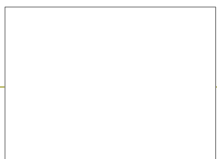


Máster Universitario en Ingeniería Informática



CyberSeguridad Tema 3: Programación segura

José Ismael Ripoll Ripoll



§ Técnicas de protección de memoria.

- ◆ DEP
- ◆ SSP
- ◆ ASLR

§ Técnicas de prevención

- ◆ Guías de estilo



§ DEP/NX, Data Execution Prevention/ Non-eXecutable:

- ◆ “prevenir que una aplicación o servicio se ejecute desde una región de memoria no ejecutable” [WP]
- ◆ Utiliza los permisos que implementa el procesador para manejar la memoria virtual (paginación) para impedir que páginas que contienen datos, no puedan ser ejecutadas.
- ◆ El procesador es capaz de diferenciar entre un acceso a memoria para cargar un operando (load) y otro para ejecutar una instrucción (fetch).
- ◆ Realizar operaciones de fetch en direcciones que pertenezcan a páginas sin permiso de ejecución produce una interrupción del proceso.

https://es.wikipedia.org/wiki/Prevenci%C3%B3n_de_ejecuci%C3%B3n_de_datos



- § Para utilizar DEP, es necesario que los programas diferencien bien entre área de datos (heap, .bss y .data) y área de código (.text).
- § En Linux existe una clara separación excepto en los programas que implementan JIT (Just In Time Compilation) .
 - ◆ Navegadores (Chrome, Firefox) para optimizar la ejecución de JavaScript.



*Investiga qué procesos tienen
Mapas con permisos de “**rwX**”*

- § Muy efectiva frente a la inyección de código.
 - ◆ Ampliamente utilizada en la mayoría de S.O.



§ SSP, Stack Smashing Protection

- ◆ El compilador genera código para colocar en la pila entre los buffers y la dirección de retorno un valor aleatorio llamado “canario”. En caso de un buffer overflow, el canario se ve alterado (sobre escrito). El compilador también genera código para comprobar el valor del canario antes de retornar de la función (utilizar la dirección de retorno).

§ Existen diversas variantes y mejoras de esta técnica:

- ◆ ProPolicy
- ◆ StackGuard
- ◆ RenewSSP
- ◆ stack-protector-strong



- § SSP es muy efectivo contra ataques de desbordamiento de pila. Puesto que la mayoría de los desbordamientos se producen de forma secuencial, y por tanto es imposible “saltarse” (dejar intacto) el canario, con lo que tiene una alta tasa de bloqueos.
- § El valor del canario debe ser un número aleatorio, pues de lo contrario, el atacante podría construir un exploit que sobrescribiera el canario con el valor correcto → se saltaría esta protección.
- § La seguridad del SSP reside en mantener secreto el valor del canario.



- § Esta técnica la tiene que implementar el compilador.
- § El GCC lo implementa desde hace más de una década.
 - ◆ Man gcc (busca stack-protector).
 - ◆ Está activado por defecto. Esto es, todos los programas que se crean tiene la pila de las funciones protegida.
- § Añade una pequeña sobrecarga a la ejecución de los procesos.
 - ◆ Es necesario insertar en la pila el canario y al salir de la función comprobar que no ha sido cambiado.




```
#include <stdio.h>

void protegida(void ){
    char nombre[16];

    gets(nombre);
    printf ("Nombre=[%s]\n", nombre);
    return;
}

int main(void ){
    protegida();
    return 0;
}
```

```
$ gcc simple-ssp.c -o simple-ssp
$ ./simple-ssp
Hola
Nombre=[Hola]
$ ./simple-ssp
123456789012345678901234567890
Nombre=[123456789012345678901234567890]
*** stack smashing detected ***: ./a.out terminated
Abortado (`core' generado)
```



\$ objdump -d simple-ssp | less

```
... . . .
00000000004005ed <protegida>:
 4005ed:      55
 4005ee:      48 89 e5
 4005f1:      48 83 ec 20
 4005f5:      64 48 8b 04 25 28 00
 4005fa:      5 f8
 400604:      48 8d 45 e0
 400608:      48 89 c7
 40060b:      e8 e0 fe ff ff
 400610:      48 8d 45 e0
 400614:      48 89 c6
 400617:      bf d4 06 40 00
 40061c:      b8 00 00 00 00
 400621:      e8 9a fe ff ff
 400626:      90
 400627:      48 8b 45 f8
 40062b:      64 48 33 04 25 28 00
```

El valor secreto del canario está en: %fs:0x28
Se copia en la pila: -0x8(%rbp)

- 1.- Se recoge el valor de la pila: -0x8(%rbp) → %rax
- 2.- Se compara con el original con una xor
- 3.- Si son iguales (je) se continua con el final de la funcion
- 4.- Else se llama a __stack_chk_fail(), que aborta el

```
40063d:      55
40063e:      48 89 e5
400641:      e8 a7 ff ff ff
400646:      b8 00 00 00 00
40064b:      5d
40064c:      c3
```

```
push    %rbp
mov     %rsp,%rbp
sub     $0x20,%rsp
mov     %fs:0x28,%rax

mov     %rax,-0x8(%rbp)
xor     %eax,%eax
lea     -0x20(%rbp),%rax
mov     %rax,%rdi
callq   4004f0 <gets@plt>
lea     -0x20(%rbp),%rax
mov     %rax,%rsi
mov     $0x4006d4,%edi
mov     $0x0,%eax
callq   4004c0 <printf@plt>
nop
mov     -0x8(%rbp),%rax
xor     %fs:0x28,%rax

je      40063b <protegida+0x4e>
callq   4004b0 <__stack_chk_fail@plt>
leaveq  %rdi
retq
```

```
push    %rbp
mov     %rsp,%rbp
callq   4005ed <protegida>
mov     $0x0,%eax
pop     %rbp
retq
```

§ ASLR, Address Space Layout Randomization

¿En qué direcciones de memoria se ejecutan los programas?

- § Gracias a la magia de la memoria virtual, todos los procesos pueden utilizar el mismo espacio de direcciones virtuales, aunque cada uno de ellos resida en posiciones de memoria física diferente, CLARO!!
- § Per tant, el compilador puede elegir en qué direcciones de memoria “colocar” el proceso.
- § El cargador del sistema operativo tiene que leer la cabecera del ejecutable y ponerlo donde este le diga.



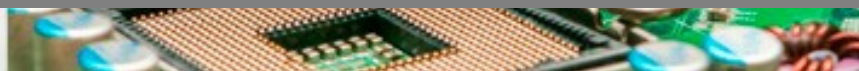
\$ readelf -h /bin/cat

Encabezado ELF:

```

Mágico:      7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
Clase:                               ELF64
Datos:                               complemento a 2, little endian
Versión:                               1 (current)
OS/ABI:                               UNIX - System V
Versión ABI:                               0
Tipo:                               EXEC (Fichero ejecutable)
Máquina:                               Advanced Micro Devices X86-64
Versión:                               0x1
Dirección del punto de entrada:                               0x402602
Inicio de encabezados de programa:       64 (bytes en el fichero)
Inicio de encabezados de sección:        46112 (bytes en el fichero)
Opciones:                               0x0
Tamaño de este encabezado:                64 (bytes)
Tamaño de encabezados de programa:       56 (bytes)
Número de encabezados de programa:       9
Tamaño de encabezados de sección:        64 (bytes)
Número de encabezados de sección:        28
Índice de tabla de cadenas de sección de encabezado: 27

```



```
$ for i in /bin/* ; do ( readelf -h $i | grep entrada:);  
done 2> /dev/null | sort
```

.....

Dirección del punto de entrada:	0x400509
Dirección del punto de entrada:	0x400a6c
Dirección del punto de entrada:	0x400b04
Dirección del punto de entrada:	0x400b1c
Dirección del punto de entrada:	0x400b72
Dirección del punto de entrada:	0x400be3
Dirección del punto de entrada:	0x400be8
Dirección del punto de entrada:	0x400c48
Dirección del punto de entrada:	0x400c7c
Dirección del punto de entrada:	0x4011fc
Dirección del punto de entrada:	0x4013e2
Dirección del punto de entrada:	0x4013e8
Dirección del punto de entrada:	0x401487
Dirección del punto de entrada:	0x401487
Dirección del punto de entrada:	0x401487
Dirección del punto de entrada:	0x401487
Dirección del punto de entrada:	0x40152a
Dirección del punto de entrada:	0x40154e



- § Si el atacante conoce las posiciones en las que se ejecutará nuestro proceso → puede construir ROPS fácilmente
- § Recuerda que para construir un ROP es necesario poner las “direcciones absolutas” de los gadgets.
- § Por tanto, **si el atacante desconociera** en qué posiciones está cargado (y ejecutándose) el proceso, entonces no podría construir ROPS!!
- § **ASLR consiste en cargar el proceso en posiciones aleatorias.**



- § Existen muchas implementaciones de la idea ASLR.
- § La primera la realizó el PaX Team:
 - ◆ <https://pax.grsecurity.net/docs/aslr.txt>
- § La implementación de ASLR depende en gran medida del soporte de **memoria virtual** del procesador y del soporte a librerías **dinámicas**.
- § Linux fue el primer S.O. en dar soporte a ASLR.
- § Existen más información sobre como “bypasear” ASLR que sobre el propio ASLR.
- § La implementación de ASLR para 32 bits es realmente lamentable tanto en Linux como Windows.



- § Los procesos están formados por muchos segmentos de memoria: la pila, el ejecutable, las librerías, el heap, etc.
- § Para que el ASLR sea efectivo TODAS las zonas o segmentos deben ser aleatorios.
- § Aleatorizar los segmentos de datos es relativamente fácil. Tanto la pila como el heap son por definición dinámicos, y los programas no asumen ninguna dirección inicial.

```
char *ptr;  
  
ptr= malloc(100);  
gets(ptr);           // El valor de ptr puede ser cualquiera
```



§ Pero el código es menos flexible (en x86).



§ In Intel (32bits), las instrucciones máquina de llamada a subrutina utilizan direcciones de memoria absolutas.

§ Para poder situar el código en cualquier posición es necesario:

- ◆ O bien modificar el código durante la carga (reubicación).
- ◆ O bien el código se reescribe para ser “independiente de la posición”

§ La segunda opción se denomina PIC o PIE:

- ◆ PIC: Position Independant Code.
- ◆ PIE: Position Independant Executable.

PIC y PIE son básicamente lo mismo, el primero se usa con librerías y el segundo con ejecutables.



```
$ cat /proc/self/maps
00400000-0040b000 r-xp 00000000 08:01 1443351 /bin/cat
0060a000-0060b000 r--p 0000a000 08:01 1443351 /bin/cat
0060b000-0060c000 rw-p 0000b000 08:01 1443351 /bin/cat
01031000-01052000 rw-p 00000000 00:00 0 [heap]
7ffc0350f000-7ffc041cf000 r--p 00000000 08:01 917680 /usr/lib/locale/locale-archive
7ffc041cf000-7ffc0438a000 r-xp 00000000 08:01 2621886 /lib/x86_64-linux-gnu/libc-2.19.so
7ffc0438a000-7ffc04589000 ---p 001bb000 08:01 2621886 /lib/x86_64-linux-gnu/libc-2.19.so
7ffc04589000-7ffc0458d000 r--p 001ba000 08:01 2621886 /lib/x86_64-linux-gnu/libc-2.19.so
7ffc0458d000-7ffc0458f000 rw-p 001be000 08:01 2621886 /lib/x86_64-linux-gnu/libc-2.19.so
7ffc0458f000-7ffc04594000 rw-p 00000000 00:00 0
7ffc04594000-7ffc045b7000 r-xp 00000000 08:01 2621872 /lib/x86_64-linux-gnu/ld-2.19.so
7ffc045b7000-7ffc04799000 rw-p 00000000 00:00 0
7ffc04799000-7ffc047b4000 rw-p 00000000 00:00 0
7ffc047b4000-7ffc047b6000 r--p 00022000 08:01 2621872 /lib/x86_64-linux-gnu/ld-2.19.so
7ffc047b6000-7ffc047b7000 rw-p 00023000 08:01 2621872 /lib/x86_64-linux-gnu/ld-2.19.so
7ffc047b7000-7ffc047b8000 rw-p 00000000 00:00 0
7ffc047b8000-7ffc047b9000 rw-p 00000000 00:00 0
7fff86853000-7fff86874000 rw-p 00000000 00:00 0 [stack]
7fff86874000-7fff869d2000 r-xp 00000000 00:00 0 [vdso]
ffffffffffff600000-ffffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]
```

En Ubuntu, desde 2019 la mayoría de procesos son
PIE.



```
$ cat /proc/self/maps
00400000-0040b000 r-xp 00000000 08:01 1443351 /bin/cat
0060a000-0060b000 r--p 0000a000 08:01 1443351 /bin/cat
0060b000-0060c000 rw-p 0000b000 08:01 1443351 /bin/cat
01c15000-01c36000 rw-p 00000000 00:00 0 [heap]
7ff6ecd7000-7ff6eda97000 r--p 00000000 08:01 917680 /usr/lib/locale/locale-archive
7ff6eda97000-7ff6edc52000 r-xp 00000000 08:01 2621886 /lib/x86_64-linux-gnu/libc-2.19.so
7ff6edc52000-7ff6ede51000 ---p 001bb000 08:01 2621886 /lib/x86_64-linux-gnu/libc-2.19.so
7ff6ede51000-7ff6ede55000 r--p 001ba000 08:01 2621886 /lib/x86_64-linux-gnu/libc-2.19.so
7ff6ede55000-7ff6ede57000 rw-p 001be000 08:01 2621886 /lib/x86_64-linux-gnu/libc-2.19.so
7ff6ede57000-7ff6ede5c000 rw-p 00000000 00:00 0
7ff6ede5c000-7ff6ede7f000 r-xp 00000000 08:01 2621872 /lib/x86_64-linux-gnu/ld-2.19.so
7ff6ee061000-7ff6ee064000 rw-p 00000000 00:00 0
7ff6ee07c000-7ff6ee07e000 rw-p 00000000 00:00 0
7ff6ee07e000-7ff6ee07f000 r--p 00022000 08:01 2621872 /lib/x86_64-linux-gnu/ld-2.19.so
7ff6ee07f000-7ff6ee080000 rw-p 00023000 08:01 2621872 /lib/x86_64-linux-gnu/ld-2.19.so
7ff6ee080000-7ff6ee081000 rw-p 00000000 00:00 0
7fff0aa3a000-7fff0aa5b000 rw-p 00000000 00:00 0
7fff0abcc000-7fff0abce000 r-xp 00000000 00:00 0 [stack]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0 [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]
```

Antes de 2020, el código del ejecutable NO era PIE.
Ahora ya la mayoría de distribuciones si que son PIE.



```
$ gcc -pie -fpie simple-ssp.c -o simple-ssp
$ ./simple-ssp
^Z
$ ps aux | grep simple-ssp
iripoll  8841  0.0  0.0   4192   356 pts/3    T   18:33   0:00  ./simple-ssp
$ cat /proc/8841/maps
7f93ceaf9000-7f93cecb4000 r-xp 00000000 08:01 2621886    /lib/x86_64-linux-gnu/libc-2.19.so
7f93cecb4000-7f93ceeb3000 ---p 001bb000 08:01 2621886    /lib/x86_64-linux-gnu/libc-2.19.so
7f93ceeb3000-7f93ceeb7000 r--p 001ba000 08:01 2621886    /lib/x86_64-linux-gnu/libc-2.19.so
7f93ceeb7000-7f93ceeb9000 rw-p 001be000 08:01 2621886    /lib/x86_64-linux-gnu/libc-2.19.so
7f93ceeb9000-7f93ceebf000 rw-p 00000000 00:00 0
7f93ceebf000-7f93ceee1000 r-xp 00000000 08:01 2621872    /lib/x86_64-linux-gnu/ld-2.19.so
7f93cf0c3000-7f93cf0c6000 rw-p 00000000 00:00 0
7f93cf0dd000-7f93cf0e0000 rw-p 00000000 00:00 0
7f93cf0e0000-7f93cf0e1000 r--p 00022000 08:01 2621872    /lib/x86_64-linux-gnu/ld-2.19.so
7f93cf0e1000-7f93cf0e2000 rw-p 00023000 08:01 2621872    /lib/x86_64-linux-gnu/ld-2.19.so
7f93cf0e2000-7f93cf0e3000 rw-p 00000000 00:00 0
7f93cf0e3000-7f93cf0e4000 r-xp 00000000 08:01 2359311    /tmp/simple-ssp
7f93cf2e3000-7f93cf2e4000 r--p 00000000 08:01 2359311    /tmp/simple-ssp
7f93cf2e4000-7f93cf2e5000 rw-p 00001000 08:01 2359311    /tmp/simple-ssp
7fff310bd000-7fff310de000 rw-p 00000000 00:00 0          [stack]
7fff310fc000-7fff310fe000 r-xp 00000000 00:00 0          [vdso]
ffffffffffff600000-ffffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]
```



§ La mejor forma de defenderse es no dar opciones al enemigo: **Escribir código SIN FALLOS.**


§ Esto es IMPOSIBLE:



- ◆ Existen tipos de fallos que todavía no se han inventado, por lo que es imposible saber que se está cometiendo un fallo.
- ◆ La complejidad de los sistemas hace que no se pueda comprender el funcionamiento global y todas sus interacciones.
- ◆ Poca gente es infalible:
 - Frabrice Bellard
 - ¿?

§ Pero bueno, una buena metodología siempre ayuda.



- § Existen multitud de estándares de estilo para programar.
 - ◆ Busca el estándar del sector en el que trabajas o adapta alguno existente a tus necesidades.
- § La programación no se limita a escribir código, sino que hay que mantenerlo → leerlo, versionarlo, documentarlo, etc.
- § Todos los que trabajen en un proyecto DEBEN utilizar un estilo de programación similar. 
- § Aunque existen herramientas de autoformateado, es importante que el programador lo haga de forma explícita.



§ Uno de los estándares de referencia es el MISRA-C:

- ◆ *"MISRA C is a set of software development guidelines for the C programming language developed by MISRA (Motor Industry Software Reliability Association)" [WP]*

§ Este estándar trata de concretar conceptos generales en reglas que se aplican a la programación en "C".

§ Ejemplos de reglas:

- ◆ No se deben usar más de dos niveles de indirección de punteros.
- ◆ Se debe definir un estilo para los identificadores.
- ◆ Los tipos estándar de "C" no se deben utilizar.
- ◆ No se deben utilizar funciones con un número de parámetros variable



- § Los nombres de las variables, tipos, macros, funciones y ficheros deben ser significativos y homogéneos
- § Todo el proyecto debe seguir la misma forma.
- § Algunos programas de chequeo de estilo consideran un error un identificador mal formado.
- § Puesto que los identificadores se construyen a partir de frases, existen dos formas principales de nombrado:

https://en.wikipedia.org/wiki/Naming_convention_%28programming%29

 - ◆ Camel Case: EstoEsUnContador
 - ◆ Underscore: esto_es_un_contador
- § Ninguno es mejor que el otro. Pero no se deben mezclar.



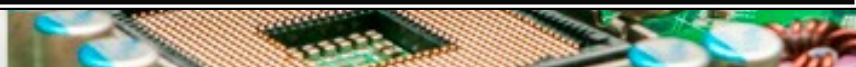
- § La indentación y la forma de alinear las estructuras de control es fundamental.
- § Es tan importante, que el famoso lenguaje de programación Python le da a la indentación significado sintáctico.
- § La mayoría de guías indentan a 2 o 4 espacios. Linux indenta a 8.
- § *"The Linux kernel has officially deprecated its coding style that the length of lines of code comply with 80 columns as the 'strong preferred limit'," (2020).*



If you need more than 3 levels of indentation, you're screwed anyway, and should fix your program.

(Linus Torvalds)

izquotes.com



- § Utilizar IDEs de programación puede ser inevitable en algunos entornos.
- § Pero se deben evitar puesto que nos hacen dependientes de tecnologías que SEGURO pasarán de moda.
- § Es preferible aprender herramientas concretas más que “frameworks” que luego no podemos actualizar.
- § Se puede saber si un programador es un profesional si no puede pasar sin su editor preferido.



- § Es básico conocer el funcionamiento de los sistemas de repositorio de software.
- § Permiten que varios desarrolladores trabajen en paralelo en un único proyecto.
- § Permiten el versionado de código.
- § Mantienen un histórico identificado por fecha y autor de todo lo que se hace.
 - ◆ En los proyectos de alta integridad esta característica es indispensable para poder asignar responsabilidades en caso de fallos.
 - ◆ Permite rastrear los orígenes de los fallos.



§ Existe toda una serie de recomendaciones genéricas e instanciadas a lenguajes concretos que se deben estudiar y repasar regularmente.

- ◆ La lista de debilidades: CWE.
- ◆ O las listas de la OWASP.

https://www.owasp.org/images/0/08/OWASP_SCP_Quick_Reference_Guide_v2.pdf

§ Estudiar las estrategias denominadas “defensive programming”:

- ◆ Principio de mínimo privilegio.
- ◆ Validación de todas las entradas.
- ◆ Comprobar los códigos de error de las funciones.
- ◆ Cuidado con la criptografía!!!
- ◆



- § https://en.wikipedia.org/wiki/Buffer_overflow_protection
- § <https://www.cert.org/secure-coding/research/secure-coding-standards.cfm>
- § https://www.owasp.org/index.php/OWASP_Top_Ten_Cheat_Sheet
- § @google "SEI CERT C Coding Standard"

