[SAP] TurboResumenExpresSap.txt

Tema 1: Introducción a las aplicaciones seguras

1.1 Definiciones

- **Autenticación**: Un usuario tiene que demostrar que es quien dice ser. Se puede realizar de diferentes formas:
  - Usuario y contraseña: Las contraseñas deben ser robustas para prevenir ataques de fuerza bruta, además de ser guardadas cifradas por si la BBDD que las almacena es comprometida
    - Las contraseñas no siempre se transmiten por un medio seguro (HTTP)
    - La misma contraseña puede ser utilizada en sitios diferentes por el mismo usuario.
  - Tarjetas inteligentes: Se utiliza una smart card con un certificado digital protegido por pin
  - Biometría: Se identifica al usuario usando métodos biométricos que lo identifican en función a sus rasgos fisiológicos.
    - Biometría Fisiológica: Huella dactilar, reconocimiento facial…
    - Biometría conductal: Reconocimiento por voz, movimientos….
  - Certificado digital: Se consta de un par de claves criptográficas (pública y privada) y se realiza la verificación de este usando ciptografía asimétrica. Los datos que se cifran con la clave pública solo pueden ser descifrados por la privada.
    - X.509 es el estándar que define el formato de los certificados digitales, conteniendo info sobre la entidad propietarria del certificado, la clave pública de esta e información sobre la entidad certificadora que ha emitido el certificado.
- **Autorización**: cuando un usuario se autentica, una política de autorización establece las acciones que puede realizar el usuario y a que recursos puede acceder.
- **Control de acceso**: Se encarga de verificar que un usuario tiene los permisos para realizar una acción. En resumidas cuentas, el control de acceso se encarga de hacer cumplir las políticas de autorización.

1.2 Aplicaciones y servicios con estado (stateful)

En las aplicaciones con estado, el servidor mantiene información sobre el cliente desde que se conecta hasta que se desconecta.

- La info de cada cliente se guarda en un objeto llamado **sesión**.
  - Se suele implementar con estructura de mapa de pares clave-valor.
  - Dentro de una sesión se pueden almacenar datos relevantes del usuario, además de otro tipo de info.
  - La sesión puede ser presistente, estando almacenada en base de datos o puede perderse cada vez que se reinicie el servidor.
  - En la API de servlets de Java la sesión se implementa con el objeto “HttpSession” y se puede obtener con “HttpServletRequest”

        - HttpSession HttpServletRequest.getSession (boolean Create): Permite obtener la sesión asociada a la petición
        - HttpSession.setAttribute(String name, Object value): Almacena un objeto en la sesión identificado por nombre
        - httpSession.getAttrubute(String name): Permite obtener un objeto almacenado en la sesión a partir de su nombre.
- Para identificar a que cliente pertenece cada sesión se suelen usar las **Cookies**.
    - Cuando el server pide una petición del cliente crea una sesión y responde a la petición con la cookie de sessión (JSESSIONID en java o ASP.NET_SessionId en .NET). En las siguiente peticiones el cliente enviará la cookie de sesión.
    - Las cookies permiten el almacenamiento de info en el cliente y se suelen usar para:
        - Mantenimiento de la sesión
        - Personalización
        - Tracking
    - El servidor envía las Cookies al cliente en la cabecera HTTP “Set-Cookie” y el cliente las envía con “Cookie”
    - En una cookie se pueden especificar parámetros como
        - Expires: Indica una fecha de caducidad. (Usado en cookies persistentes)
        - Max-Age: Limita el tiempo de vida de la cookie estableciendo un máximo de tiempo en segundos.
            - Si no se especifica el tiempo, la cookie es eliminada al cerrar el navegador.
            - Se usa en cookies persistentes.
        - Secure: Se enviarán las cookies por HTTPS
        - HttpOnly: Las cookies no serán accesibles desde JavaScript
        - Domain: Especifica los subdominios a los que se puede enviar la cookie
        - Path: Indica los direcctorios a los que se enviará la cookie
        - Same Site: permite especificar que no se envíe la cookie si la petición viene de un sitio diferente.
            - *Strict*: solo se envía al mismo domino de la cookie
            - *Lax*: Se puede enviar a un dominio externo cuadno la petición cambia visiblemente la URL (Bloquea iFrames y AJAX)
    - Cuando un navegador no soporta cookies, la sesión puede ser manejada por reescritura de URL

1.3 Aplicaciones y servicios web sin estado (stateless)

En las aplicaciones stateless el server no almacena información sobre el estado del cliente, las sesiones no existen. El server procesa las solicitudes sin tener en cuenta peticiones anteriores del cliente. Las peticiones realizadas deben incluir la información necesaria para que el server pueda identificar al usuario.

- La info necesaria para identificar al usuario se enviará en la primera conexión con el server una vez realizada la autenticación de forma exitosa.
- En las siguientes peticiones el cliente manda su información identificativa:
    - De forma eficiente: No será necesario acceder a una BBDD para identificar al usuario
    - De manera segura: La información debe garantizar que los datos no son manipulados de forma maliciosa.
- Para almacenar esta información se usan JSON Web Tokens (JWT), que guardan información del usuario y mantienen la integridad de los datos a través de una firma.

- Si el token contiene info sensible se puede cifrar
- La transmisión de estos tokens se pueden hacer mediante Cookies o Cabeceras HTTP.

1.4 Aplicaciones Web tradicionales y SPA

- En las **aplicaciones tradicionales**, cada petición generada por un evento del usuario produce una recarga completa de la página, perdiendo el navegador el estado que tenía antes de la recarga. Esto dificulta hacer interfaces de usuario interactivas
- En las aplicaciones web **SPA (Single Page Application)** no se produce una recarga de la página tras un evento de usuario. En este caso solo se modifica un fragmento del arbol DOM, permitiendo itnerfaces más interactivas.

Tema 2: Vulnerabilidades y mecanismos de prevención

2.1 Marcos de referencia

- **MITRE** (MITRE): Organización encargada de registrar y publicar información relativa a vulnerabilidades y ataques conocidos dentro del ámbito de la seguridad.
  - CWE (Common Weakness Enumeration): Clasificación de debilidades de software. Dirigido a desarrolladores y profesionales de la seguridad
    - Se catalogan debilidades o malas prácticas de programación que pueden dar lugar a vulnerabilidaeds
    - Se recomienda conocer esta lista a la hora de desarrollar cualquier tipo de software.
    - La CWE tiene diferentes tipos de clasificaciones, desde listado de todas las entradas hasta agrupaciones por criterios
      - Agrupación de vulnerabolidades relacionadas con la investgación
      - Agrupación de vulnerabilidades relacionadas con el desarrollo
      - Agrupación de vulnerabilidades relacionadas con el hardware
    - Los tipos delemenetos del catálogo del CWE son los siguientes:
      - Clases: Vulnerabilidades descritas de forma genérica
      - Vulnerabilidades base: Descritas de forma báscia pero con suficientes detalles como para detectarlas y prevenirlas
      - Variantes: vulnerabilidad descrita de forma muy detallada
      - Composiciones: elemetno compuesto de dos o mas vulnerabilidades
      - Vistas: Subconjunto de elementos agrupados a mejorar la viusalicación dentro de la web del CWE
      - Categorías: Agrupación de elementos que comparten las mismas características.
  - CVE (Common Vulnerabilites and Exposures): Lista de vulnerabildiades conocidas detectadas en programas o librerías.
    - Cada entrada del CVE explota uno o varios tipos de vulnerabilidades, por lo que se asocian a una o más entradas de CWE
    - Normalmente el formato que usan es “CVE-<Año>-<Número>”
  - Las entidades que designan los identificadores CVE se denominan CNA (CVE Numbering

Authorities)
        - La principal CNA es el MITRE
        - Algunas compañías que participan en el programa también pueden identificar CVEs
- **NVD** (National Vulnerability Database): Proyecto del gobierno de EEUU encargado de recopilar y gestionar info sobre vulnerabildiades. Contiene los datos de CVE ampliados con análisis adicionales.
- **CAPEC** (Common Attack Pattern Enumeration and Classification): Catálogo de patrones de ataque (Recopilación de métodos que se utilizan para explotar vulnerabilidaeds). Cada patrón de ataque se detalla con una descripción, los pasos para realizar el ataque, prerrequisitos necesarios y posibles soluciones y mitigaciones.
- **CVSS** (Common Vulnerability Scoring System): Métrica para determinar la criticidad e impacto de las vulnerabildaes. Es gestionado por el FIRST (Forum of Incident Response and Security Teams), una confederación internacional de equipos de respuestas a incidentes
    - Métrica Base: Cualidades intrínsecas independientes del entrono y tiempo:
        - Vector de ataque (Local, lan, remoto…)
        - Complejidad del ataque
        - Privilegios necesarios (autenticación)
        - Métricas de impacto sobre la confidencialidad, integridad y disponibilidad
    - Métrica temporal: Características que varían con el timepo
        - Explotabilidad
        - Estado de la medida correctora
        - Fiabilidad del informe sobre la vulnerabilidad
    - Métrica de entorno: Características relacionadas con el entorno que sufre el problema
- CWSS (Common Weakness Scoring System): Mecanismo que permite priorizar las debilidades en el software. Agrupa varios tipos de métricas
    - Base: Riesgos inherentes a la vulnerabilidad
    - Superficie de ataque: Barreras que el ataque debe superar
    - Entorno: Características específicas de un entorno concreto en el que se ha encontrado la vulnerabilidad
- OWASP (Open Web Application Security Project): Comunidad dedicada a promover y mantener diferentes materiales relacionados con la seguridad.
    - OWASP Vulnerable Web Application Directory: Colección de aplicaciones web vulnerables
    - OWASP Testing Guide: Metodología para realizar una auditoría de seguridad sobre una aplicación web.
    - OWASP Top 10: Los 10 riesgos de seguridad más importantes en aplicaciones web.
        - *A01:2021 - Broken Access Control*: Vulnerabilidades en el control de acceso, detectado en el 94% de las aplicaciones web analizadas.
        - *A02:2021 - Cryptographic Failures*: Problemas criptográficos que peuden derivar en la exposición de datos sensibles.
        - *A03:2021 - Inyección de Código*: Detectado en el 94% de las aplicaciones analizadas.
        - *A04:2021 - Diseño Inseguro*: Diversos problemas estructurales que pueden resultar en riesgos de seguridad
        - *A05:2021 - Configuración de seguridad incorrecta*: Problemas de seguridad que podrían compormeter el 90% de las aplicaciones analizadas
        - *A06:2021 - Utilización de componentes obsoletos*: Al utilizar librerías de terceros se deben monizar los problemas de seguridad que puedan ir apareciendo en estas.
        - *A07:2021 - Problemas de identificación y autenticación*
        - *A08:2021 - Problemas de integridad de la aplicación o de los datos*
        - *A09:2021 - Problemas en el log y la monitorización*
        - *A10:2021 - Falsificación de peticiones realizadas por el servidor*: Se produce cuando

la aplicación realiza peticiones al exterior con URLs proporcionadas por el usuario.

2.2 Vulnerabilidades en el tratamiento de los datos de entrada

Una vulnerabilidad es una debilidad en alguno de los componetnes de un sistema informático que puede ser explotada por un atacante para causar algún tipo de daño. La debilidad más común suele ser el no validar losd atos de entrada provenientes del usuario o del entorno.

2.2.1 Inyección de código

Las vulnerabildiades de inyección de código utilizan como datos de entrada determinadas palabras o tokens especiales usados en el lenguaje en el eque están programadas. Cuando estos datos de entrada no se validan para tratar las palabras especiales, una entrada de un usuario puede cambiar la semántica del mensaje original, causando daños.

2.2.1.1 Inyección SQL

Se produce cuando:

- Los datos de entrada de un usuario se utilizan para componer una consulta SQL
- Los datos incluyen palabras especiales SQL como comillas o puntos y comas
- El uso de estas palabras especiales puede cambiar la semántica de la consulta de forma que se ejecute algo diferente contra la base de datos.
- Permite ejecutar consultas de borrado o modificación

Un ataque de inyección SQL puede resultar en revelación de información o borrado de datos importantes. Un ejemplo de inyección SQL sería:

- Tenemos las siguientes casillas:
  - Email
  - Password
- Normalmente estas casillas recibirían un email y la contraseña y se realizaría una consulta SQL como la siguiente:

```
String consulta = "SELECT * FROM users WHRE email='" + email + "' AND
password='" + password + "'";
```

- Si alguien introdujera lo siguiente en la casilla de email:
  - Email: email@email.com' OR '1' = '1' AND password='any'
- Se realizaría un ataque de inyección SQL ya que la consulta se vería modificada y ahora sería de la siguiente forma:

```
String consulta = "SELECT * FROM users where email='email@email.com' OR '1'
= '1' AND password='any'"
```

- Esta nueva consulta permitirá acceder directamente, pasando por encima de los sistemas de

autenticación.

Otras formas de realizar ataques de inyección SQL serían las siguientes:

- Otras técnicas de inyección SQL intentan explotar características o sintaxis específica de algún gestor de bases de datos concreot, identificando qué gestor de BBDD se usa para almacenar la información mediante el análisis de mensajes de error de dicho gestor, a esto se le llama **Database Fingerprint**. Esto se puede prevenir ocultando los mensajes de error para que no sean visibles.
- Blind SQL Inyection: Tipo de inyección en la que el atacante genera consultas booleanas para descubrir información de forma progresiva, se suele combinar con ataques de fuerza bruta de diccionario. A veces se puede usar en aplicaciones que no muestran mensajes de error.

Se pueden **prevenir inyecciones SQL** de las siguientes formas:

- **Prepared Statements**: Se usan consultas parametrizadas, lo que evita la construcción de una consulta a través de concatenación de cadenas de texto
  - Se encargan de formatear los datos de entrada para generar consultas SQL válidas
  - Se realiza espacado de las palabras y caracteres reservados de SQL
  - En Java se usa JDBC para formatear los parámetros de los datos y espacar los datos que provienen del usuario.
  - Permieten establecer parámetros cuando se trata de valores de columnas
  - No permiten establecer parámetros de consulta para nombres de tablas y columnas.

```
String query = "SELECT email, password, full_name FROM users WHERE email =
?"; #La ? representa los parámetros y se pueden establecer con métodos como
setString, setDate....
PreparedStatement stmt = connection.prepareStatement(query);
stmt.setString(1,email);
```

- **Listas blancas**: Se recomienda su uso si no se pueden usar Prepared Statements. Se acota un conjunto de valores válidos, de forma que el sistema rechaza cualquier dato de entrada no incluído en dicho conjunto.
- **Escapado Manual**: Si no queda más opciones, se puede realizar un espacado de forma manual de los datos de usuario.

La inyección SQL tiene las siguientes referencias:

- CWE-89: Improper neutralization of Special Elements used in a SQL Command
- CWE-564: SQL Injection: Hibernate
- CAPEC-66: SQL Inyection
- CAPEC-7: Blind SQL Inyection
- CAPEC-110: SQL Inyection through SOAP Parameter Tampering

2.2.1.2 Inyección en ficheros de log

Los ficheros de log permiten consutrir un historial de todos los eventos que ocurren durante la ejecución de un programa. Muchas veces se pueden usar para reconsutrir el escenario si se a producido algún problema. La inyección en ficheros de logn (Log inyection o Log Forgery) se produce cuando en el Log aparecen mensajes generados a raíz de entradas de usuario que no han sido correctamente validadas o escapadas.

Este tipo de ataque puede tener la siguientes consecuencias:

- Falsificación de los regisotros del log para enmascarar otros ataques, dificultando su localización
- Inyección de códugo ejecutable apra explotar capacidades del software de visualziación de logs.

Para **prevenir la inyección en ficheros de log** se puede hacer lo siguiente:

- Procesar y escapar todas las entradas del usuario
- Utiliza las capacidades de la librería que genera los ficheros de log
    - log4j en java para gestionar mensajes de log enviados desde la aplicación
        - A la hora de especifica el formato de un mensaje esposible escapar caracteres con la función “enc”.

Referencias:

- CWE-117: Improper output neutralization for logs
- CAPEC-93: Log injection-Tampering-Forging
- CAPEC-81: Web Logs Tampering

2.2.1.3 Inyección en cabeceras HTTP

Se produce cuando se usan entradas de usuario incorrectamente escapadas para añadir cabeceras HTTP de forma dinámica e inesperada.

- **HTTP Response Splitting**: El ataque más conocido es la inyección de saltos de línea para partir la cabecera para insertar contenido adicional

Estos ataques se pueden prevenir de la siguiente forma:

- Validar las entradas de usuario mediante una lista blanca
- Codificar las entradas de usuario para escapar los caracteres que puedan resultar problemáticos

Referencias:

- CWE-113: Improper Neutralization of CRLF Sequences in HTTP Headers
- CWE-644: Imporper neutralization of HTTP headers for scripting sysntax
- CAPEC-34: HTTP Response Splitting
- CAPEC-86: XSS Through HTTP Headers

2.2.1.4 Inyección en SMTP

Se introduce una inyección en un pensaje SMTP cuando un atacante inserta cabeceras adicionales dentro de un correo electrónico mediante entradas de usuario. Este tipo de ataque se pueden prevenir de la siguiente forma:

- Listas blancas de valroes validos
- Escapado de caracteres reservados de SMTP.

Referencias:

- CWE-150: Improper Neutralization of Escape, Meta, or Control Sequences
- CAPEC-134: Email Injection
- CAPEC-41: Using Meta-Characters in email headers to inject malicious payloads

2.2.1.5 Inyección de comandos del sistema operativo

Permite la ejecución arbitraria de comandos en el sistema en el que se ejecuta la aplicación. Para prevenir este tipo de ataque:

- Se debe evitar realizar llamadas al sistema operativo desde la aplicación
- Validar bien losd atos de entrada mediante escapado o listas blancas.

Referencias:

- CWE-78: Improper Neutralization of Special Elements Used un a OS Command
- CAPEC-88: OS command injection
- CAPEC-248: Command Inyection
- CAPEC-15: Command Delimiters

2.2.1.6 Inyección en LDAP

De forma similar a la inyección SQL, se pueden realziar inyecciones mediante el uso de cartacteres especailes en las entradas de usuario. Para prevenir esto se recomeinda:

- Realizar Espacado
- Usar Listas blancas
- Limitar los permisos de los usuarios de LDAP

Referencias:

- CWE-90: Improper neutralization of special elements used un an LDAP Query
- CWE-943: Improper Neutralization of special elements y DAta Quering Logic
- CAPEC-136: LDAP injection

2.2.1.7 Inyección en XML y Xpath

Se produce cuando los datos de entrada contienen caracteres reservados de XML, de forma que se genera un documento no esperado. Se puede prevenir este tipo de inyección de la siguiente forma:

- Realizar validación de los datos de entrada de usuarios y bloquear los siguientes caracteres:
  - “ → "
  - ' → '
  - < → <
  - > → >
  - & → &
- Las reglas de escapado varían en función de donde se añada el contenido:
  - En texto dentro de una etiqueta se recomienda escapar los 5 caracteres especiales

    aunque la comilla doble, comilla simple y carácter mayor no sería estrictamente necesario
    - En los valroes de atributo se recomienda escapar los 5 caracteres, pero el caracter de mayor no sería necesario escaparlo.
    - En los comentarios no es necesario escapado
    - Dentro de un CDATA no se debe realizar escapado
    - Dentro de las instrucciones de procesamiento tampoco es necesario hacer escapado.
- Uso de XML Schemas o DTD (Ojo que pueden ser vulnerables a XEE, inyección de vulnerabilidades externas)
- Algunas APIs permieten establecer variables similares a las sentencias parametrizadas de java
    - javax.xml.namespace.QName;
    - javax.xml.xpath.XPathVariableResolver;

Referencias:

- CWE-91: XML injection
- CWE-643: Improper neutralization of Data Within XPath Expressions
- CAPEC-250: XML injection
- CAPEC-83: XPath injection

2.2.1.8 Conlusiones de la inyección de código

- Siempre que existan metodos para formatear la entradad de datos con métodos estándar como las sentencias parametrizadas, se deben usar.
- Se deben usar listas blancas
- En caso de no haber otras opciones se debe realizar Escapado
- Algunos lenguajes traen librerías para realizar el escapado de forma automatizada.

2.2.2 Inyección de Javascript

Cross-Site Scripting (XSS) es un tipo de ataque de inyección de código que inyecta código javascript en el navegador del usuario cuando está accediendo al sitio web afectado. Las consecuencias pueden ser:

- Sustracción de información
- Sustracción de credenciales
- Secuestro de la sesión e implantación de identidad

2.2.2.1 Tipo 1: Reflected XSS

El servidor lee los datos de la petición y los inserta sin validar en la respuesta, estos datos pueden contener código ejecutable. El servidor incluye en la respuesta HTTP el código JavaScript que el atacante ha insertado en la petición. Se añade el contenido JS a la URL, ejecutando el código en el navegador del usuario. Un ejemplo sería:

```
http://patata.com/<script>alert("Ejemplo de Reflected XSS")</script>
```

2.2.2.2 Tipo 2: Stored XSS

El atacante consigue insertar texto javascript en la BBDD y pueden generar contenido HTML. Estos ataques pueden ser muy peligrosos en webs colaborativas como foros o redes sociales.

2.2.2.3 Tipo 0: DOM-Based XSS

La inyección se realiza en el navegador.

2.2.2.4 XFS y ClickJackign

Cross-Frame Scripting (XFS) es un ataque que combina la inyección de JS con el uso de Iframes. A través del iframe el atacante carga la web legítima con el objetivo de robar info del usuario. El clickjacking es un ataque que consiste en superponer un inframe transparente sobre la página que esta viendo el usuario.

2.2.2.5 XSHM

Cross Site History Manipulation (XSHM) es un ataque que se basa en el hecho de que el historial del navegador contiene entradas a las webs visitadas por el usuario. Este historial se puede accewder mediante JS con el objeto history.

2.2.2.6 Prevención de la inyección de javascript

- Siempre que sea posible se deben usar librerías para realziar los escapados
- Regla 0: Evitar siempre que sea posible la inserción de datos inseguros con la regla denyall
- Regla 1: Escapar cualquier contenido que se inserte dentro de un elemento HTML
- Regla 2: Escapar el contenido insertado dentro de los valores de atributos
- Regla 3: Ecapado dentro de etiquetas script o de manejadores de eventos.
- Regla 4: Escapado dentro de hojas de estilo CSS
- Regla 5: Escapado de enlaces a URLs siempre que sea posible.

2.2.2.7 Prevención de XFS

- El servidor web debe enviar la cabecera HTTP X-Frame-Options con uno de los siguientes valores:
    - DENY: No mostrar nunca una web dentro de un iframe
    - SAMEORIGIN: Permitir mostrar una web en un iframe cuando sea del mismo origen
    - ALLOW-FROM <sitio>: Se permite mostrar webs en iframes siempre que vengan del sitio definido.

2.2.3 Content Security Policy (CSP)

Es un mecanismo que permite restringir los contenidos que el navegador puede cargar en un sitio Web. Se centra en detener ataques de inyección de código, centrándose en XSS. Se puede especificar en una cabecera dentro de la respeusta HTTP que el servidor envía al navegador. Los CSP siguen la siguiente sintaxis:

```
Content_security-Policy: <directiva>; <directiva>
```

Existen directivas para restringir:

- El origne de los datos
- El documento
- En la navegación
- Directivas de nontificación de contenido que no cumple la política definida.

2.2.3.1 script-src

Controla de manera exhaustiva el origen de todos los scripts.

```
Content-Security-Policy: scriptsrc <origen>;
Content-Security-Policy: scriptsrc <origen> <origen>;
```

Valores de los source:

- self: Sitio web de origen
- unsafe-evarl: permite el uso de la función eval()
- unsafe-inline: permite ejecutar contenido dentro de los scripts

No se recomienda activar ninguno de los valores usanfe para minimizr los riesgos de ataques de inyuección JS.

2.2.3.2 frame-ancestors

Permite definir cuando es posible incluir un sitio web dentro de un iframe, similar a la cabcera X-FRAME-OPTIONS

```
Content-Security-Policy: frame-ancestors 'none';
Content-Security-Policy: frame-ancestors 'self' http://patata.org;
```

El valor none es equivalente a DENY.

2.2.3.3 report-uri y report-to

Permiten especificar una URL a la que enviar las notificaciones cuando el navegador detecta contenido que no cumple con la política definida.

```
Content-Security-Policy: defautl-src https:; report-uri /csp-violation-
report-endpoint/
```

2.2.4 Inyección de entidades externas en XML (XXE)

Es un ataque que afecta al lenguaje XML que consiste en añadir referencias a elementos externos dentro de un documento XML con el objetivo de causar daño. Se produce un ataque de SSRF (Server Side Request Forgery) cuando un atacante consigue acceder a un servicio interno de una organización incluso cuando este servicio se encuentra protegido por un cortafuegos Para prevenir estos tipos de ataques se pueden usar:

- Parsers de XML: Ofrecen diferentes opciones para deshabilitar total o parcialmente el soporte de entidades externas
- Deshbailitar el soporte de DTDs
- Configurar una clase para resolver referencias externas: XMLUrlResolver

2.2.5 Deserialización y carga dinámica

2.2.5.1 Deserialización

La serialización es un proceso que transforma un objeto de un lenguaje en un flujo de texto o binario. La deserialización es el proceso contrario, el cual transforma un flujo de datos en objetos. Se produce una deserialización insegura cuando no se valida si el flujo de datos de entrada va a crear objetos del tipo esperado. En java para serializar y deserializar un objeto a XML se pueden usar las siguientes clases:

- java.beans.XMLEncoder
- java.beans.XMLDecoder

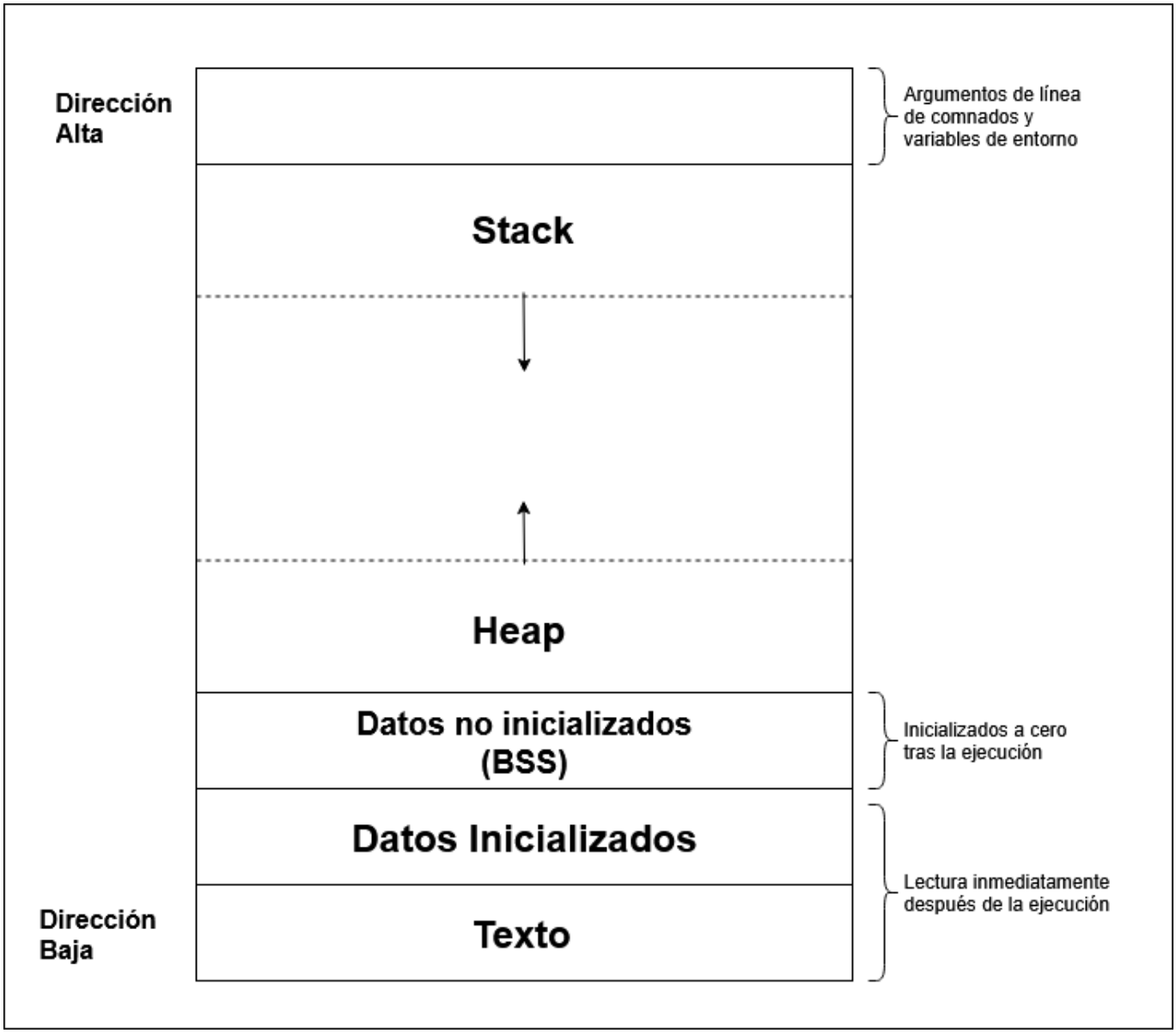
Si no se hacen validaciones, la deserialización puede ser peligrosa ya que podría permitir la ejecución arbitraria de cualquier método de cualquier clase de java. Para mitigar este tipo de ataque se puede hacer lo siguiente:

- Añadir validaciones de integridad en los objetos serializados
- Validar si el tipo de objeto que se va a crear está dentro de una lista blanca.
- Aislar el código que realiza la deserialización para que se ejecute con los minimos permisos necesarios.
- Monitorización de los procesos de deserialización para detectar si un usuario realiza demasiados intentos.

2.2.5.2 Carga dinámica

Una vulnerabilidad similar a la deserialización es la de la carga dinámica segura (Unsafe reflection). El API de reflection de algunos leguajes permite crear objetos a partir de su nombre

2.2.6 Desbordamiento de Pila

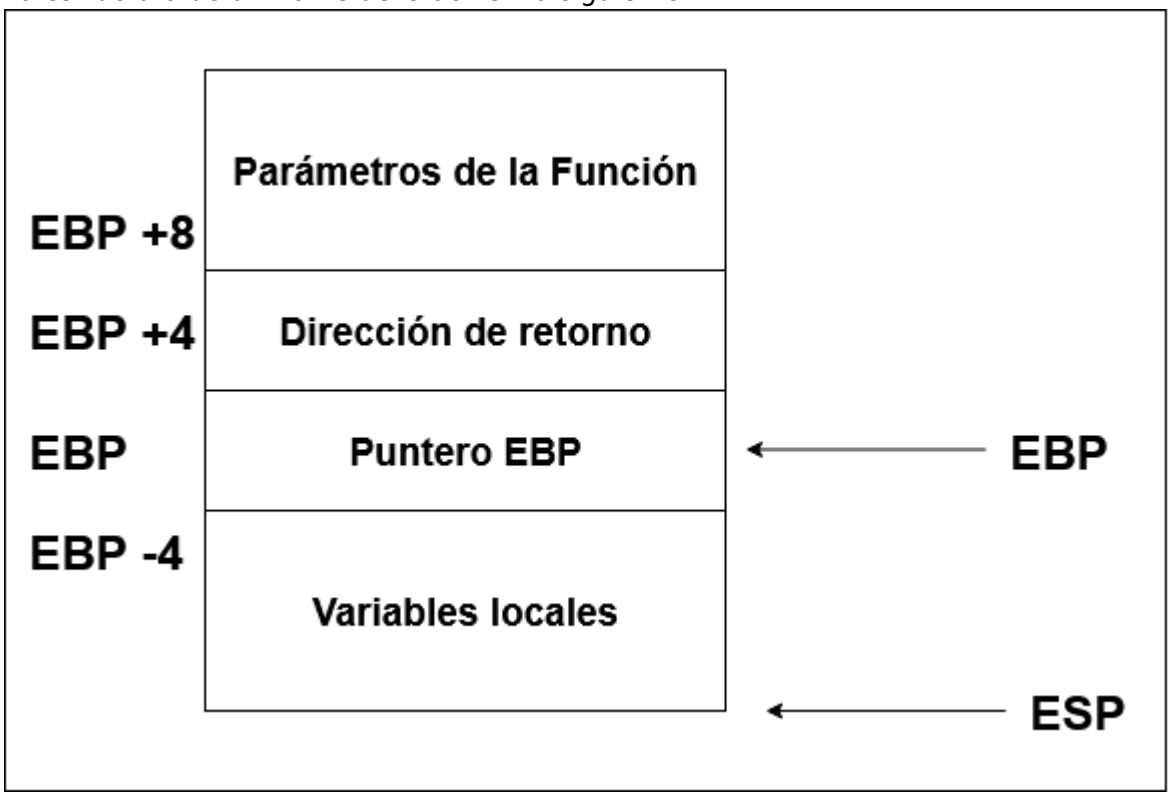


Se produce un desbordamiento de buffer (Buffer overflow o Buffer Overrun) cuando un programa permite la escritura de datos mas allá del buffer asignado. La a estructura de memoria de un programa en C es la siguiente:

- Segmento de Texto: Contiene instrucciones ejecutables
  - Situado debajo del heap y del stack para prevenir ataques de overflow
- Segmento de datos no inicializaods (BSS): Contiene variables globales y estáticas inicializadas a 0 o que no tienen inicialización
- Segmento de datos inicializados: Contiene variables globales y estáticas inicializadas explícitamente
- Segmento de heap: Comienza después del BSS, crece hacia direcciones altas y contiene la memoria dinámica con el almacenamiento a largo plazo.
  - Es común para todas las librerías compartidas y todos los módulos que se cargan de forma dinámica.
  - Se manipula con malloc y free en C y con new y delete en C++
- Segmento de stack: Estructura LIFO (Last In First Out) localizada generalmente en las zonas más altas de memoria.
  - Los elementos que se añaden en la pila cada vez que se llama a una función se llaman stack frames
    - Contienen variables locales y parámetros de una función

    - Contienen la dirección de retorno.
- Puntero EIP: Apunta a la siguiente instrucción que va a ser ejecutada
- Puntero base EBP (base pointer): Apunta al comienzo del stack frame actual
- Puntero de pila ESP (Stack Pointer): Apunta al principio de la pila

La estructura de un frame del stack en la siguiente:



Al comienzo de la llamaca a una función se ejecuta un fragmento de ensamblador denominado secuencia de entrada que:

- Guarda en la pila el puntero al frame actual
- EBP se establece a ESP con lo que pasa a apuntar al inicio de la pila
- Reserva 12 bytes para las variables locales de la función

```
_Funcion:
  push ebp
  mov ebp, esp
  sub esp, 12
```

Si la función se invoca con argumentos, los parámetros se añaden al principio del frame con una llamada call equivalente a push + jump.

```
push eip + 2
jmp _funcion2
```

Al terminar la llamada de una función se ejecutan las siguientes secuencias de salida:

- Se mueve el puntero ESP al valor de EBP y se libera el espacio de las variables lcoales
- Se establece el valor EBP al que tenía antes de la llamada
- Devuelve el control a la dirección de retorno.

```
mov esp, ebp
```

```
pop ebp
ret
```

2.2.6.1 Prevención del desbordamiento de pila

- **ASLR (Address Space Layout Radomization)**: Permite distribuir de forma aleatoria los espacios de direcciones de memoria para evitar que un atacante transfiera el control a una dirección de memoria conocida
- **Stack Canaries**: Técnica de detección de stack overflow, añade diferentes valores numéricos a la pila elegidos de forma aleatoria al arrancar el programa, si se modifican estos valores, se detecta un desbordamiento de pila.

2.2.6.2 Ensamblador x86

- **Push** coloca su operando en el principio de la pila

```
push <mem>
```

- **pop** eleimina el elemento situado al principio de la pila y establece su valor en el operando especificado.

```
pop <mem>
```

- **mov** copia el dato referenciado en el segundo operando a la ubicación referenciada en el primer operando.

```
mov <destino> <origen>
```

- **sub** almacena en el primer operando el resultado de restarle a este el valor del segundo

```
sub <mem1> <mem2>
```

- **jmp** transfiere el control del programa a la dirección de memoria indicada

```
jmp <etiqueta>
```

- **ret** obtiene la dirección del stack y devuelve el control a esa dirección

2.2.7 Validación de datos

La principal utilidad de realizar estas validaciones en el cliente son:

- Mejorar la experiencia de usuario
- Minimizar el número de peticiones que se realizan al servidor.

Generalmente el comprobar que los datos proporcionados por el usuario son validops sdebería ahcerse siempre en el servidor. En java hay API de validación estándar formado por las clases del paquete javax.validation. Este API permite establecer anotaciones sobre los atributros delos objetos en los que se reciben los datos de entrada.La especificación 2.0 del API de validación de Java incluye

las siguientes validaciones de serie:

- @Null, @NotNull: Permite validar si un objeto es nulo o no
- @AssertTrue, @AssertFalse: Permite comprobar el valor de un booleano
- @Min, @Max: Permite validar un valor mínimo o máximo
- @Size: Permite validar el tamaño de un objeto
- @Past, @PastOrPresent, @Future y @FutureOrPresent: Validación de fechas
- @Email: Permite valdiar que una dirección de correo electrónico es correcta.

2.3 Vulnerabilidades en la autenticación

Permiten a un atacante obtener las credenciales de los usuarios de la aplicación. Explotan debilidades como las siguientes:

- La aplicación permite contraseñas poco seguras: Pueden ser obtenidas mediante ataques de fuerza bruta de diccionario.
    - Se recomienda política de contraseñas que contenga una longitud mínima, un mínimo de complegidad y valdiar que la cotnraseña no esté dentro del top 1000 Worst Passwords.
- No se limita el número de intentos fallidos de autenticación
- Las contraseñas se almacenan de forma poco segura
- Las contraseñas se transmiten en claro.

Recomendaciones:

- Se recomienda no revelar que parte de la autenticación no es correcta, evitando mensajes que digan que el eMail o la contraseña no son correctos. Con el mensaje de usuario inexistente el atacante puede tratar de tantear cuetntas de usuarios.
- Autenticación multi-factor
    - Se pueden combinar 2 o más tipos de autenticación (Contraseña, biometría, certificados…)
- Almacenar los datos de usuarios en una BBDD cifrada, de forma que si se compromete esta, sea más difícil obtener los datos de los usuarios

2.3.1 Hashes

- Se recomeinza usar la técnica Hash and Salt, que usa una función criptográfica de hash añadiendo ruido de por medio.
- Como los hashes son irreversibles, son más recomendados que el típico cifrado.
- Los cifrados más seguros son bcrypt, scrypt y PBKDF2, mientras que los más inseguros son MD5 y SHA1.
- A la hora de procesar contraseñas se busca que la función hash sea lo más lenta posible para que el atacante pueda realizar emenos ataques por unidad de tiempo.
- Las técnicas de hash básicas son vulnerables a ataques de diccionario Rainbow Tables
    - Diccionarios con códigos hash junto con las palabras que los han generado.
    - Se mitiga añadiendo fragmentos aleatorios (salt) que se concatenan a la contraseña para generar un hash más difícil de predecir.

2.3.2 PBKDF2

Algoritmo hash muy utilizado, permite especificar un salta y un número de interaciones que se realizarán para calcular el hash final.Se suele combinar con HMAC-SHA que es una forma especial de usar SHA.

2.3.3 Bcrypt

Otra de las funciones hash más utilizadas. Diseñado para ralentizar los ataques de fuerza bruta. Incluye un factor de trabajo configurable para establecer el tiempo que tarda el algoritmo en calcular el hash. En la mayoría de sus implementaciones añade Salt aleatorio de forma automática en cada llamada.

2.3.4 Escenarios

Formas de almacenar las contraseñas desde las más inseguras hasta las más seguras:

- Almacenar contraseñas en claro: Lo más inseguro, si un atacante logra sustraer la BBDD todas las cuentas queda comprometidas.
- Almacenar contraseñas con SHA o MD5: Algoritmos rápidos que posibilitan realziar muchos intentos en poco tiempo, altamente vulnerables al ataque Rainbow Tables.
- Hash and Salt: Se concatena a la contraseña una serie de bytes fijos antes de aplicar la función hash para dificultar los ataques de fuerza bruta. Si el atacante logra sustraer la BBDD es posible que también logre extraer el salt y aplicar el ataque de rainbow tables igualmente
- Utilizar un salt diferente para cada usuario. El atacante no puede obtener todas las cuentas fácilmente, tendrñia que generar un diccionaro de hash diferente para cada usuario.
- Usar bcrypt con un factor de trabajo relativamente alto para contrarrestar ataques de fuerza bruta.

2.3.5 Transmisión

- La información de autenticación no debe ser transmitida a través de un medio inseguro.
- En muchos casos se usa una capa intermedia de cifrado
- Se recomeinda uisar HTTPS en vez de http para la transmisión de información confidencial, lo que añade una capa de cifrado por encima del protocolo TCP

2.4 Vulnerabilidades en el manejo de la sesión

Consisten en el robo de sesiones una vez que el usuario ya esté autenticado.

2.4.1 Secuestro de la sesión

- Por medio de un secuestro de sesión (Session hijacking) un atacante logra obtener una coockie de sesión válida de un usuario, pudiendo ejecutar acciones a su nombre.
    - La cookie de sesión se puede obtener explotando las siguientes vulnerabilidades:
        - Transmisión de cookie en claro
        - Filtración de la sesión
        - Inyección de JavaScript
    - Para mitigar estos ataque se recomienda:
        - No transimitr cookies por medios inseguros, usar HTTPS
        - No permitir el acceso desde JavaScript a las cookies de sesión
        - Implementar mecanismos de caducidad de sesión

2.4.2 Fijación de la sesión

Session fixation es una variante del secuestro de sesión en la que el atacante consigue que la víctima se autentique usando un identificador de sesión generado por el atacante.

1. El atacante accede a la pagina de autenticación
2. El servidor crea un ID de sesión y se lo asigna a esa sesión
3. El atacante envía a la víctima un enlace que de alguna manera incluya una URL con el identificador de sesión del atacante
4. La víctima utiliza el enlace para autenticarse, enviando el identificador de sesión que le envió el atacante
5. la víctima proporciona sus credenciales y pasa a ser una sesión autenticada.
6. El atacante conoce el identificador de la víctima y lo puede usar para realizar acciones en su nombre.

Existen otros métodos más sofisticados:

- Enviar el identificador de sesión en un campo oculto de usuario
- Enviar el identificador de sesión a través de una cookie

Para prevenir este tipo de ataques se recomienda generar una cookie de sesión nueva cada vez que el usuario se autentica, de esta forma el identificador de sesión que conoce el atacante deja de ser válido.

2.4.3 Reescritura de URl

Muchos servidores envían el identificador de sesión en la URL, esto propicia que el identificador de sesión sea altamente vulnerable a distintos tipos de sustracción. La cabecera HTTP refer contiene información sobre la web desde la que se origina la petición, por lo tanto, si el atacante consigue redirigir el navegador de la víctima hasta una web que el controla, pude capturar la URL de origen con la cabecera en cuestión. Si la cabecera es el identificador de sesión el atacante lo puede usar para realizar peticiones en nombre de la víctima.

2.4.4 Política de mismo origen

La Same-origin Policy define una serie de reglas para restringir como un documento o sus scripts pueden interacturar con otros recursos localizados en dominios diferentes. Esta política se implementa en el navegador web. Dos webs tienen el mismo dominio si el protocolo, nombre de dominio y puerto son los mismos.

- Los navegadores proporcionan acceso limitado al documento y a la ventanad e otros dominios
- La política de mismo origen también define qé interacciones están permitidas entre objetos JS
    - Window.parent
    - window.opener
    - iframe.content.window
- Para las cominicaciones entre documetnos de diferentes origenes se puede usar window.postMessage
- Tambien impone restricciones sobre cabeceras HTTP
- Solo se pueden realizar peticiones AJAX al mismo dominio por defecto
- El acceso al almacenamiento de info (localStorage y SessionStorage) está separado por dominios.
- Las cookies también están separadas por dominos y una página solo puede establecer una cookie en su dominio.
- No se pueden realizar peticiones HTTP a otros dominios desde JS con fetch o con el objeto XmlHttpRequest
- JSONP (JSON con Padding) permite realizar llamadas asíncronas a dominios diferente

2.4.5 CORS

El intercambio de recursos de origenes cruzados permite al navegador solicitar recursos de dominios diferentes. Para permitir al navegador solicitar recursos a troso domios, el servidor web del dominio destino envía ciertas cabeceras http adicionales para aceptar o denegar la solicitud.

2.4.5.1 CORS en peticiones simples

Son peticiones simples aquellas cuyo método HTTP es HEAD, GET o POST con solicitudes con cabeceras como Accept, Accept-Language, Content-Type… El navegador enviará la cabecera Origin en la petición. la respuesta del servidor envía varias cabeceras con el prefijo Access-Control:

- Access-Control-Allow-Origin: Indica si están permitidas las peticiones desde el origen. Si la respuesta no incluye esta cabecera, la petición CORS fallará
- Access-Control-Expose-Headers: El navegador expondrá más cabeceras a tra´ves del método getResponseHeader()
- Access-Control-Allow-Credentials: Indica si el navegador debe exponer o no las credenciales

2.4.5.2 CORS en peticiones complejas

Se consideran complejas aquellas que no son GET, HEAD o POST, como PUT o DELETE y aquellas que no usen cabeceras simples. Estas solicitudes hacen una solicitud inicial de verificación a través de método OPTIONS con las siguientes cabeceras:

- Access-Control-Request-Method: Indica el método que se utiliza en la petición entre dominios.
- Access-Control-Request-Headers: Indica la lista de cabeceras no estándar que se utilziarán en la petición

La respuesta que envía el servidor puede incluir las siguientes cabeceras:

- Access-Control-Allow-Origin: Indica si están permitidas las peticiones desde el dominio origen. Esposible especifciar el valor * para permitier que cualquier dominio pueda realziar peticiones. Si esta cabecera falta, CORS fallará.
- Access-Control-Allow-Methods: Lista de métodos que se permiten en la petición CORS
- Access-Control-Allow-Headers: Lista de las cabeceras no estádnar que se permiten

2.4.6 CSRF

Cross Site Request Forgery es una vulnerabilidad en la que el atacante usa el navegador de la víctima para ejecutar una petición maliciosa en su nombre sobre una web vulnerable. Al usarse el navegador de la víctima también se añade la cookie de esta, por lo que no es posible diferenciar si la víctima ejecuta la acción voluntariamente o no. Esta vulnerabilidad explota la confianza que tiene un sitio web en sus usuarios.

2.5 Exposición de Información Sensible

Es una vulnerabilidad transversal relacionada con las otras vulnerabilidades. Abarca un amplio espectro desde el robo de credenciales y ataques man in the middle hasta cualquier otro tipo de caputra de información sin cifrar que pase por la red. Para obtener información sensible del sistema o del usuario se pueden usar las siguientes vulnerabilidades:

- Envío de información a través de comunicaciones sin cifrar
- Uso de algoritmos de cifrado débiles
- Información expuesta por la propia aplciación.
    - Trazas de error de java
    - Mensajes de Error SQL
    - Mensajes de Error PHP
    - Se puede exponer:
        - Info de librerías o frameworks
        - Info del sistema operativo
        - Info sobre el sistema de ficheros
        - Info sobre el código fuente de la app

2.5.1 Man in The Middle

El atacante es capaz de itnerferir en las comunicaciones entre un cliente y un servidor, capturando los mensajes entre ambos interlocutores e incluso inyectando mensajes modificados en el canal de modificación. Con este tipo de ataque es posible ejecutar session hijacking. También puede utilizar:

- Sniffing: Captura de tráfico de red
- Sidejacking: Capturar paquetes de datos con el objetivo de obtener cookies de sesión

- Evil Twin: Se crea una red con el mismo nombre que una legítimas para conseguir que el usuario se conecte a esta.

2.6 Vulnerabilidades en el control de acceso

Las directivas de control de acceso tienen como objetivo garantizar que los usuarios solo puedan usar las funcionalidades de las que tenga permiso. Las vulnerabilidades de este tipo suelen ser:

- Acceso a cuentas sin autenticación
- Acceso a funciones de admin desde usuarios sin permisos
- Acceso a recursos no permitidos

El controlde acceso debe seguir el principio de menor privilegio (POLP), que consiste en limitar los permisos de acceso a un usuario a los mínimos posibles para que pueda usar la aplicación. Cuando no existe control de acceso, el atacante puede acceder a algunas zonas de la aplicación a las que no debería.

2.6.1 Prevención

- La política por defecto debe ser de denegación de acceso
- Los controles de acceso deben implementarse en el servidor
- Se debe imponer la propiedad de dueño para que un usuario solo pueda acceder a sus registros
- Se deben incluir pruebas de control de acceso

2.6.2 Directory Traversal

Es una vulnerabilidad que permite a un atacante acceder a ficheros a los que no debería tener acceso. Se explota a través de la concatenación de cadenas y del uso del string "../" para conseguir acceder fuera del directorio raíz de la web. Estos ataques pueden ser mitigados de la misma forma que los ataques de inyección de código.

2.7 configuración de seguridad incorrecta

Puede lelgar a ser una vulnerabildiad muy importante que peude afectar a cualquier elemento. Destacan los siguientes elementos susceptibles:

- Funcionalidades innecesarias habilitadas
- Cuentas por defecto habiltadas
- Páginas de error por defecto
- Control de acceso que permite llamadas usando métodos HTTP no controlados

También se pueden incluir diferentes parámetros relacionados con el control de acceso:

- Limitar el número de sesiones por usuario
- Cerrar sesión tras período de inactividad
- Bloquear sesión tras cierto tiempo de uso

- Limitar el número de registros que puede devolver una consulta
- Separar las aplicaciones de usuarios de las de gestión.

2.8 Monitorización y Log insuficientes

- la falta de registros de log puede llevar a que no exista información sobre un ataque, dificultando la detección y corrección del problema
- Del mismo modo, los logs deben mostrar información clara y concisa, si el log es excesivo y/o poco claro, también resulta difícil estraer info útil.
- Lamacenar el log solo de forma local puede ser peligroso si el atacante se hace con el control de la máquina.
- Los eventos de inicio de sesión se deben guardar en el log.
- Los eventos de control acceso también deben alcacenarse en el log, sobre todo los sospechosos.
- Las transacciones relevantes también se deben guardar en el log.

TEMA 3. Ciclos de desarrollo de software seguro

para minimizar los riesgos de que se produzcan vulnerabildiades en el software se deben abordar los aspectos de seguridad en las primeras fases del ciclo de vida de desarrollo. El coste de encontrar vulnerabilidades en las primeras fases es muy inferiror al coste de detectarlas más tarde. El coste de un problema de seguridad puede llegar a multiplciarse por 25 desde la fase de diseño.

3.1 Secure Development Lifecycle (SDL)

Estandariza un conjunto de buenas prácticas para aplicar durante el desarrollo de una aplicación.

- Se define una fase previa al inicio de proyecto para el training
- Para cada fase del ciclo de desarrollo se define una serie de tareas
- Define contramediadas que se depen poner en marcha cuando se porduce alguna incidencia de seguridad.

3.1.1 Fase de Formación

Se debe impartir a Desarrolladores, Equipos de pruebas y jefes de proyecto. Esta formación debe ser continuada y periódica para que todos los involucrados en el proyecto estén al día de los cambios en los procedimientos.

- Principios de diseño seguro: reducción de la superficie de ataque, principio de menor privilegio…
- Modelado de amenazas (Threat Modeling): Proceso de identificar todas las posibles amenazas que pueden afectar a una aplicación

- Codificación de forma segura: Gestión de buffers, uso de cifrado…
- Pruebnas: Diferencia enrtre pruebas funcionales y pruebas de vulnerabilidades, evaluación de riesgos, pruebas de seguridad…
- Privacidad: identifciación de los diferentes tipos de datos confidenciales.

3.1.2 Fase de Análisis

Definición de los requisitos de seguridad

- Asignación de expertos
- cofiguración del sistema de seguimiento…

Definición de umbrales de calidad que se espera obtener Modelado de casos de uso o historias de usuario relacionadas con la seguridad desde el punto de vista de un atacante Evaluación de riesgos identificando los elementos funcionales más críticos.

3.2.3 Fase de diseño

- Establecimiento de los requerimientos de diseño como la privacidad de los datos y la criptografía
- Análisis de las superficies de ataque.
- Modelado de las amenazas (Threats modeling): identificación de riesgos y definición de las mediads para contrarrestarlos.

3.2.4 Fase de implemetnación

- Se define un documento de buenas prácticas de implementación relacionado con ascpectos de seguridad
- Se determina que herramientas se utilizarán para analizar el código y para la ejecución de pruebas automáticas.
- Ejecución de análisis estáticos periódicos utilizando herramietnas SAST (Static Application Security Testing)
    - Diseñadas para analizar el código fuente o compilado para encontrar problemas de seguridad.
    - Útiles para detectar Inyeccion SQL y Desbordamiento de Buffer
    - Salida precisa y útil.
    - El problema es que no son capaces de detectar vulnerabildiades como problemas de unteticación, problemas de control de acceso…
    - Pueden dar falsos positivos
    - No detecta errores de configuración.
    - Ejemplos de Herramientas SAST:
        - SonarQube
        - Find Security Bug

3.2.5 Fase de Pruebas

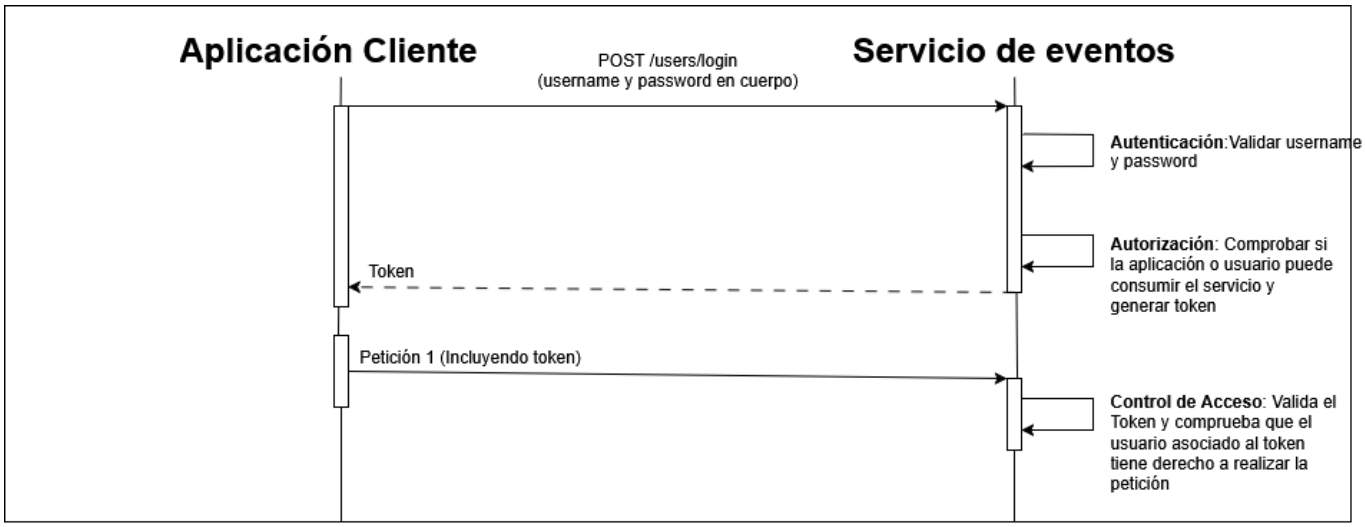
- revisión de código por parte de otros miembros del equipo

- Realización de pruebas de caja negra
- Realización de análisis dinámico y fuzz testing con herramientas DAST (Dynamic Aplication Security Testing). Se introducen en la aplicación datos iválidas, inesperados o aleatorios
- Reevaluación de las superficies de ataque

3.2.6 Fase de pentesting

Son pruebas que se realizan sobre el software simulando ataques que podría realizar un hacker. Pueden ser de caja blanca (El auditor tiene toda la info sobre el sistema) o de Caja Negra (El auditor va a ciegas)

Tema 4: Mecanismos de autenticación, Autorización y control de acceso



- **Autenticación**: Muchas de las funcionalidades que expone el servicio requieren que el usuario de la aplicación cliente se autentique.
  - una forma trivial de tratar la autenticación sería que el servicio ofreceiese un punto de acceso para que el usuario se autenticase
    - Recibe Username y Password como parámetros
    - Comprueba que las contraseñas sean correctas.
- **Autorización**: El punto de acceso de autenticación puede devolver un token que autoriza a la aplicación cliente a hacer peticiones al servicio en nombre del usuario.
  - La aplicación cliente enviará dicho token como parte de cada petición que realice, pudiendose conocer la identidad del usuario y su info a través del token.
  - El token debe ser seguro, un atacante no debe poder generar un token arbitrario válido para una aplicación que consume un servicio.
- **Control de acceso**: Cuando el servicio recibe una petición, su procesamiento no debe estar disponible para cualquier usuario.
  - El control de acceso comprueba que la petición se puede ejecutar con el token de acceso.
    - Valida el token de acceso
      - Si la validación es correcta, la aplicación puede consumir el servicio
    - Validar que al usuario se le permite la acción

4.1 Autenticación y Autorización

4.1.1 Autenticación en HTTP

HTTP proporciona cabeceras estándar como Authorization para una petición y WWW-Autehnticate para la respuesta que pueden ser usadas para autenticación y autorización.

- **Basic**: Cuando el servicio recibe la petición solo tiene que decodificar la cadena que sigue a Basic para obtener el username y la password.
    - OJO: estas peticiones se deben mandar en HTTPS.
    - Si el usuario o la contraseña son incorrectos el servicio responde error 401 incluyendo la cabecera WWW-Authenticate
    - El problema de este tipo de autenticación es que es ineficiente ya que hay que validar el usuario y contraseña cada vez que se hace una petición y recuperar sus roles de usuario.
- **Digest**: Ofrece un esquema más complejo que permite no enviar la contraseña en cada petición, aunque no es muy usado ya que también es bastante ineficiente

4.1.2 JSON Web Token (JWT)

Define un formato compacto para obtener info JSON de forma segura entre dos partes.

- Base64URL: la cabecera y el cuerpo pueden contener caracteres no ASCII
    - Los algoritmos de firma generan datos binarios
    - Para que JWT se pueda enviar en protocolos que envían texto ASCII, JWT usa un sistema de codificación que permita codificar los textos binarios en ASCII
- Tipos de campos (Claims) en objetos JSON.
    - Registrados: Están estandarizados pero no son obligatorios
    - Públicos: Los puede definir cualquiera aunque se pueden usar proveedores definidos por la IANA para garantizar la compatibilidad
    - Privados: Los puede definir cualquiera. Los campos se pueden utilizar en contextos locales.
- Firma: Un JWT se puede firmar con un algoritmo de criptografía
    - Con un algoritmo de criptografía simétrica la firma esun MAC, garantizando la integridad y Autenticidad
    - Con un algoritmo de cirptografía simétrica es una firma digital, lo que garantiza el no repudio
    - RFC 7515: JSON Web Signature (JSW)
- JWTs Cifrados: Si los datos enviados en un JWT son confidenciales, se deben cifrar con RFC 7516 (JSON Web Encription “JWE”)
    - El formato de los tokens cifrados consta de 5 partes separadas por ”.“ y en base64url

4.1.3 OAuth

Problemas que suelen encontrarse:

- Problema 1:
    - Como obtener el token de acceso para un servicio de otro proveedor

    - La aplicación necesita 2 tokens de acceso, uno para el backend y otro para el servicio
    - Para obtener el segundo token no sería seguro que el servicio ofreciese una operación similar a la del backend de la aplicación
        - El usuario tendría que teclear su usuario y contraseña en un proveedor de servicio externo
        - La aplicación prodría quedarse con ele identificador y la contraseña del usuario.
- Problema 2:
    - Una empresa dispone de mútiples aplicaciones y servicios corporativos.
        - Sería necesario implementar un servicio para que las aplicaciones pudieran obtener tokens de acceso para acceder a los servicios que necesitan y una aplicación de gestión de aplicaciones cliente y tokens.
    - Si no se hace nada especial, cada aplicación cliente tendría que implementar su propio formulario de autenticación.

OAuth surge para dar solución al problema y en la práctica también resuelve el problema 2. En la actualidad está en su versión 2.0 que no es compatible con la 1.0 ya que usan protocolos distintos.

- Con el tiempo surgieron RFCs adicionales que complementan a OAuth 2.0 como PKCE, que se usa en aplicaciones web SPA y en aplicaciones nativas.
- Usando las herramientras adminsitrativas de la implementación de OAuth se debe registrar cada aplicación cliente
    - Datos de entrada
        - Info básica
        - URL de redirección
    - Datos de Salida
        - client_id: identificador de la app
        - client_secret: clave secreta de la app
- Cliente confidencial:
    - Aquellos que pueden guardar de forma segura el client_secret
- Clientes públicos
    - Aquellos que no pueden guardar de forma segura el client_secret
    - Durante el registro no se genera el client_secret.

4.1.3.1 Flujos

Para soportar distintos escenaros que difieren en el tipo de app cliente o si son o no del mismo proveedor Oauth contempla varios flujos, aunque nos centraremos en el Authorization Code, el más relevante.

- response_type: indica el tipo de flujo
- redirect_uri: El servidor de autorización, por seguridad, comprueba que el valor del redirect_uri corresponda con una de las url de redirección especificadas durante el registro del client_id
- scope: Especifica los permisos que solicita la aplicación cliente
- El formulario de autenticación puede permitir Single Sing On (SSO) con una cookie persistente para no tener que volver a autenticar el usuario en el server de autenticacion.
- code: Código de autorización que llega en la redirección y que el cliente tendrá que usar posteriormente para solicitar el token, tiempo de vida muy corto inferior a 10 min
- El servidor de autorización comprobará que el código de autorización no está caducado y que había sido generado para ese client_id y redirection_uri.
- La app guarda el token de acceso en la sesión.

- Si el cliente es púlico no se pasa la cabecera Authorization y se incluye el parámetro Client_id en el cuerpo.
- Token de Acceso (Access_token):
    - Ristra de caracteres
    - Token que usará la APP cliente para acceder al servicio
    - Expira al poco tiempo
    - La especificación no indica un formato concreto
    - Puede ser no autocodificado, una ristra de caracteres no parseables. Puede usar el endpoint /introspect del server de autorización
    - Puede ser autocodificado, una ristra de caracteres parseables. Puede ser verificado autónomamente.
        - RFC 9068: Especifica un token en formato JWT
- Token de refresco (refresh_token):
    - La aplicación cliente lo puede usar para solicitar un nuevo token de acceso al servidor de autorización cuando el token de acceso actual expira evitando volver a iniciar el flujo.
        - Se puede hacer por anticipado o cuando recibe una respuesta de error indicando que el token actual caduco.
    - Tiempo de vida largo
    - Se emite solo a clientes confidenciales
    - Para solicitar un nuevo token de acceso a partir de un token de refresco se usa el endpoint /token.
- Verificación de un token de acceso autocodificado
    - para que el servidor de recursos pueda validar la firma, el token debería estar firmado con un algoritmo de criptografía simétrica.
        - El server de autorización firma los tokens con clave privada
        - Los servidores de recursos validan los tokens con clave pública.
        - Si se usa un algoritmo de cirptografía simétrica, la unica clave tendrá que compartirse entre los servidores de autorización y de recursos.
- kid
    - identifica de forma única la clave pública dentro de un JWK Set
    - la cabecera de cada token generado por el servidor de autorización incluye el campo kid con el valor que identifica a la clave púlica que permite verificar la firma del token.
- El servidor de recursos puede verificar autónomamente los tokens que recibe en su petición.
    - Recupera claves públicas del servidor de autorizacón GET /jawks y cachea el resultado
    - Cada vez que recibe una petición
        - recupera de caché la clave pública correspondiente al kid del token que viene en la petición
        - Verifica la firma del token

4.1.3.2 PKCE

El flujo antes descrito está sujeto a dos tipos de vulnerabilidades:

- Vulnerabilidad 1 CSRF:
    - El atacante inicia el flujo
    - Lo para cuando se devuelve la redirección con el código de autorización
    - Obtiene un código de autorización válido.
    - Envía a la víctima el enlace que se le devolvió de la redirección
    - La victima que asumimos autenticada recibe el correo del atacante
    - Hace click en el enlace del atacante

    - La víctima sin saberlo inicial el flujo desde donde lo dejó el atacante.
    - La aplicación web recibe el código de autorización del atacante y solicita un token de acceso que se almacena en la sesión web
    - Las acciones que haga a partir de ahora la víctima se verán reflejadas en la cuenta del atacante.
- Vulnerabilidad 2:
    - La víctima inicia el flujo y obtiene el código de autorización
    - El atacante usando alguna vulnerabilidad consigue obtener el código de autorización de la víctima.
    - Mientras no expire y no se haya usado dicho código existe una ventana de tiempo en la que el atacante puede obtener el token de acceso a partir del código de autorización de la víctima.
    - Si el cliente es público el atacante puede pasar directamente al endpoint
    - No pasa la cabecera Authorization e incluye el parámetro client_id
    - El atacante puede usar el token de acceso para manejar la aplicación de la víctima.

Oauth dispone del parámetro state para hacer frente a la vulnerabilidad 1 y la técnica PKCE protege contra ambas vulnerabilidades.

- PKCE protege contra la vulnerabilidad 1 por que en la sesión web de la víctima no hay un code_verifier.
- PKCE protege contra la vulnerabilidad 2 por que el atacante necesitaría el code_verifier de la víctima.

4.1.4 OpenID Connect

Es un protocolo de autenticación y autorización consutrido por encima de OAuth, reutilizando los flujos, tokens de acceso y refresco, endpoints Authorization, troken, introspection… Lo que sería todo. OIDC añade las siguientes novedades:

- Nuevo tipo de token (id_token)
    - Es un token en formato JWT que contiene info del usuario
    - En el flujo de codigo de autorización, la respuesta al POST sobre el endpoint TOKEN incluye el id_token
    - ütil cuando la aplicación del cliente quiere obtener inmediatamente info sobre el usuario a la vez que recibe el token de acceso.
- Estandarización de la información acerca de un usuario
- Un flujo adicional
- Más endpoints.

OIDC tiene los siguientes claims y scopes:

- Estandariza la ifnormación que se deuvelve sobre un user dentor del id_token, modelándose como claims JWT
- Independientemente del flujo utilizado, id_token puede contener claims standar:
    - iss: identificador de autoridad que expide el token
    - sub: id del usuario
    - aud: valor del parámetro client_id
    - exp: fecha en segundos en la que expira el token
    - iat: fecha en segundos en la que se generó el token.
- La petición de autenticación o autorización en los flujos OAUT incluye el parámetro scaope,

OIDC estandariza una serie de valores para el scope que conllevan la aparición de otros claims estándares en el id_token. Siempre debe contener al menos el valor openid.

OIDC se diferencia de OAuth en que, además de ser un protocolo de autorización, también es un protocolo de autenticación.

4.1.5 SAML

Security Assertion Markup Languaje es un protocolo de autenticación y autorización estandarizado por OASIS. OIDC es un rediseño conceptual de SAML en base a principios REST y JSON. La mayor parte de los IdPs proprocionan una implementación de SAML además de OAuth/OIDC.

- Contempla varios escenarios de uso llamados perfiles
- Suele ir muy ligado al suo de Active directory y LDAP, por lo que suele usarse solo en entornos corporativos.
- SAML no es apto para SSO en aplicaciones nativas, aunque existen soluciones comerciales que lo permiten.

4.1.6 Kerberos y SPNEGO

- Kerberos es un protocolo de autenticación y autorización bastante complejo basado en criptografía simétrica para establecer conexión entre cliente y servidor. esta pensado para cominicación entre cliente pesado y servicio dentro de la red corporativa de una empresa.
    - Una vez autenticado el usuario, las aplicaciones que arrancan pueden solicitar tickets (tokens) para los servicios que usan y en los que el usuario esté autorizado sin que el usuario tenga que introducir el nombre y la contraseña en cada aplicación.
- SPNEGO: para poder hacer uso de kerberos en aplicaciones web se unsa SPNEGO (Simple and Protected GSSAPI NEGOTIATION MECHANISM), un protocolo para que un cliente y un servicio negocien el uso de un esquema de seguridad para que ambos soporten RFC4178.
    - Uno de los usos más habiituales de SPNEGO es el esquema negotiate de HTTP Authorization y WWW-Authenticate, que se usa par aautenticar el navegador con una aplicación web, usando kerberos como esquema de autenticación y autorización.

4.3 Control de acceso

4.3.1 Basado en Roles

- **Modelo RBAC (Role Based Access Control)**: Se basa en restringir la ejecución de unas operaciones de un servicio a un usuario que tenga uno o varios roles.
    - El cliente envía una petición en nombre de un usuario e incluye con la petición algún tipo de token
    - El servicio a recibir la petición tiene que validar el token y a partir de este averiguar los rles de usuario y comprobar si tiene los permisos necesarios para ejecutar la operación solicitada

4.3.2 Basado en Atributos

Modelo ABAC (Attribute Based Access Control) hace frente a las limitaciones del modelo RBAC.

- Se proteje de la ejecución de una operación o acción con una política que permite o deniega en función de los atributos del sujeto que solicita la operación, el recurso sobre el que se realiza la operación y el entorno.
    - Atributo: Par nombre/valor
    - Sujeto: quien invoca la operación, un usuario u otro tipo de entidad
    - Recurso: Objeto sobre el que se realiza la operación, atributos del propio recuros
    - Entorno: Atributos como el día/hora, ubicación, dispositivo desde el que se accede…
    - Política: Conjunto de reglas que determinan si se permite la ejecución de una operación en función de los atributos del sujeto, recursos y entorno.
        - Se define declarativamente.
    - ABAC busca que las principales políticas de control de acceso no estén cableadas en las aplicaciones y servicios, de forma que si se deciden variar no es necesario actualizar las aplicaciones. Las políticas las pueden editar personas diferentes a los desarrolladores.

XACML (eXtensible Access Control Markup Languaje) es un estándar de OASIS que define un leguanje XML para la definición de políticas, una arquitectura de control de acceso y el formato de las peticiones /respuestas del PEP al PDP.

From:
http://www.knoppia.net/ - **Knoppia**

Permanent link:
http://www.knoppia.net/doku.php?id=app:turboresumen

Last update: **2025/07/14 14:48**