Programación segura

www.flagsolutions.net

flag solutions

Índice

- Introducción
- Desbordamientos de memoria.
- Condiciones de carrera
- Cadenas de formato
- Consejos para programar de forma segura.

Introducción

- Los programas malos son mucho más abundantes de lo que creemos.
- La forma de desarrollar los programas es responsable en gran medida del problema.
- Se invierte mucho tiempo, dinero y esfuerzo en seguridad a nivel de red por la mala calidad de los programas.

Introducción (2)

- Las tasas de defectos en productos comerciales se estiman entre 10 y 17 por cada 1000 líneas de código.
- Los programas no tienen garantía.
- La seguridad es un problema. Gestión de riesgos.
- Seguridad durante el diseño.

Situación

- Diciembre de 1990: Miller, Fredrickson. 'An empirical study of the reliability of Unix Utilities' (Communications of the ACM, Vol 33, issue 12, pp.32-44).
  - Entre el 25 y el 33% de las utilidades en Unix podían interrumpirse o colgarse proporcionándoles entradas inesperadas.

Situación(2)

- 1995: Entradas difusas en Unix:
  - Fallos entre un 15 y un 43%
  - Fallos avisados en el 90 persistentes
  - Utilidades de la FSF (7%) y a las incluidas junto con Linux (9%)
  - No falló ningún servidor de red ni X

Situación(3)

- 2000: Miller y Forrester. Win NT.
  - 45% de los programas se colgaron o se interrumpieron
  - Enviar mensajes aleatorios Win32 a las aplicaciones hacía fallar al 100%
- Miller, Cooksey y Moore. Mac OS X.
  - 7% de las aplicaciones de línea de órdenes.
  - De las 30 basadas en GUI 8 no se colgaron o se pararon.

Tras el error, hay que repararlo

- Julio 2002: Desbordamiento en OpenSSH
- Septiembre 2002: Sale Slapper

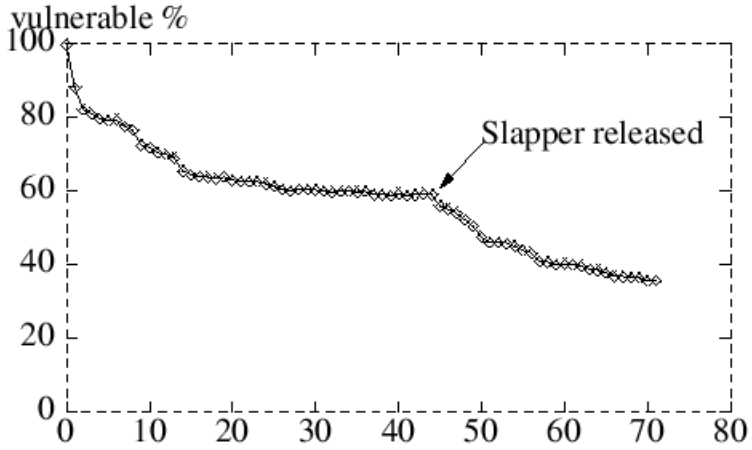


Figure 1 Vulnerable servers over time

flag solutions

¿Por qué es importante?

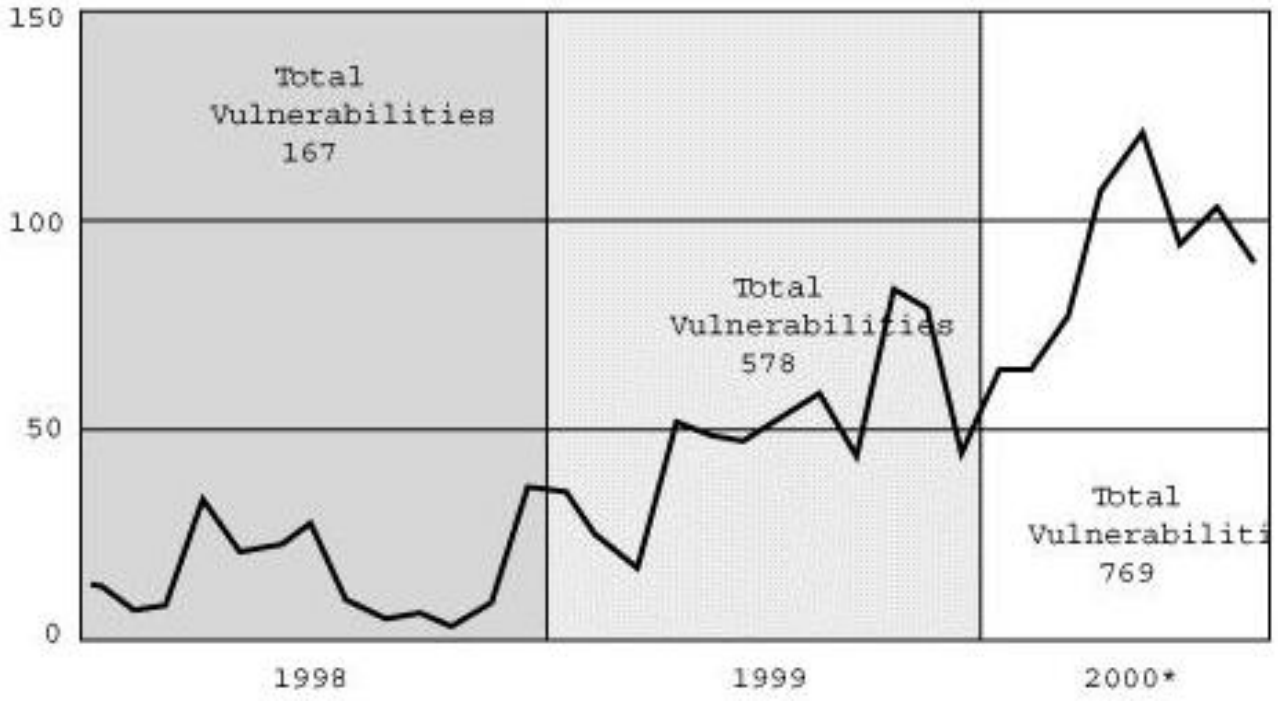
- Cada vez hay más computadores en más sitios.
- Al usuario no le interesan estos temas.
- Responsabilidad civil.
- Imagen.

¿Responsabilidad?

- 31 de diciembre de 1999. Las autoridades chinas obligaron a los ejecutivos de la compañía aérea nacional a volar durante esa noche en los vuelos programados.
- Modificación Código Civil de 12 de abril de 1994:
  - *"[…] si las variaciones se deben a falta de previsión de alguna de las partes, ésta indemnizará los daños y perjuicios y quedará privada de la facultad de desistir. […]"*

Cifras

- Bugtraq (Enero 1998 – Septiembre 2000)



Cifras (2)

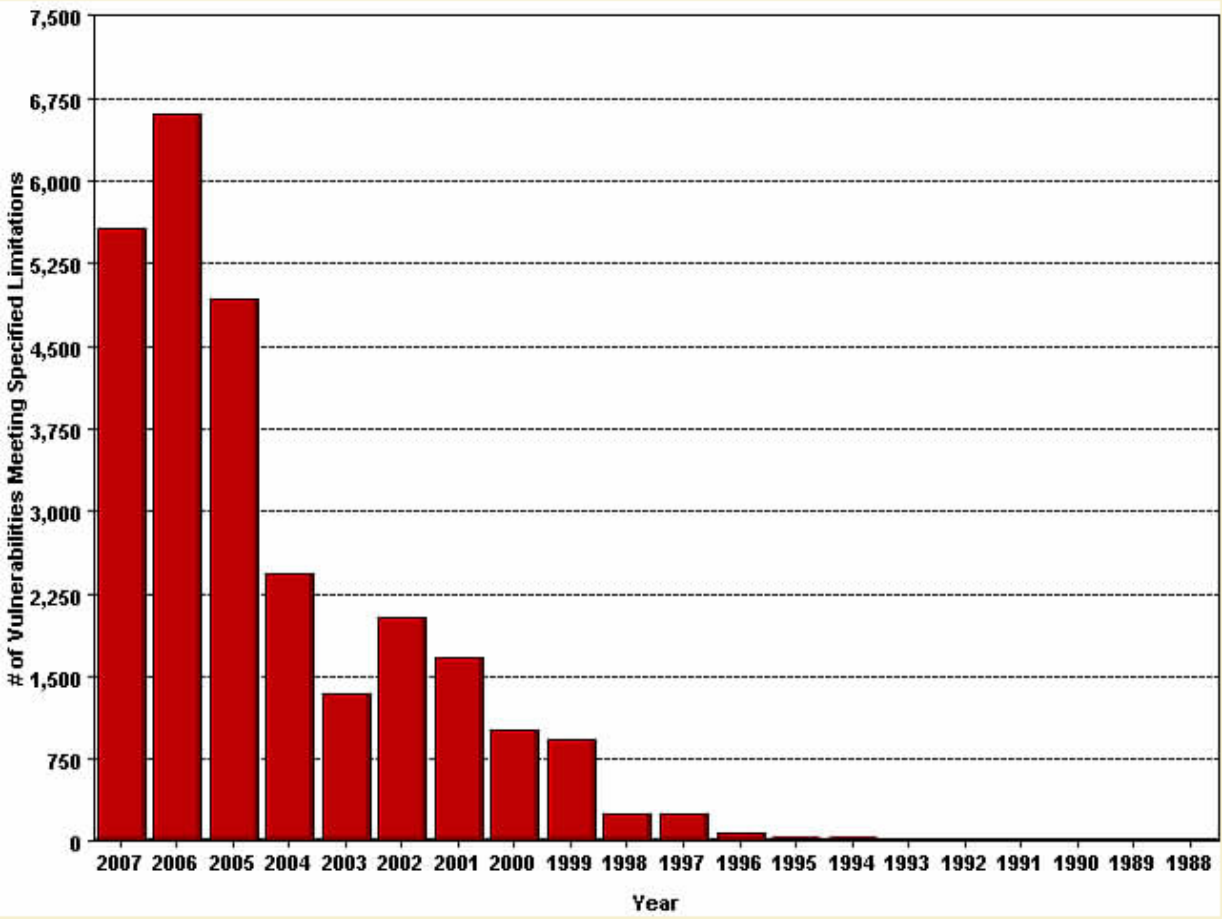
- NIST: National Institute of Standards and Technology
- NVD: National Vulnerabilities Database

Year	2007	2006	2005	2004	2003	2002	2001	2000	1999
# of Vulns	5573	6601	4926	2442	1339	2043	1677	1020	920
% of Total	101%	100%	100%	100%	100%	100%	100%	100%	100%

1998	1997	1996	1995	1994	1993	1992	1991	1990	1989	1988
246	252	75	25	25	13	13	15	11	3	2
100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

Cifras (3)

- http://nvd.nist.gov/statistics.cfm?results=1



Cifras (4)

- 2000: 20 vulnerabilidades / semana
- Unix y Windows están equilibrados
- Siguen apareciendo problemas en programas probados y usados.

flag solutions

¿Dónde conocerlas?

- Bugtraq (http://www.securityfocus.com/)
- CERT Advisories http://www.cert.org/
- http://www.rediris.es/cert/
- http://escert.upc.es/
- http://nvd.nist.gov/
- OSVDB, Open Source Vulnerability Database (http://osvdb.org/)
- Help Net Security http://www.net-security.org/

Causas

- La mayoría de los desarrolladores ni siquiera saben que hay un problema.
- Complejidad. Difícil probar que un sistema de complejidad mediana es seguro.
- Código de terceros (plugins y drivers)
- Falta de previsión
- A veces ***“no vale la pena”***

Complejidad (LOC)

- **Windows NT**: 35 millones.
- **Windows XP**: 40 millones.
- **Windows Vista**: 50 millones.
- **Linux 2.2:** 1.78 millones.
- **Solaris 7:** 400000.
- **Debian GNU/Linux 2.2**: 55 millones
- **Red Hat 6.2** 17 millones.
- **Mac OS X Darwin** 790000 (el kernel)

flag solutions

DESBORDAMIENTOS DE MEMORIA

Introducción

- **Definición:** Guardar mayor número de datos que el permitido por espacio reservado.

- Es el que lleva más tiempo con nosotros.
  - Internet worm (1988)
  - Más del 50% de los problemas de seguridad en 1999 (CERT)
  - 48% de los problemas de seguridad entre 2000 y 2004 (CERT)

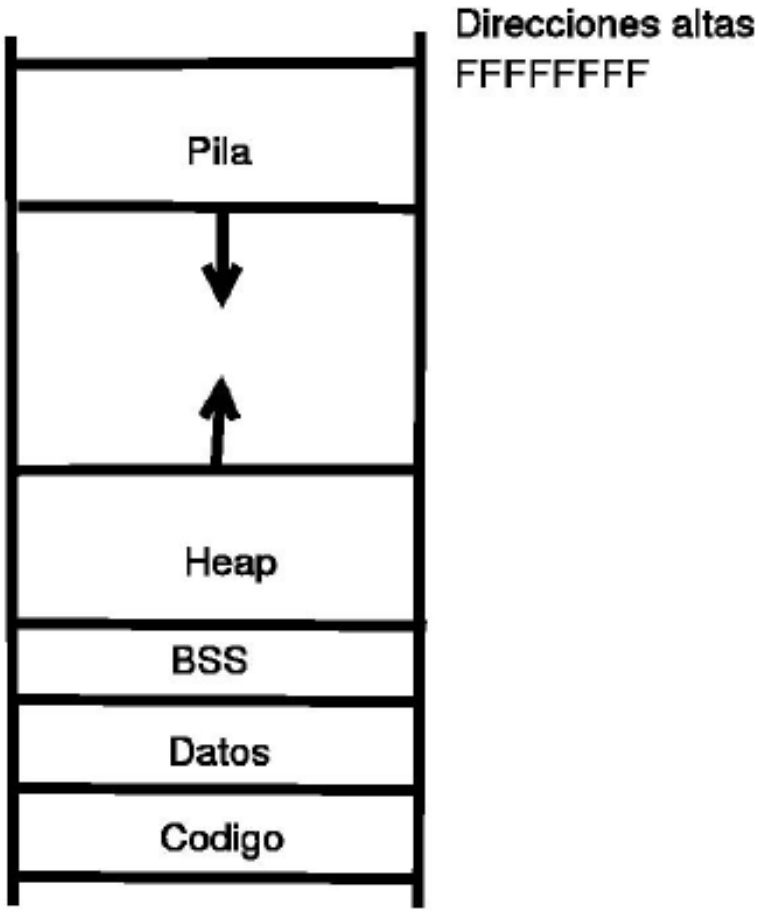
¿Obsoleto?

- Es extremadamente sencillo equivocarse
- Mal diseño del lenguaje
- Malas prácticas de programación
- Hay lenguajes inmunes, pero no siempre podremos usarlos
- Los lenguajes inmunes utilizan bibliotecas escritas en lenguajes ‘peligrosos’.

Qué son los desbordamientos

- Los programas necesitan almacenar datos en la memoria
- Se puede alojar en la pila (‘stack’) y en la zona de memoria dinámica (‘heap’)
  - En la pila se almacenan variables no estáticas y parámetros
  - pasados por valor
  - En el heap se guardan datos reservados con malloc y new.
- Desbordamiento de enteros
- Ataques de cadenas de formato (format strings)

Estructura en memoria



Zonas de memoria

- Zonas de memoria
  - La pila . . .
    - Parámetros y entorno del programa
    - Crece hacia la zona de memoria dinámica
  - La zona de memoria dinámica
    - Crece hacia la pila
  - Zona estática
    - Segmento de almacenamiento de bloques: datos de acceso global
    - Segmento de datos. Datos de acceso global, inicializados.
    - Segmento de texto. Código sólo lectura.

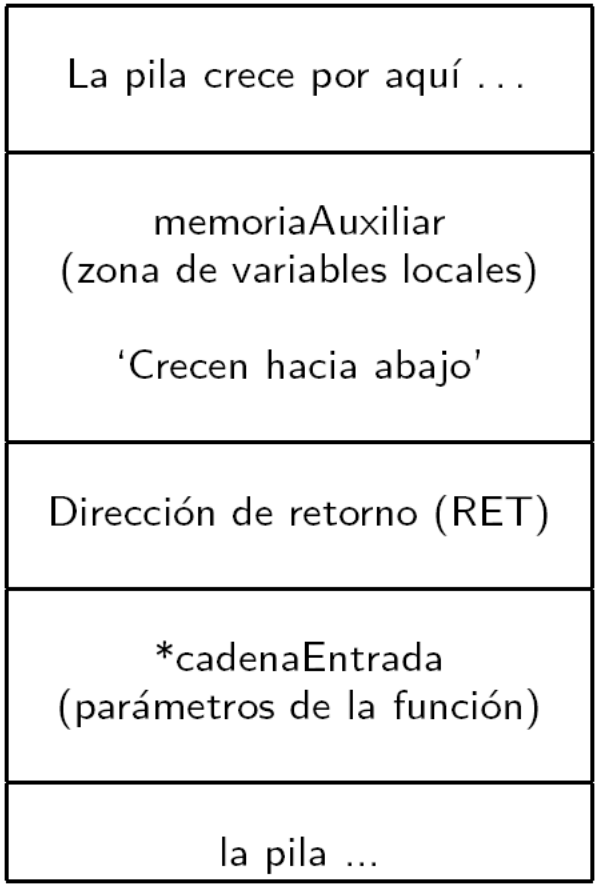
Mapa de memoria

- Salida del ejemplo1:

```
08048000-08049000 r-xp 00000000 03:42 175206     /home/fjp/memoria
08049000-0804a000 rw-p 00000000 03:42 175206     /home/fjp/memoria
0804a000-0806b000 rwxp 00000000 00:00 0
40000000-40016000 r-xp 00000000 03:42 157956     /lib/ld-2.3.2.so
40016000-40017000 rw-p 00015000 03:42 157956     /lib/ld-2.3.2.so
40017000-40019000 rw-p 00000000 00:00 0
40022000-4014a000 r-xp 00000000 03:42 158004     /lib/libc-2.3.2.so
4014a000-40152000 rw-p 00127000 03:42 158004     /lib/libc-2.3.2.so
40152000-40155000 rw-p 00000000 00:00 0
bffff000-c0000000 rwxp 00000000 00:00 0
Funci    s n   main :0x8048494
Variable global sin valor inicial:0x8049904
Variable global con valor inicial:0x80497ec
Variable local escalar: 0xbffffc20
Variable local vectorial: 0xbffffbf0
```

La pila

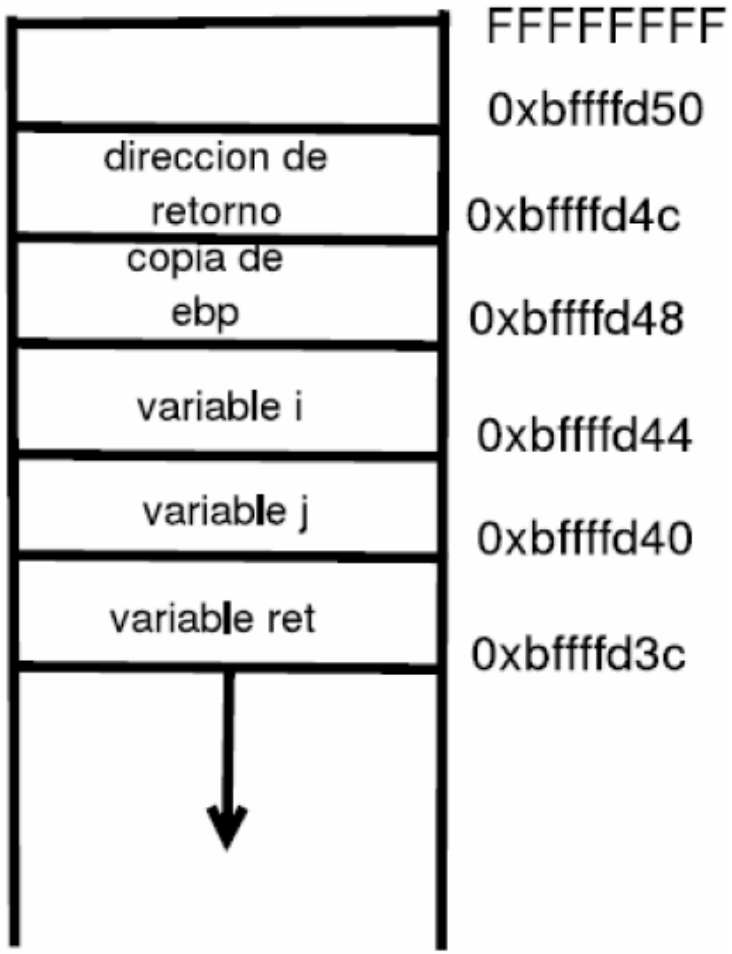
```
void funcion (char *cadenaEntrada)
{
    char memoriaAuxiliar[10];
    strcpy(memoriaAuxiliar,
        cadenaEntrada);
}
```



La pila (2)

- Dos registros especiales:
  - *esp: Stack Pointer*
  - *ebp: Base Pointer*
- Dos instrucciones especificas:
  - *push: guardar datos en la pila*
  - *pop: recuperar dato de pila*
- Otras instrucciones también usan la pila como *call* o *ret*.

Ejemplo


FFFFFFFF
0xbffffd50
direccion de retorno
0xbffffd4c
copia de ebp
0xbffffd48
variable i
0xbffffd44
variable j
0xbffffd40
variable ret
0xbffffd3c

Funcionamiento

- Si guardamos más datos de los debidos, irá al siguiente trozo de memoria.
- Consecuencias:
  - Ninguna. Funciona correctamente.
  - Actua de forma rara
  - Falla
- Depende de:
  - Cuántos datos
  - Qué datos
  - Qué se escribió
  - Quién lee esos datos

Objetivo de los errores

- Algunos de las acciones que pueden realizar los atacantes son:
  - Ejecutar código propio, con un nivel de privilegios mayor que el autorizado, se buscan los programas con bit Set-UID, Set-GID y Administrador
  - Modificar o acceder a ficheros privados.
  - Modificar el flujo de ejecución del programa.
  - Modificar el entorno de ejecución del fichero.

Desbordamientos de pila

- Siempre hay una zona interesante que sobreescribir: la dirección de retorno
  - Encontrar una zona de memoria candidata en la pila
  - Colocar código hostil en algún sitio
  - Sobrescribir sobre la dirección de retorno a ese código.

Desbordamientos de heap

- Hay que saber qué variables son críticas
- Colocar código para alterar esas variables.
- Puede ocurrir que la modificación ‘rompa’ la ejecución del programa
- Más difícil de atacar que la pila
  - No es la solución

Consejos de programación

- Comprobar los índices de los arrays
- Evitar funciones que no comprueben índices
  - strcmp(), strcat(), scanf(),…
- O reemplazarlas por funciones que lo hagan
  - strncpy(), strncat(),…
- Evitar strlen si no sabemos que la cadena tiene el carácter nulo (\0)

Consejos de programación (2)

- Comprobar las entradas
- Evitar funciones “peligrosas”
  - gets(), getchar(), fgetc(), getc(), read()
- Tener cuidado con el tamaño:
  - bcopy(), fgets(), memcpy(), snprintf(), strccpy(), strcadd(), strncpy(), vsnprintf(), getenv()
- No fiarse de código ajeno

Consejos de administradores

- Tener actualizados los programas que se usan.
- Estar atento a los anuncios de vulnerabilidades
- Ser capaz de parchear el código fuente o buscar soluciones.

Protección contra desbordamientos

- Bit NX o Bit XD
  - Separa áreas de código de áreas de datos
  - Evitan la inyección de código
  - Implementada en hardware
    - SPARC, PowerPC y modernos x86
  - Implementaciones software
    - Exec Shield y Pax (Linux)
    - DEP (Windows)

Protección contra desbordamientos (2)

- ASLR (*Address space layout randomization*)
  - Coloca las zonas de memoria en zonas aleatorias de memoria
  - Presente en Linux 2.6.20 y Windows Vista por defecto
  - Implementado mediante PAX y Exec Shield en Linux
  - Herramientas comerciales para windows (http://www.wehnus.com/)

Protección contra desbordamientos (3)

- Canarios
  - Similar a canarios en minas de carbón
    - Coloca valores entre los objetos de la pila
    - Cuando se produce una reescritura se modifican los canarios y se detecta.
  - Existen 3 tipos principales:
    - Terminadores
    - Aleatorios
    - Aleatorios usando XOR
  - Implementaciones:
    - StackGuard y Propolice (GCC)
    - Opción /GS en Visual Studio

Librerías seguras

- Libsafe
  - reemplazar las funciones inseguras por versiones seguras
- Strsafe
- Strlcpy, strlcat

Desbordamiento de enteros

- El tipo de datos no puede albergar los datos de nuestro programa.
- Signed y Unsigned

```
short int a = 25000;
short int b = 25000;
short int c = a + b ;
```

- Índices, tamaño de memoria, etc.

Cadenas de formato

- Permiten modificar el flujo del programa vulnerable.
- Se usan introduciendo cadenas de formato en las entradas del programa y que luego se van a tratar por funciones del tipo printf.

```
int main ( int argc , char  **argv )
{
        int num;
        printf ( ” %s %n\n” , ” foobar ” , &num) ;
        printf ( ” %d\n” , num) ;
}
```

flag solutions

HORA DEL CAFÉ

Condiciones de carrera

- Una fuente muy común de errores
- Comportamiento anómalo, donde intervienen varios hilos o procesos que utilizan de forma inapropiada los recursos de la máquina.
- Existen dos tipos de condiciones de carrera:
  - Interferencias causadas por programas malintencionados.
  - Interferencias causadas por programas legítimos.

Ejemplo

```
import          java.io.*;
import          javax.servlet.*;
import          javax.servlet.http.*;

public class Counter extends HttpServlet {

        int             count = 0;

        public void     doGet(HttpServletRequest in, HttpServletResponse out)
        throws          ServletException, IOException {

                out.setContentType("text/plain");
                Printwriter     p = out.getWriter();
                                count++;
                                p.println(count + " hits so far!");
        }
}
```

Algo más que suerte

- Si la posibilidad es baja, aumentando las peticiones, aumenta la probabilidad.
- Puede permitir escalada de privilegios o acceso a información no autorizada.

Soluciones

- Asegurarse de que las condiciones se cumplen.
- Operaciones atómicas
- Cerrar la puerta al entrar.
- Bloqueos

Bloqueos

- Sólo entra un hilo cada vez.
- Puede destrozar el desempeño

```
import          java.io.*;
import          javax.servlet.*;
import          javax.servlet.http.*;

public class Counter extends HttpServlet {
        int             count = 0;
        public synchronized void
                        doGet(HttpServletRequest ir, HttpServletResponse out)
        throws          ServletException, IOException {
                out.setContentType("text/plain");
                Printwriter     p = out.getWriter();
                                count++;
                                p.println(count + " hits so far!");
        }
}
```

Opción mejor

```
import          java.io.*;
import          javax.servlet.*;
import          javax.servlet.http.*;
public class Counter extends HttpServlet {
        int             count = 0;
        public void
                        doGet(HttpServletRequest in, HttpServletResponse out)
        throws          ServletException, IOException {
                int             my_count;
                                out.setContentType("text/plain");
                Printwriter     p = out.getWriter();
                                synchronized(this) {
                        my_count = ++count;
                }
                                p.println(my_count + " hits so far!");
        }
}
```

Condiciones de concurrencia

- Varios procesos acceden al recursos compartidos.
- Ficheros
- Más fácil en Unix que en Windows

Objetivos

- Crear un recurso temporal permitiendo que acceda un atacante.
- Crear un recurso temporal en un directorio con permisos débiles
- Crear un recurso en un directorio que creó un atacante
- Usar un recurso creado por un atacante
- Acceder al recursos mientras se está usando
  - Se crea el recurso y se modifican los permisos después
  - Se crea el recurso en un directorio y luego se mueve a otro
  - Los permisos del proceso se modifican temporalmente

Forma de conseguirlos

```
Si (tiene_propiedad(fichero)) {
    //Accedemos al fichero
    //Realizamos operaciones sobre él
}
```

- **TOCTOU:** time–of–check, time–of–use

Soluciones

- Utilizar descriptores de ficheros.
  - fstat, lstat
- Recoger los valores devueltos por las funciones.
- Crear los ficheros usando O_CREAT | O_EXCL con permisos restrictivos.

¿Y si no?

- Funciones especiales:
  - link, mkdir, mknod, rmdir, symlink, unmount, unlink, utime
- Ficheros en su propio directorio
- Sólo accesible para la UID del programa que hace las operaciones.
- Asegurarse nadie tenga acceso
  - Recorrer el directorio hacia arriba y comprobar permisos

¿Y el borrado?

- Sin asegurar el directorio no es posible
- **Unlink** recibe un nombre !!
- Las condiciones no son el único problema
  - Borrado seguro del contenido
  - Seguridad de que es el fichero que queremos borrar. Recuperación.

En resumen …

- **NO ESCRIBIR DATOS IMPORTANTES EN DISCO**
- Encriptarlos
- Guardar bien la clave
- Desencriptar en memoria

Ficheros temporales

- Poseen los mismos problemas de los anteriores.
- Además los nombres predecibles suponen un problema.
  - Nombres aleatorios
- Sticky-bit
- Bloqueo de acceso
  - En algunos casos es *sólo* un indicativo

Orden de acción

- Utilizar un prefijo propio.
- Generar n bits aleatorios y codificar en base64 o hexadecimal y concatenar.
- Poner una máscara adecuada umask, 0066
- Creamos el fichero con fopen()
- Marcar el fichero para borrar al cerrar
- Trabajar
- Cerrar el fichero
- Asegurar el borrarlo

Bloqueos

- Sistema V **lockf**
- BSD **flock**
- POSIX **fcntl**
- Unix **flock**

Ejemplo de bloqueo

```
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
int main(int argc, char *argv[]) {
    int fd; struct flock fl;
    fd = PrivoxyWindowOpen("testfile", O_RDWR);
    if (fd == -1) /* Handle error */;
    /* Make a non-blocking request to place a write lock
tfile */
    fl.l_type = F_WRLCK; fl.l_whence = SEEK_SET;
    fl.l_start = 100; fl.l_len = 10;
    if (fcntl(fd, F_SETLK, &fl) == -1) {
        if (errno == EACCES || errno == EAGAIN) {
            printf("Already locked by another process\n");
            /* We can't get the lock at the moment */
        } else {
            /* Handle unexpected error */;
        }
    } else { /* Lock was granted... */
```

Ejemplo de bloqueo (2)

```
    } else { /* Lock was granted... */
        /* Perform I/O on bytes 100 to 109 of file */
        /* Unlock the locked bytes */

        fl.l_type = F_UNLCK;
        fl.l_whence = SEEK_SET;
        fl.l_start = 100;
        fl.l_len = 10;
        if (fcntl(fd, F_SETLK, &fl) == -1)
            /* Handle error */;
    }
    exit(EXIT_SUCCESS);
} /* main */
```

Otros lenguajes

- Perl y PHP: flock
- Java
  - En Windows bloquea los ficheros abiertos
  - En Unix no

Bloqueo en Java

```
try {
    // Get a file channel for the file
    File file = new File("filename");
    FileChannel channel = new RandomAccessFile(file, "rw").getChannel();

    // Use the file channel to create a lock on the file.
    // This method blocks until it can retrieve the lock.
    FileLock lock = channel.lock();

    // Try acquiring the lock without blocking. This method returns
    // null or throws an exception if the file is already locked.
    try {
        lock = channel.tryLock();
    } catch (OverlappingFileLockException e) {
        // File is already locked in this thread or virtual machine
    }

    // Release the lock
    lock.release();

    // Close the file
    channel.close();
} catch (Exception e) {
}
```

Java y .NET

- **El cargador de clases** (*Class Loader)*
- **El verificador de archivos de clases** (*Class file verifier*)
- **El gestor de seguridad** (*Security Manager)*

Modelo de seguridad Java

Característica	JDK 1.0	JDK 1.1	Java 2 SDK
Acceso a los recursos del código local sin firma	Sin restricciones	Sin restricciones	Basado en política
Acceso a los recursos del código local firmado	No disponible	Sin restricciones si fiable o restringido por el *sandbox*	Basado en política
Acceso a los recursos del código remoto sin firma	Restringido por el *sandbox*	Restringido por el *sandbox*	Basado en política
Acceso a los recursos del código remoto firmado	No disponible	Sin restricciones si fiable o restringido por el *sandbox*	Basado en política
Servicios de firmado digital de código	No disponibles	JCA (DSA)	JCA (DSA)
Servicios criptográficos	No disponibles	JCE 1.1	JCE 1.2

Sobreescritura en web

- Impedir la sobreescritura de parámetros mediante .htaccess

```
<Directory />
  AllowOverride None
</Directory>
```

```
<Directory /admin>
  AllowOverride All
</Directory>
```

Sobreescritura en web (2)

- Impedir el acceso por defecto a los directorios

```
<Directory />
    Order Deny,Allow
    Deny from all
</Directory>
```

```
<Directory /usr/local/httpd>
    Order Deny,Allow
    Allow from all
</Directory>
```

Sobreescritura en web (3)

- Impedir el acceso a ficheros peligrosos

```
<Files ~ "^\.ht">
    Order allow,deny
    Deny from all
</Files>
```

Sobreescritura en web (4)

- Instalar el servidor sobre NTFS
- Instalar el directorio del servidor en otra partición.
- Crear un usuario para cada sitio
  - Usuarios anónimos

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.web>
        <identity impersonate="true" userName="MySiteAnUser2"
password="***">
    </system.web>
</configuration>
```

ISS 6

- AllowRestrictedChars
- MaxFieldLength
- MaxRequestBytes
- UrlSegmentMaxCount
- UrlSegmentMaxLength
- EnableNonUTF8

Comparativa IIS 5 y 6

IIS Component	IIS 5.0 default install	IIS 6.0 default install
Static file support	Enabled	Enabled
ASP	Enabled	Disabled
Server-side includes	Enabled	Disabled
Internet Data Connector	Enabled	Disabled
WebDAV	Enabled	Disabled
Index Server ISAPI	Enabled	Disabled
Internet Printing ISAPI	Enabled	Disabled
CGI	Enabled	Disabled
Microsoft FrontPage® server extensions	Enabled	Disabled
Password change interface	Enabled	Disabled
SMTP	Enabled	Disabled
FTP	Enabled	Disabled
ASP.NET	N/A	Disabled
Background Intelligence Transfer Service	N/A	Disabled