivo: Saturar el servidor agotando sus recursos internos (como procesos, base de datos, CPU).

Ejemplos:Enviar miles de solicitudes a /login, /search, /checkout..., HTTP GET/POST flood.Es como entrar a un restaurante y pedir 1.000 cafés. El sistema funciona, pero colapsa por exceso de trabajo.

Low-rate DoS (LDoS) attacks(de baja frecuencia o "inteligentes"):

No envían grandes cantidades de tráfico, sino que **aprovechan errores o debilidades específicas** en el software.

Objetivo: Provocar un fallo o comportamiento anormal en la aplicación usando pocos datos.

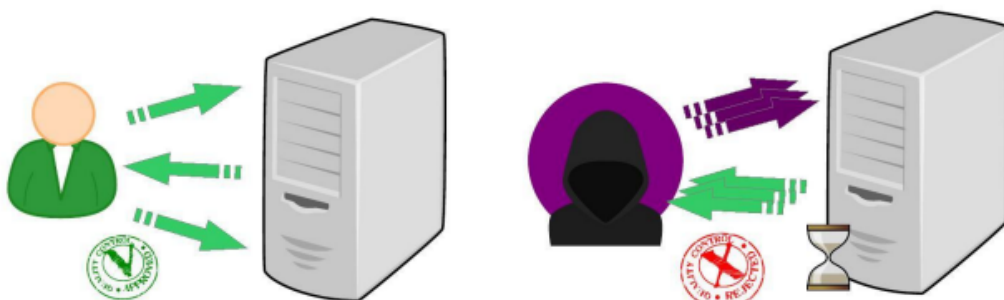
Ejemplos:Mandar un paquete malformado que hace **crashear** un servidor.Hacer una conexión y dejarla **abierta sin cerrarla nunca**.

Explotar un **bug de consumo excesivo** con una sola solicitud.

Es como tirar un palillo en los engranajes de una máquina y hacer que se trabe por completo, sin fuerza ni cantidad.

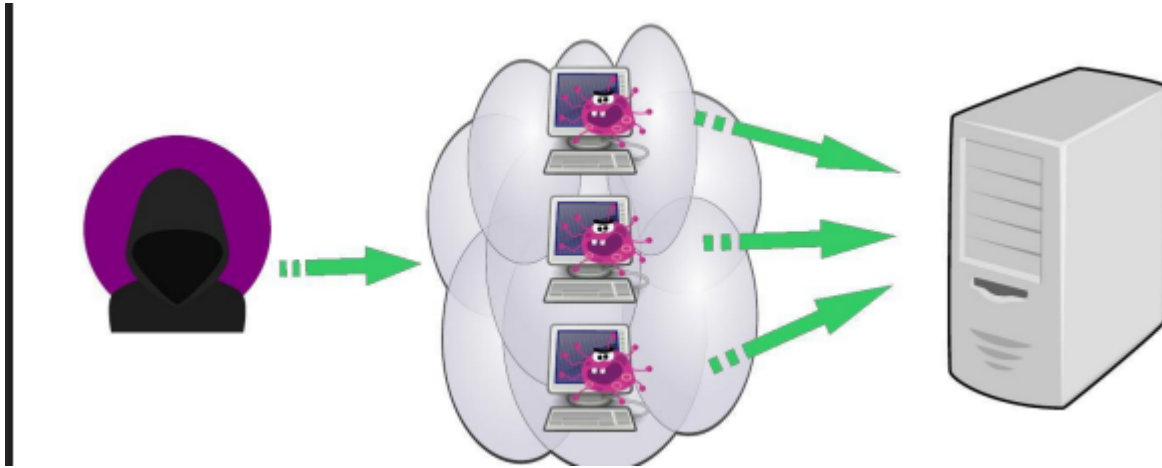
Referencia: Flooding

La técnica de Flooding o Inundación busca generar solicitudes maliciosas a un servicio con la finalidad de hacer que el mismo se sature o entre en un modo de espera, de esta forma anula o limita su funcionamiento.



Referencia: BotNet

Es un conjunto de terminales que ejecutan software que permite su control total o parcial desde ubicaciones remotas. Las terminales se denominan bots o zombies.



Ejemplos de Denegación de servicio (DoS)

- Inundación SYN (SYN Flood)
- Inundación ICMP (ICMP Flood)
- SMURF (ICMP Flood)
- Inundación UDP (UDP Flood)
- Peer-to-peer Utilización de recursos
- A nivel de aplicación
- Degradación de servicio
- Slowloris (HTTP requests parciales. Low-rate)
- Mirai malware (TP-Link routers)

Sniffers



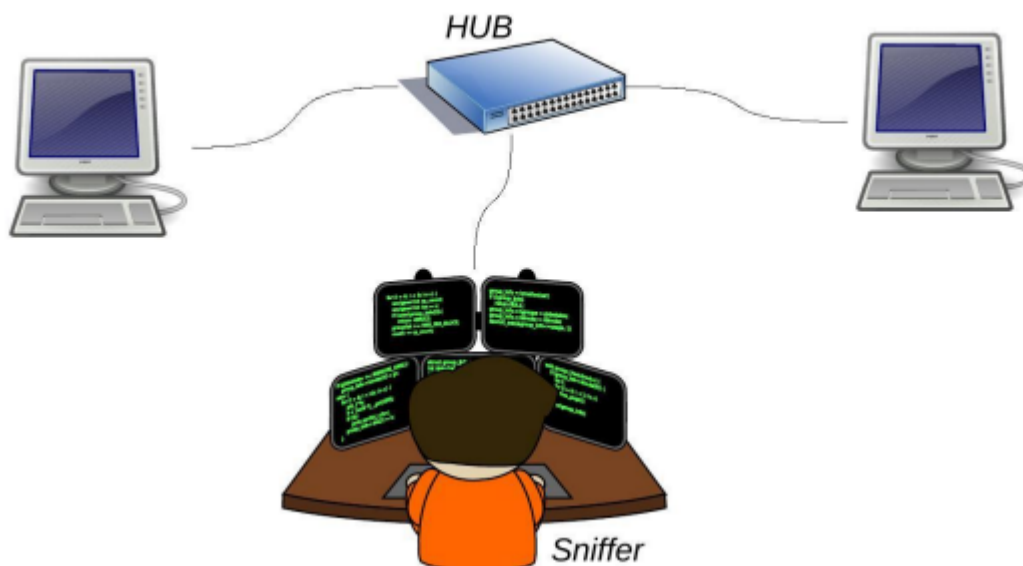
Productos para hacer Sniffing

Un sniffer es un programa que captura y analiza las tramas de red que circulan por una red local (LAN o Wi-Fi). Se utiliza con fines legítimos como la administración de redes, auditorías o aprendizaje, aunque también puede ser empleado con fines maliciosos, como la interceptación de contraseñas, cookies o datos no cifrados.

Existen sniffers pasivos, que solo observan el tráfico, y activos, que también lo modifican. Herramientas conocidas incluyen Wireshark, Tcpdump, Ettercap (para LAN) y Kismet, Aircrack-ng (para redes inalámbricas).

El uso no autorizado de sniffers puede representar una violación legal y de privacidad.

Implementación de Sniffers en la red interna



Atacando a los navegadores (clientes)

Tampering o Data Diddling: Se refiere a la modificación no autorizada de la información. Por ejemplo, múltiples sitios web han sido afectados al detectar cambios en el contenido de sus páginas.

Ataques Mediante JavaScript: JavaScript es un lenguaje de programación usado por los diseñadores de sitios web. Este tipo de programas son utilizados para explotar fallas de seguridad de navegadores web y servidores de correo.

Ataques drive-by download: Infectan de forma masiva a los usuarios, simplemente ingresando a un sitio web determinado. Mediante esta técnica, los desarrolladores de malware (programas maliciosos) propagan sus creaciones e inyectan código dañino entre su código original.

Otras tecnologías generadoras de riesgos

- Javascript
- Activex
- Shockwave
- Java Applets
- Microsoft Silverlight
- Portable Document Format (PDF) Flash
- Plugins de navegador

Otros ataques a clientes

Hijackers: Son programas que alteran el funcionamiento o configuración del cliente para que el atacante pueda "secuestrar" información de interés. Ejemplo, Page hijacking, Session hijacking, Browser hijacking... **Rootkits:** Son programas que permiten que una aplicación maliciosa permanezca oculta en el sistema operativo o que la misma no pueda ser eliminada normalmente. Ejemplo, procesos fantasmas en paralelo.

Backdoors: Son programas que habilitan un acceso alternativo al sistema permitiendo evitar el método de autenticación principal. Normalmente se instalan en los sistemas comprometidos para facilitar su posterior uso(local o remoto) por parte del atacante.

Otros ataques a clientes

Stealers: Son programas que acceden a la información almacenada en el equipo para facilitársela al atacante. Su principal objetivo son contraseñas almacenadas o recordadas en navegadores y clientes de email o mensajería.

Keyloggers: Son programas o dispositivos físicos que registran la actividad de los dispositivos de entrada, comúnmente el teclado.

Ransomware: Son programas que retienen el control del equipo o cifran información almacenada en el mismo para que no pueda ser accedida, en muchos casos solicitan un pago para que sean desactivados. Algunos ejemplos son "Virus Ukash", "WannaCry", "Petya/NotPetya", "Cryptolocker" y "Cryptowall".

OWASP - Riesgos de seguridad en aplicaciones

Proveedores de datos:

- AppSec Labs
- Cobalt.io
- Contrast Security
- GitLab HackerOne
- HCL Technologies
- Micro Focus
- PenTest-Tools
- Probely
- Sgreen
- Veracode
- WhiteHat (NTT)

Principales fuentes de datos:

- Herramienta asistida por Humanos (HaT)
- Humano asistido por Herramientas (TaH)
- Herramientas en bruto

La tasa de incidencia se refiere al porcentaje de la población de aplicaciones que tiene al menos una instancia de un tipo de vulnerabilidad. No importa si fue algo puntual o sistémico; sino que se considera cuántas aplicaciones tenían al menos una instancia. Se analizaron las ocho categorías con las tasas de incidencia más altas, luego se evaluaron los resultados de la encuesta a la comunidad del Top 10 para ver cuáles ya se encuentran presentes en los datos. Las dos más votadas que no están presentes fueron seleccionadas para los otros dos puestos del Top 10. Una vez seleccionadas las diez, se determinan los factores generalizados de

explotabilidad e impacto, y por último se procede a ordenarlas en función del riesgo.

Factores de datos:

CWES mapeadas: Cantidad de *tipos de debilidades* (CWEs) asociadas a una categoría del Top 10. *Ejemplo: "Inyección" puede tener varias CWEs como SQLi, LDAPi, etc.*

Tasa de incidencia: Porcentaje de aplicaciones analizadas que **presentaron al menos una vulnerabilidad de ese tipo**. *Si 30% de apps tenían algún tipo de inyección → tasa de incidencia = 30%*

Cobertura (de pruebas): Porcentaje de aplicaciones que fueron testeadas específicamente para esa vulnerabilidad. Evalúa cuán representativa es la muestra de datos

Explotabilidad ponderada: Qué tan fácil es explotar esa categoría de vulnerabilidad, medido con datos reales (CVSS). Escala de 0 a 10. Cuanto más alto, más fácil de explotar.

Impacto ponderado: Qué tanto daño causa esa vulnerabilidad si se explota, también medido con CVSS. Escala de 0 a 10. Cuanto más alto, más grave el efecto.

Total de ocurrencias: Cantidad de veces que se detectaron esas CWEs en todas las apps analizadas. Mide cuán frecuente es en la práctica.

Total de CVEs: Cantidad de vulnerabilidades documentadas públicamente (CVEs) en el NVD que se relacionan con esas CWEs. Muestra cuántas veces esa falla fue reportada como problema real en software.

Top Ten Web Application Security Risks 2021

- A01 - Pérdida de Control de Acceso
- A02 - Fallas Criptográficas

- A03 - Inyección
- A04 - Diseño Inseguro
- A05 - Configuración de Seguridad Incorrecta
- A06 - Componentes Vulnerables y Desactualizados
- A07 - Fallas de Identificación y Autenticación
- A08 - Fallas en el Software y en la Integridad de los Datos
- A09 - Fallas en el Registro y Monitoreo
- A10 - Falsificación de Solicitudes del Lado del Servidor

A1 - Pérdida del Control de Acceso

Muchas aplicaciones Web verifican el nivel de acceso a las funciones justo antes de hacer estas funcionalidades y/o datos visibles en la interfaz gráfica. Sin embargo, las aplicaciones necesitan realizar el mismo control de acceso del lado del servidor al momento que cada función y/o dato es accedido. Si las solicitudes no son verificadas, los atacantes serán capaces de forzar peticiones con la finalidad de acceder a la funcionalidad sin la autorización apropiada.

El atacante simplemente navega forzosamente a la URL objetivo. Considere las siguientes URLs las cuales se supone que requieren autenticación. Para acceder a la página "admin_getapplInfo" se necesitan permisos de administrador.

`http://ejemplo.com/app/getapplInfo` `http://ejemplo.com/app/admin_getapplInfo` Si un atacante no autenticado puede acceder a cualquiera de estas páginas entonces se ha permitido acceso no autorizado. Si un usuario autorizado, no administrador, puede acceder a la página "admin_getapplInfo", esto es un fallo, y puede llevar al atacante a otras páginas de administración que no están debidamente protegidas. Este tipo de vulnerabilidades se encuentran con frecuencia cuando links y botones simplemente se ocultan a usuario no autorizados, pero la aplicación no protege adecuadamente las páginas de destino.

(Referencia Directa Insegura a Objetos): La aplicación utiliza datos no verificados en una llamada SQL que accede información sobre la cuenta: `String query = "SELECT * FROM accts WHERE account = ?"; PreparedStatement pstmt = connection.prepareStatement(query, ...) pstmt.setString(1, request.getParameter("acct"));` `ResultSet results = pstmt.executeQuery();` El atacante simplemente modificaría el parámetro "acct" en su navegador para enviar cualquier número de cuenta que quiera. Si esta acción no se verifica, el atacante

podría acceder a cualquier cuenta de usuario, en vez de a su cuenta de cliente correspondiente.

Medidas de prevención:

- Utilizar referencias indirectas por usuario o sesión
- Comprobar el nivel de acceso al objeto
- Control de acceso único y reutilizado en toda la aplicación. Minimizar el control de acceso HTTP (CORS).
- Deshabilitar el listado de directorios del servidor web y asegurar que los metadatos de archivos
- Limite la tasa de acceso a APIs y al control de acceso
- Los tokens JWT deben ser invalidados luego de la finalización de la sesión por parte del usuario.
- El proceso para gestión de accesos y permisos debe ser actualizable y auditable fácilmente
- La implementación debería negar todo acceso por defecto, requiriendo el establecimiento explícito de permisos a roles específicos para acceder a cada funcionalidad
- Si la funcionalidad forma parte de un Workflow, se debe verificar y asegurar las condiciones del flujo para permitir el acces
- NO implementar controles en la capa de visualización

A2 - Fallas Criptográficas

La información sensible demanda protección adicional como la encriptación en su almacenamiento y tránsito, al igual que precauciones especiales cuando es intercambiada con el cliente o navegador.

Escenario #1: Una aplicación encripta los números de tarjetas de crédito utilizando el cifrado automático de la base de datos. De todas formas, esto significa que dicha información también es descifrada automáticamente al ser solicitada, permitiendo que una falla de Inyección de SQL pueda permitir que se obtengan los números de tarjetas de crédito en texto limpio. El sistema debería haber encriptado los números de tarjetas de crédito en un nivel superior utilizando un soporte criptográfico adecuado; por ejemplo cifrando los valores con una llave pública y permitiendo que sólo las aplicaciones de back-end pudieran descifrarlos con la llave privada.

Escenario #2: Un sitio web no utiliza o fuerza el uso de TLS para todas las páginas, o utiliza cifradores débiles. Un atacante monitorea el tráfico de la red, degrada la conexión de HTTPS a HTTP e intercepta los pedidos, robando las cookies de sesión del usuario. Reutiliza esta cookie y secuestra la sesión del usuario, accediendo o modificando datos privados.

Escenario #3: La base de contraseñas utiliza "unsalted hashes" para almacenar las contraseñas de todos los usuarios. Una debilidad de upload de archivos permitiría al atacante obtener el archivo de contraseñas. Todos los "unsalted hashes" podrían quedar expuestos con una tabla de hashes precalculados.

Medidas de prevención:

- Cifrar los datos sensibles o en tránsito de manera de defenderse de las amenazas
- No almacene datos sensibles innecesariamente
- Aplicar algoritmos de cifrado fuertes y estándar al igual que claves robustas y gestionadas de forma segura.
- Almacenar claves con algoritmos diseñados para tal fin (bcrypt, scrypt, PBKDF2)
- Deshabilitar el auto-completar en los formularios y cache de páginas que manejan datos sensibles.

A3 - Inyección

Las fallas de inyección, tales como SQL, NoSQL, Object-Relational Mapping (ORM), OS, y LDAP, ocurren cuando datos no confiables son enviados a un intérprete como parte de un comando o consulta. Los datos hostiles del atacante pueden engañar al intérprete en ejecutar comandos no intencionados o acceder datos no autorizados.

La aplicación utiliza datos no confiables en la construcción de la siguiente consulta vulnerable SQL: String query = "SELECT * FROM accounts WHERE custID=" + request.getParameter("id") + .; El atacante modifica el parámetro 'id' en su navegador para enviar: ' or '1'='1. Esto cambia el significado de la consulta devolviendo todos los registros de la tabla ACCOUNTS en lugar de solo el cliente solicitado. http://example.com/app/accountView?id=' or '1'='1 En el peor caso, el

atacante utiliza esta vulnerabilidad para invocar procedimientos almacenados especiales en la base de datos que permiten la toma de posesión de la base de datos y posiblemente también al servidor que aloja la misma.



Medidas de prevención:

- Uso de APIs con manejo parametrizado de interpretes o una herramienta de Mapeo Relacional de Objetos (ORMs)
- Codificación de los caracteres especiales en función del interprete a utilizar
- Validación de entradas positiva o de "lista blanca"
- Utilice LIMIT y otros controles SQL dentro de las consultas para evitar la divulgación masiva de registros.

Referencia: CSRF La aplicación permite que los usuarios envíen peticiones de cambio de estado, que no incluyen nada secreto. Por ejemplo:

http://example.com/app/transferFunds?

amount=1500&destinationAccount=4673243243 El atacante podrá insertar su ataque dentro de una etiqueta de imagen en un sitio web, o iframe, que esté bajo su control y al que la víctima se podrá dirigir.Cuando la víctima visite el sitio, en lugar de cargarse la imagen, se realizará la petición HTTP falsificada.

Medidas preventivas:Utilizar un token único y no predecible, este se debería incluir en un campo oculto Requerir nueva autenticación del usuario antes de ejecutar el pedido.

Secuencia de comandos en sitios cruzados (XSS-Cross Site Scripting)

XSS es una vulnerabilidad que ocurre cuando una aplicación **inserta datos no confiables en una página web sin validarlos ni codificarlos** correctamente. Esto permite a un atacante **ejecutar scripts maliciosos en el navegador de la**

víctima, robando sesiones, redirigiendo al usuario o manipulando el contenido del sitio.

Existen tres formas usuales de XSS:

XSS Reflejado (Reflected XSS):

- El script malicioso se inserta en una URL o formulario.
- La respuesta lo refleja directamente en el HTML sin validación.
- Requiere que la víctima haga clic en un enlace manipulado.

 Común en correos phishing o enlaces falsos.

XSS Almacenado (Stored XSS):

- El script se guarda en la base de datos u otro sistema del servidor.
- Cuando otro usuario carga la página, el código se ejecuta automáticamente.
- Es el más peligroso, porque puede afectar a muchos usuarios sin interacción previa.

 Ejemplo típico: comentario malicioso en un foro.

XSS Basados en DOM(DOM-based XSS):

- El ataque ocurre dentro del navegador, usando JavaScript.
- La página web usa datos del usuario sin sanear y los inyecta en el DOM.
- No pasa por el servidor, lo ejecuta directamente el navegador.

 Ejemplo: JavaScript que usa `location.hash` sin validación.

Medidas de prevención:

- Codificar los datos no confiables basados en el contexto HTML(cuerpo, atributo, JavaScript, CSS o URL) donde serán ubicados.
- Validación de entrada positiva o de "Lista blanca"
- Para formato enriquecido considere utilizar APIs de auto sanitización. (Ejemplo AntiSamy)
- Considerar el uso de políticas de seguridad de contenido CSP

A4 - Diseño inseguro

Se centra en los riesgos relacionados con el diseño y las fallas arquitectónicas, con un llamado a un mayor uso del modelado de amenazas, patrones de diseño seguros y arquitecturas de referencia. Como comunidad, debemos ir más allá de solo abandonar las metodologías tradicionales (movimiento "shiftleft" en inglés) en el espacio de codificación para precodificar actividades que son críticas para los principios de Secure by Design.

Escenario #1: Un flujo de trabajo de recuperación de credenciales puede incluir "preguntas y respuestas", lo cual está prohibido por NIST 800-63b, OWASP ASVS y OWASP Top 10. No se puede confiar en preguntas y respuestas como evidencia de identidad como más de una persona puede conocer las respuestas, por lo que están prohibidas. Dicho código debe eliminarse y reemplazarse por un diseño más seguro.

Escenario #2: Una cadena de cines permite descuentos en la reserva de grupos y tiene un máximo de quince asistentes antes de solicitar un depósito. Los atacantes podrían modelar este flujo y probar si podían reservar seiscientos asientos y todos los cines a la vez en unas pocas solicitudes, lo que provocaría una pérdida masiva de ingresos.

Escenario #3: El sitio web de comercio electrónico de una cadena minorista no tiene protección contra bots administrados por revendedores que compran tarjetas de video de alta gama para revender sitios web de subastas. Esto crea una publicidad terrible para los fabricantes de tarjetas de video y los propietarios de cadenas minoristas y una mala sangre duradera con entusiastas que no pueden obtener estas tarjetas a ningún precio. El diseño cuidadoso de anti-bot y las reglas

de lógica de dominio, como las compras realizadas a los pocos segundos de disponibilidad, pueden identificar compras no auténticas y rechazar dichas transacciones.

Medidas de prevención:

- Establezca y use un ciclo de vida de desarrollo seguro con Aplicaciones de Seguridad profesionales para ayudar a evaluar y diseñar la seguridad y controles relacionados con la privacidad. Establecer y utilizar una biblioteca de patrones de diseño seguros componentes de "Paved Road"
- Utilice el modelado de amenazas para autenticación crítica, control de acceso, lógica empresarial y flujos clave.
- Integre el lenguaje y los controles de seguridad en las "historias de usuario".
- Integre verificaciones de admisión en cada nivel de su aplicación (de frontend a backend)
- Escribir pruebas de integración y pruebas unitarias para validar que todos los flujos críticos son resistentes al modelo de amenazas.
- Compilar casos de uso y casos de uso indebido para cada nivel de su aplicación.
- Separe las capas de niveles en el sistema y las capas de red según las necesidades de exposición y protección.
- Separe a los tenants(usuarios que comparten privilegios de acceso al software) de manera robusta por diseño en todos los niveles. Limitar el consumo de recursos por usuario o servicio.

A5 - Configuración de Seguridad Incorrecta

Una buena seguridad requiere tener definida e implementada una configuración segura para la aplicación, marcos de trabajo, servidor de aplicación, servidor web, base de datos, y plataforma. Todas estas configuraciones deben ser definidas, implementadas, y mantenidas ya que por lo general no son seguras por defecto. Esto incluye mantener todo el software actualizado, incluidas las librerías de código utilizadas por la aplicación.

Escenario #1: La aplicación está basada en un ambiente de trabajo como Struts o Spring. Se han presentado defectos de XSS en algunos de los componentes que utiliza la aplicación. Se ha liberado una actualización que sirve para corregir esos

defectos. Hasta que no se realicen dichas actualizaciones, los atacantes podrán encontrar y explotar los fallos, ahora conocidos, de la aplicación.

Escenario #2: La consola de administración del servidor de aplicaciones está instalada y no ha sido removida. Las cuentas predeterminadas no han sido cambiadas. Los atacantes descubren que las páginas de administración están activas, se registran con las claves predeterminadas y toman posesión de los servicios.

Escenario #3: La configuración del servidor de aplicaciones permite que se devuelvan a los usuarios mensajes de error detallados, por ejemplo, seguimientos de pila(stack traces), exponiendo información confidencial o fallas subyacentes, como versiones de componentes vulnerables.

Entidad Externa de XML (XXE)

XXE es una vulnerabilidad que ocurre cuando un procesador XML antiguo o mal configurado permite cargar entidades externas definidas en el XML, lo que puede ser aprovechado por un atacante.

¿Qué puede causar?

- Lectura de archivos internos del servidor
- Exposición de rutas SMB (en Windows antiguos)
- Escaneo de puertos internos
- Ejecución remota de código (RCE)
- Ataques de denegación de servicio (DoS) como el "Billion Laughs"

Ejemplo: El XML incluye una entidad como:

```
<!ENTITY xxe SYSTEM "file:///etc/passwd">
```

Escenario #1: El atacante intenta extraer datos del servidor:

Escenario #2: Un atacante sondea la red privada del servidor cambiando la línea ENTITY anterior por:]>

Escenario #3: Un atacante intenta un ataque de denegación de servicio incluyendo un archivo potencialmente infinito: !ENTITY xxe SYSTEM "file:///dev/random".

Medidas de prevención:

- Usar formatos de datos menos complejos como JSON y evitar la serialización de datos confidenciales
- Actualizar procesadores y bibliotecas XML
- Usar validadores de dependencias y SOAP a 1.2 o superior
- Deshabilitar entidades externas de XML y procesamiento
- DTD. Validar el XML entrante usando validación XSD
- Implementar validación de entrada positiva ("lista blanca"), filtrado, o sanitización.
- Disponer de un proceso rápido, fácil y repetible de fortalecimiento para obtener un entorno apropiadamente asegurado.
- Un proceso para mantener y desplegar las actualizaciones y parches en cada entorno
- Una arquitectura de aplicación que proporcione separación segura y efectiva entre los componentes
- Ejecutar escaneo y realizar auditorías regularmente para identificar fallos de configuración o parches omitidos

A6 - Componentes Vulnerables y Desactualizados

Componentes, como librerías, frameworks, y otros módulos de software, son siempre ejecutados con privilegios completos. Si un componente vulnerable es explotado, podría darse lugar a un ataque que pueda facilitar una seria pérdida de datos o comprometer el servidor. Las aplicaciones que utilizan componentes con vulnerabilidades conocidas deberían determinar las defensas de la aplicación para el caso y habilitar un rango de posibles ataques e impacto de los mismos.

Apache CXF Authentication Bypass - Por fallar al proveer un bloque de identificación, los atacantes pudieron invocar cualquier web service con permisos completos. (Apache CXF es un framework de servicios, no debe ser confundido con Apache Application Server.) Spring Remote Code Execution - Abusando de la implementación del "Expression Language" en Spring se permitía a los atacantes la ejecución de código arbitrario, tomando posesión efectiva del servidor.

Dispositivos del internet de las cosas (IoT) - frecuentemente son imposibles o muy dificultosos de ser actualizados, la importancia de éstas actualizaciones puede ser enorme (por ejemplo dispositivos biomédicos).

Medidas de prevención:

- Identificar todos los componentes y la versión que están ocupando, incluyendo dependencias .
- Revisar la seguridad del componente en bases de datos públicas, lista de correos del proyecto, y lista de correo de seguridad, y mantenerlos actualizados
- Establecer políticas de seguridad que regulen el uso de componentes, como requerir ciertas prácticas en el desarrollo de software, pasar test de seguridad, y licencias aceptables
- Sería apropiado, considerar agregar capas de seguridad alrededor del componente para deshabilitar funcionalidades no utilizadas y/o asegurar aspectos débiles o vulnerables del componente.

A7 - Fallas de Identificación y Autenticación

Las funciones de la aplicación relacionadas a autenticación y gestión de sesiones son frecuentemente implementadas incorrectamente, permitiendo a los atacantes comprometer contraseñas, claves, token de sesiones, o explotar otras fallas de implementación para asumir la identidad de otros usuarios (temporalmente o permanentemente).

Escenario #1: Relleno de credenciales, el uso de listas de contraseñas conocidas, es un ataque común.

Escenario #2: La mayoría de los ataques de autenticación ocurren debido al uso de contraseñas como único factor.

Escenario #3: Los tiempos de vida de las sesiones de aplicación no están configurados correctamente. Un usuario utiliza una computadora pública para acceder a una aplicación. En lugar de seleccionar "logout", el usuario simplemente cierra la pestaña del navegador y se aleja. Un atacante usa el mismo navegador una hora más tarde, y el usuario continúa autenticado.

Escenario #4: Aplicación de reserva de vuelos que soporta re-escritura de direcciones URL poniendo los identificadores de sesión en la propia dirección: `http://example.com/sale/saleitems.jsessionid=2P0OC2JDPXM0OQSNLPSKHJCJUN2JV?dest=Hawaii` Un usuario autenticado en el sitio quiere mostrar la venta a sus amigos. Envía por correo electrónico el enlace anterior, sin ser consciente de que está proporcionando su identificador de sesión. Cuando sus amigos utilicen el anterior enlace utilizarán su sesión y su tarjeta de crédito.

Medidas de prevención:

- Disponer de un único y robusto conjunto de controles de autenticación y gestión de sesiones
- Realizar un gran esfuerzo para evitar vulnerabilidades de XSS que pueden dar espacio al robo de IDs de sesión
- Implementar autenticación multifactorial
- Implemente un control contra contraseñas débiles
- Límite o incremente el tiempo de respuesta de cada intento fallidos de inicio de sesión
- Alinear las políticas de largo, complejidad y rotación de las contraseñas

A8 - Fallas en el Software y en la Integridad de los Datos

Deserialización Insegura:

Los fallos de integridad del software y de los datos están relacionados con código e infraestructura no protegidos contra alteraciones (integridad). Incluiremos la dependencia en plugins, bibliotecas o módulos de fuentes, repositorios o redes de entrega de contenidos (CDN) no confiables. Así como también actualizaciones automáticas adulteradas o alteraciones de objetos o datos serializados.

Escenario #1: Actualizaciones no firmadas: Muchos routers domésticos, decodificadores de televisión, firmware de dispositivos, entre otros, no verifican las firmas de sus actualizaciones de firmware. El firmware sin firmar es un objetivo creciente para los atacantes y se espera que empeore. Esto es una gran preocupación, ya que muchas veces no existe otro mecanismo para remediarlo que corregirlo en una versión futura y esperar a que las versiones anteriores caduquen.

Escenario #2: Actualización maliciosa de SolarWinds: un caso de pública notoriedad fue el sufrido por SolarWinds Orion. La compañía que desarrolló el software poseía procesos seguros de construcción y mecanismos de integridad en sus actualizaciones. Sin embargo, estos fueron comprometidos y, durante varios meses, la firma distribuyó una actualización maliciosa a más de 18.000 organizaciones, de las cuales alrededor de un centenar se vieron afectadas. Se trata de una de las brechas de este tipo de mayor alcance y más importantes de la historia.

Escenario #3: Deserialización insegura: Una aplicación React utiliza un conjunto de microservicios implementados en Spring Boot. Tratándose de programadores funcionales, intentaron asegurarse de que su código sea inmutable. La solución implementada consistió en serializar el estado de la sesión para el usuario y enviarlo entre los componentes con cada solicitud. Un atacante advierte el uso de un objeto Java serializado y codificado en base64 (identifica un string que comienza con "rO0") y utiliza la herramienta Java Serial Killer para obtener una ejecución remota de código en el servidor de aplicación. Defectos de deserialización insegura ocurren cuando una aplicación recibe objetos serializados hostiles. La deserialización insegura conduce a la ejecución remota de código. Incluso si el defecto de deserialización no resultara en la ejecución remota de código, los objetos serializados pueden ser reproducidos, manipulados o borrados por el atacante, realizar ataques de inyecciones o elevar sus privilegios.

Deserialización Insegura Escenario #1: Un foro PHP utiliza serialización de objetos PHP para almacenar una "super" cookie, conteniendo el ID, rol, hash de la contraseña y otros estados del usuario:

Medidas de prevención:

- El único patrón de arquitectura seguro es no aceptar objetos serializados de fuentes no confiables o utilizar medios de serialización que sólo permitan tipos de datos primitivos
- Implementar verificaciones de integridad como firmas digitales en objeto serializados
- Cumplimiento estricto de verificaciones de tipo de dato durante la deserialización y antes de la creación del objeto
- Restringir o monitorear las conexiones de red entrantes y salientes vinculadas a deserialización.

A9 - Fallas en el Registro y Monitoreo

El registro y monitoreo insuficiente, junto con la falta integración ineffectiva de respuesta de incidentes permiten a los atacantes persistir en el tiempo el ataque al sistema, pivotear a más sistemas y manipular, extraer o destruir datos. La mayoría de los estudios muestran que el tiempo de detección de una violación de seguridad es mayor a 200 días, siendo típicamente detectado por terceros en lugar de procesos internos o monitoreo.

Escenario #1: El software de un foro de código abierto es operado por un pequeño equipo que fue hackeado utilizando una falla de seguridad en su software. Los atacantes lograron eliminar el repositorio del código fuente interno que contiene la próxima versión, y todos los contenidos del foro. Aunque el código fuente pueda ser recuperado, la falta de monitorización, registro y alerta condujo a una brecha de seguridad aún peor. El proyecto de software de éste foro ya no está activo debido a éste problema.

Escenario #2: Un atacante escanea usuarios utilizando la contraseña por defecto, pudiendo tomar el control de todas las cuentas utilizando ésta contraseña. Para todos los demás usuarios, éste proceso deja únicamente 1 solo registro de fallo de inicio de sesión. Luego de algunos días, esto puede repetirse con una contraseña distinta.

Escenario #3: De acuerdo a reportes, un importante minorista de los Estados Unidos tenía un sandbox de análisis de malware interno para el análisis de archivos adjuntos de correos electrónicos. El sandbox había detectado software potencialmente indeseable, pero nadie respondió a esta detección. El sandbox

había estado generando advertencias por algún tiempo antes de que la brecha de seguridad fuera detectada debido a transacciones de tarjeta fraudulentas por un banco externo.

Escenario #4: Un operador de salud que provea un plan de salud para niños no pudieron detectar una brecha debido a la falta de monitoreo y registro. Alguien externo informó al proveedor de salud que un atacante había accedido y modificados miles de registros médicos sensibles de más de 3.5 millones de niños. Una revisión post incidente encontró que los desarrolladores del sitio web no habían encontrado vulnerabilidades significativas. Como no hubo ni registro ni monitores del sistema, la brecha de datos pudo haber estado en proceso desde el 2013, un período de más de 7 años.

Escenario #5: Una gran aerolínea India tuvo una brecha de seguridad que involucró a la pérdida de datos personales de millones de pasajeros por más de 10 años, incluyendo pasaportes y tarjetas de crédito. La brecha se produjo por un proveedor de servicios de almacenamiento en la nube, quien notificó a la aerolínea después de un cierto tiempo. Escenario #6: Una gran aerolínea Europea sufrió un incumplimiento de la GRPD. Se reporta que la causa de la brecha se debió a que un atacante explotó una vulnerabilidad en una aplicación de pago, obteniendo más de 400,000 registros de pagos de usuarios. La aerolínea fue multada con 20 millones de libras como resultado del regulador de privacidad.

Medidas de prevención:

- Todos los errores de inicio de sesión, control de acceso y de validación de entradas de datos del lado del servidor se deben registrar con el contexto de usuario suficiente para identificar cuentas sospechosas o maliciosas, y mantenerlo durante el tiempo suficiente para permitir un eventual análisis forense.
- Asegúrese de que las transacciones de alto impacto tengan una pista de auditoría con controles de integridad para prevenir alteraciones o eliminaciones.
- Todas las transacciones de alto valor deben poseer una traza de auditoría con controles de integridad que permitan detectar su modificación o borrado.
- Implementar una monitorización y alerta efectivos de tal manera que las actividades sospechosas sean detectadas y respondidas dentro de periodos de tiempo aceptables.

A10 - Falsificación de Solicitudes del Lado del Servidor (SSRF)

Las fallas de SSRF ocurren cuando una aplicación web está obteniendo un recurso remoto sin validar la URL proporcionada por el usuario. Permite que un atacante coaccione a la aplicación para que envíe una solicitud falsificada a un destino inesperado, incluso cuando está protegido por un firewall, VPN u otro tipo de lista de control de acceso a la red (ACL).

Medidas de prevención:

- Sanitice y valide todos los datos de entrada proporcionados por el cliente .
- Haga cumplir el esquema de URL, el puerto y el destino con una lista positiva de ítems permitidos
- No envíe respuestas en formato "crudo" a los clientes
- Deshabilite las redirecciones HTTP
- Tenga en cuenta la coherencia de la URL para evitar ataques como el enlace de DNS y las condiciones de "tiempo de verificación, tiempo de uso"

Escenario #1: Escaneo de puertos de servidores internos - Si la arquitectura de la red no está segmentada, los atacantes pueden trazar un mapa de las redes internas y determinar si los puertos están abiertos cerrados en los servidores internos a partir de los resultados de la conexión o del tiempo transcurrido para conectar o rechazar las conexiones de payload SSRF.

Escenario #2: Exposición de datos sensibles: los atacantes pueden acceder a archivos locales como servicios internos para obtener información confidencial como file:///etc/passwd y <http://localhost:28017/>.

Controles Pro-activos 2024

- C1 - Implementar control de acceso
- C2 - Usar la criptografía de la manera adecuada
- C3 - Validar todas las entradas y manejar las excepciones
- C4 - Abordar la seguridad desde el inicio
- C5 - Configurar de manera segura por defecto
- C6 - Mantener seguros tus componentes

- C7 - Implementar identidad digital
- C8 - Aprovechar las características de seguridad del navegador
- C9 - Implementar registros y monitoreo de seguridad
- C10 - Detener la falsificación de solicitudes del lado del servidor (SSRF)