Sistemas Operativos

Protección y Seguridad

Sistemas Operativos

- ☑ Versión: Junio 2020
- ☑ Palabras Claves: Seguridad, Protección, Permisos, Matriz de Acceso, ACL, Acceso, Dominios, Capacidades
- ☑ Bibliografía recomendada: Sistemas Operativos Modernos, Andrew S. Tanembaum

Sistemas confiables

- La clase pasada analizamos distintos mecanismos de protección que nos permiten elevar el nivel de seguridad de un sistema.
- Otro factor que afecta el nivel de seguridad de un sistema, es el código:
  - Código mal intencionado: Virus, gusanos
  - Código con errores de programación: Backdoors
- ¿Es posible construir sistemas seguros? La respuesta es si, pero el gran enemigo de la seguridad es el agregado de funcionalidad.
  - Un sistema minimalista será probablemente muy seguro, pero pobremente usable
  - Un sistema con muchas características, probablemente tenga mas errores y abra más la posibilidad a que sea vulnerado

Sistemas confiables - Ejemplos

- MULTICS desarrollado en 1960 tenía como objetivo principal la seguridad
- El protocolo SMTP para el envío de mail se definió en el año 1982:
  - En aquel entonces solo permitía el envío de mensajes en formato de texto ASCII (seguro, pero poco funcional)
  - Con el tiempo se agregó la posibilidad de enviar otros tipos de documentos (ofimática, multimedia, etc.). Estos archivos pueden incluir macros o código malicioso.
- HTTP es otro claro ejemplo. Inicialmente solo permitía transferir datos desde un servidor a los clientes. Hoy en día permite ejecutar aplicaciones desde el lado del cliente y servir contenido dinámico, lo cual podría contener vulnerabilidades

Explotación de errores en código

- ☑ Los procesos, junto con el Kernel son una potencial amenaza a la seguridad de un sistema.
- ☑ Los atacantes aprovechan errores en la codificación del SO, o algún proceso con alto nivel de privilegios con el fin de que los mismos cambien su funcionamiento normal:
  - ☑ Desbordamiento de buffer
  - ☑ Cadenas de formato
  - ☑ Retorno a libc
  - ☑ Desbordamiento de enteros
  - ☑ Inyección de código

https://www.exploit-db.com/

Desbordamiento de buffer (buffer overflow)

- ☑ Tiene como objetivo sobreescribir datos de ciertas zonas de memoria intencionalmente
- ☑ Se trata de un error del software. Se quiere copiar una cantidad de datos más grande que el área definida
- ☑ Ejemplo simple:

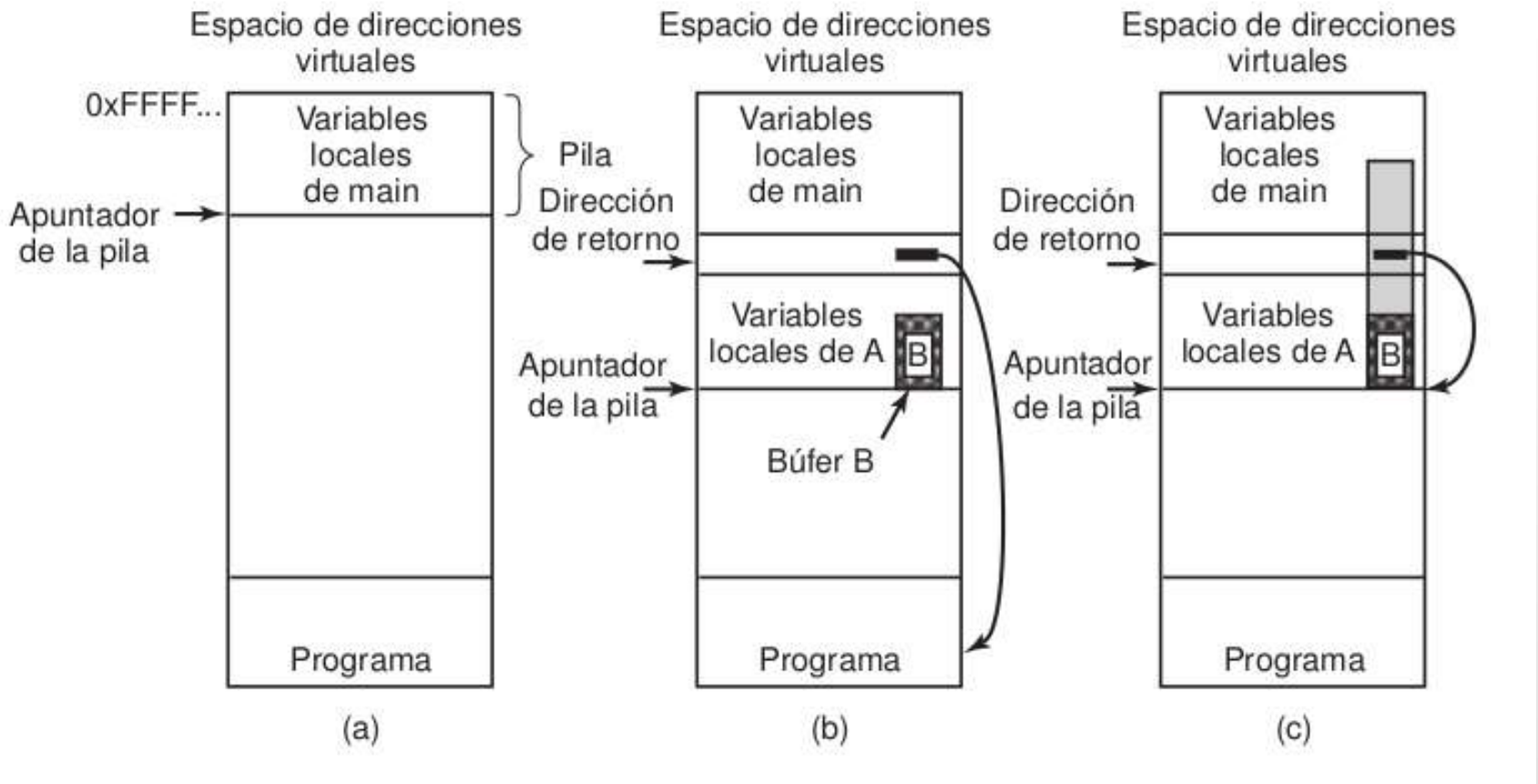
```
int i;
char c[1024];
i = 12000;
c[i]=0;
```

- ☑ El resultado es que se sobrescribe cierto byte de memoria que esté a una posición de 10,976 bytes fuera del arreglo
- ☑ Los compiladores de C no realizan esta comprobación, con lo cual este error debe ser manejado en tiempo de ejecución

Desbordamiento de buffer (buffer overflow)

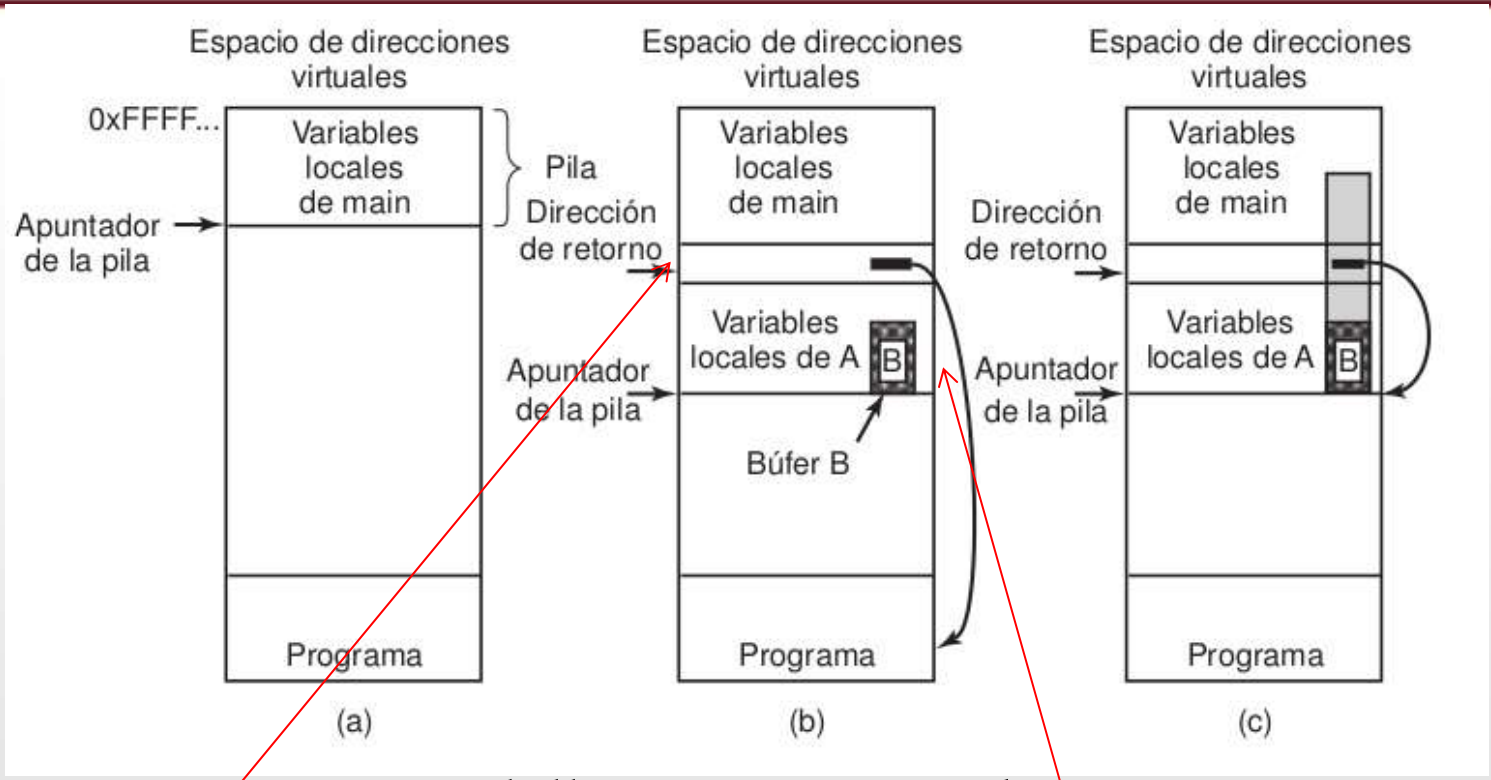
- ☑ Si el error del ejemplo anterior no pudiera ser detectado en runtime, podría alterarse el flujo del programa
- ☑ Si el dato que se sobreescribe no tiene un sentido, entonces probablemente lo que se verá es un error en la ejecución
- ☑ Si la posición de memoria se sobreescribe con un valor sensible, la ejecución podría encadenar una nueva ejecución de un programa malicioso
- ☑ Shellcode: código copiado especialmente preparado para obtener los privilegios del programa atacado.

Ejemplo



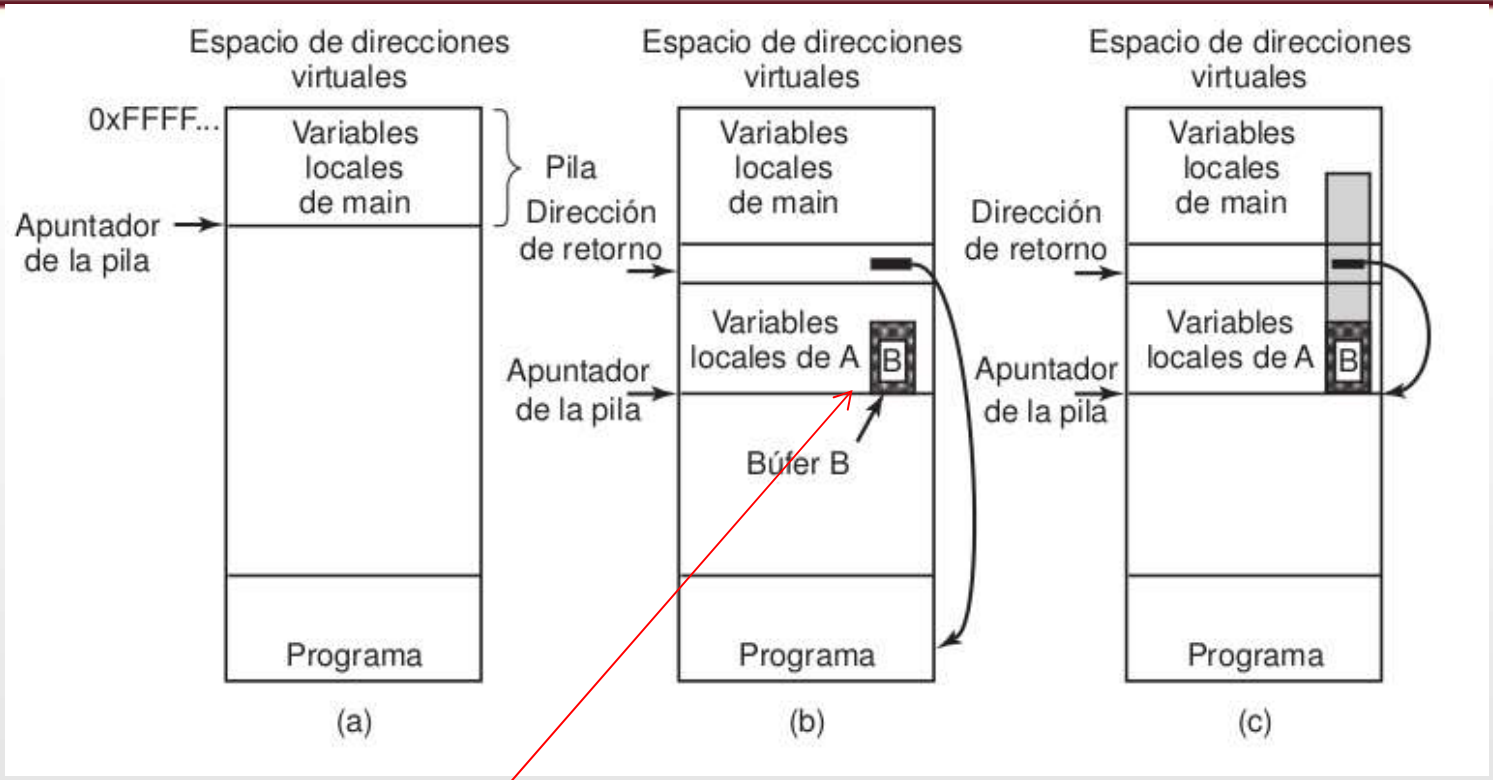
- ☑ En la primera figura se puede ver un programa en ejecución con sus variables locales en la pila

Ejemplo (cont.)



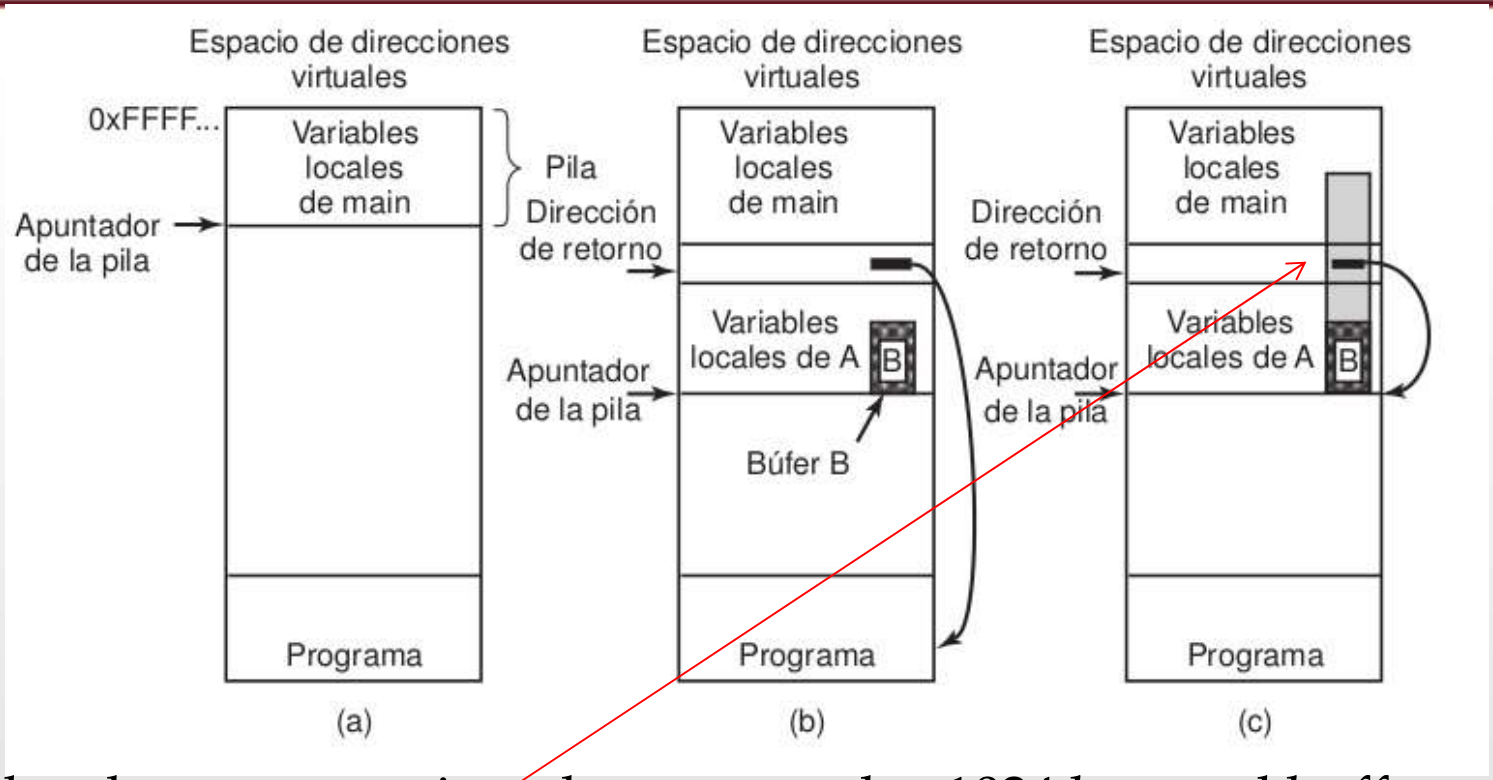
- ☑ El programa principal, llama a un procedimiento A:
  1. Apila la dirección de retorno (siguiente instrucción de main)
  2. Se transfiere el control a "A" quien asigna espacio a sus direcciones locales e incrementa el apuntador de la pila

Ejemplo (cont.)



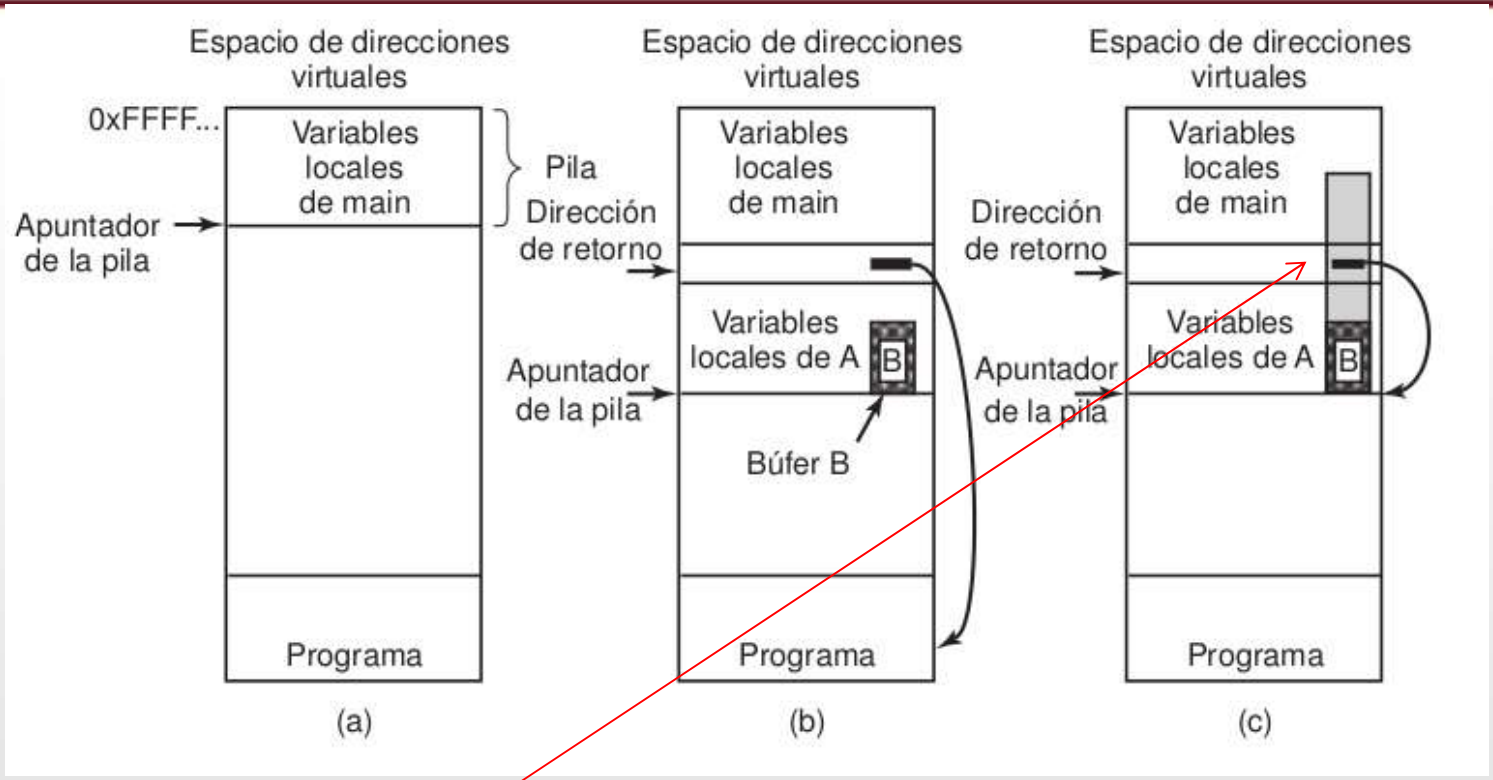
- Supongamos que A debe leer y formar el nombre de una ruta completa a un archivo. Para ello define un buffer "B" de 1024 bytes (teniendo en cuenta los nombres de archivos y directorios, esta capacidad debería ser suficiente) donde irá concatenando los datos leídos.

Ejemplo (cont.)



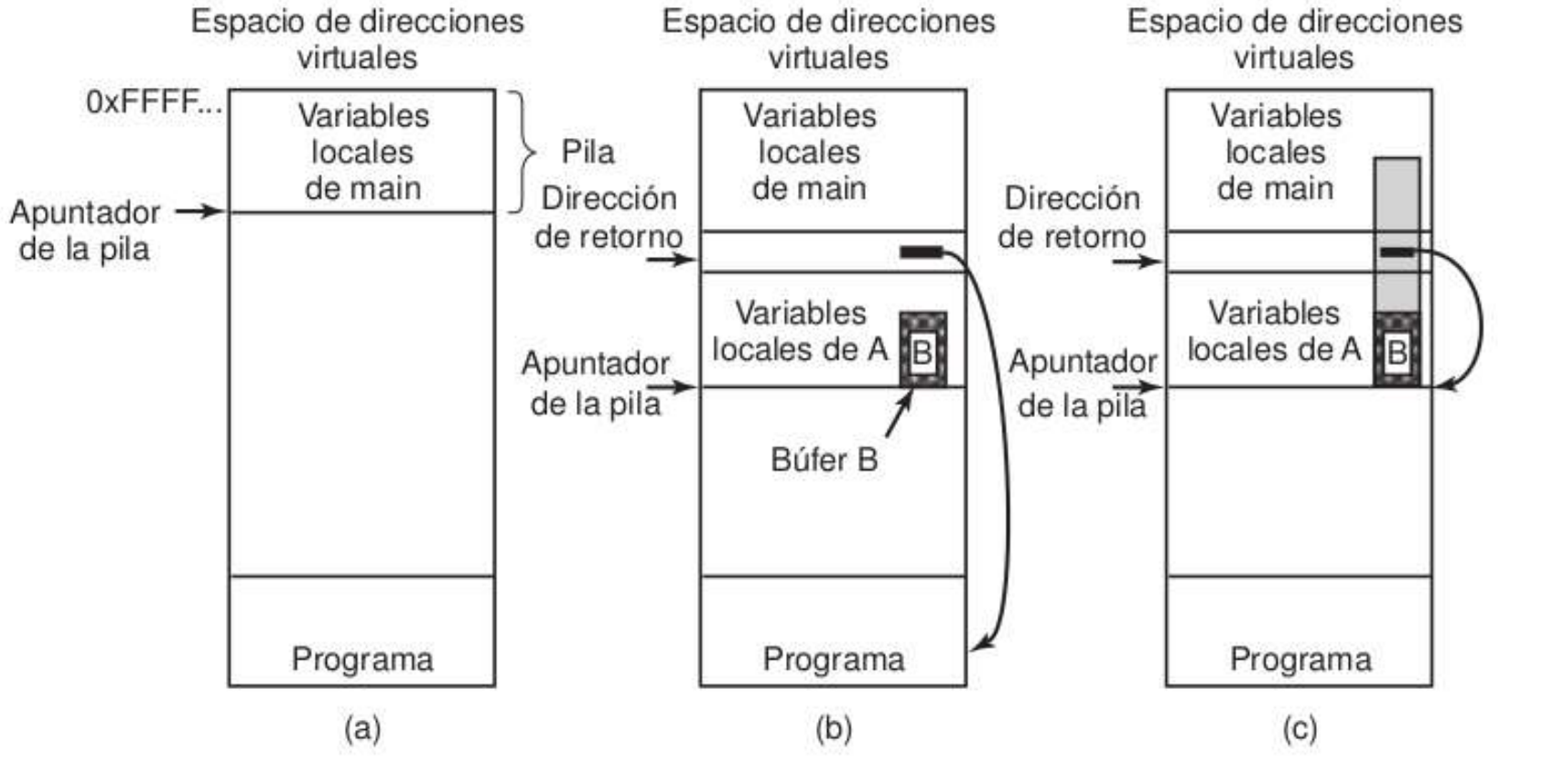
- ☑ Si los datos proporcionados superan los 1024 bytes, el buffer podría seguir creciendo, sobrescribiendo de este modo otras áreas de memoria con "posibles datos sensibles"
- ☑ Al abrir el archivo se producirá un error (no existe), pero la memoria ya se sobreescribió

Ejemplo (cont.)



- ☑ Si se sobrescribe la dirección de retorno, se podría saltar a una dirección aleatoria
- ☑ Si se sobrescribe inteligentemente, se podría lograr la ejecución de código malicioso

Ejemplo (cont.)



- ☑ Si el programa que se ataca es SETUID a root en Unix, el Shell code ejecutado se ejecutaría como root.

Cadenas de formato

- ¿Qué hace esta pequeña fracción de código escrito en C?:

```
char *s="Hola programador";
printf ("%s",s);
```

"%s", que le indica que debe imprimir una cadena

- ¿Y esta otra?¿Hacen lo mismo?:

```
char *s="Hola programador";
printf (s);
```

- Supongamos que tiempo mas tarde se modifica el código, de modo tal de que el mismo pida el nombre de usuario e imprima un mensaje que lo incluya:

```
char s[100], g[100] = "Hola";
gets(s); /* lee de teclado */
strcat(g, s); /* concatena */
printf(g); /* imprime */
```

Vulnerable a ataque de desbordamiento de buffer

- ¿Qué problema podría generar este código?

Cadenas de formato

```
char s[100], g[100] = "Hola";
gets(s); /* lee de teclado */
strcat(g, s);
printf(g);
```

- ☑ Lo que se ingresa por teclado, podría no se una cadena válida, sino que podría incluir caracteres que hagan que el printf funcione de modo no deseado
- ☑ Por ejemplo:

```
int i=0;
printf("Hola %nmundo\n", &i); /* %n se almacena en i */
printf("i=%d\n", i); /* ahora i es 6 */
```

- ☑ Cuando el programa se compila y ejecuta, su resultado es:

```
Hola programador
i=6
```

Cadenas de formato

```
int i=0;
printf("Hola %nmundo\n", &i); /* %n se almacena en i */
printf("i=%d\n", i); /* ahora i es 6 */
```

- ☑ Ejemplo: %n no imprime. Genera que se guarde un valor (posición) en memoria en la dirección del siguiente argumento de printf
- ☑ Se puede sobrescribir la dirección de retorno de printf y saltar donde quiera.
- ☑ Si el programa se ejecuta con SETUID root, se podría saltar a un Shell code que se ejecute con esos privilegios…

Ataque por inyección de código

- ☑ Este ataque apunta a ejecutar código sin darse cuenta.
- ☑ ¿Qué hace el siguiente programa?

```
int main(int argc, char *argv[])
{
   char org[100], dst[100], cmd[205] = "cp ";                  /* declara 3 cadenas */
   printf("Escriba el nombre del archivo de origen: ");        /* pide el archivo de origen */
   gets(org);                                                  /* obtiene la entrada del teclado */
   strcat(cmd, scr);                                           /* concatena src después de cp */
   strcat(cmd, " ");                                           /* agrega un espacio al final de cmd */
   printf("Escriba el nombre del archivo de destino: ");       /* pide el nombre del archivo de salida */
   gets(dst);                                                  /* obtiene la entrada del teclado */
   strcat(cmd, dst);                                           /* completa la cadena de comandos */
   system(cmd);                                                /* ejecuta el comando cp */
}
```

Ataque por inyección de código (cont.)

```
int main(int argc, char *argv[])
{
   char org[100], dst[100], cmd[205] = "cp ";          /* declara 3 cadenas */
   printf("Escriba el nombre del archivo de origen: "); /* pide el archivo de origen */
   gets(org);                                          /* obtiene la entrada del teclado */
   strcat(cmd, scr);                                   /* concatena src después de cp */
   strcat(cmd, " ");                                   /* agrega un espacio al final de cmd */
   printf("Escriba el nombre del archivo de destino: "); /* pide el nombre del archivo de salida */
   gets(dst);                                          /* obtiene la entrada del teclado */
   strcat(cmd, dst);                                   /* completa la cadena de comandos */
   system(cmd);                                        /* ejecuta el comando cp */
}
```

- ☑ En un estado de ejecución normal, podría ejecutar por ejemplo `cp abc xyz`
- ☑ Si un usuario malintencionado, ingresara otra cadena al soliticar `dst`, como por ejemplo "`xyz; rm -rf /`", el programa podría ejecutar `cp abc xyz;rm -rf /`