COMO PENETRAR UN SISTEMA POR MEDIO DE DESBORDAMIENTO DE BUFFER

OBJETIVO

Entender los aspectos técnicos, por los cuales un intruso puede ingresar a un sistema y tomar el control de este

AGENDA

- Programas
- Memoria
- Pila
- Funciones
- Etapas de invocación
- Arreglos y buffers
- Desbordamiento de buffer
- Impacto
- Tecnicas de protección

PROGRAMAS

- Lenguaje de alto nivel
  - Compilador
- Lenguaje ensamblador
  - Ensamblador
    - Mnemonics
    - Opcode
- Lenguaje de maquina

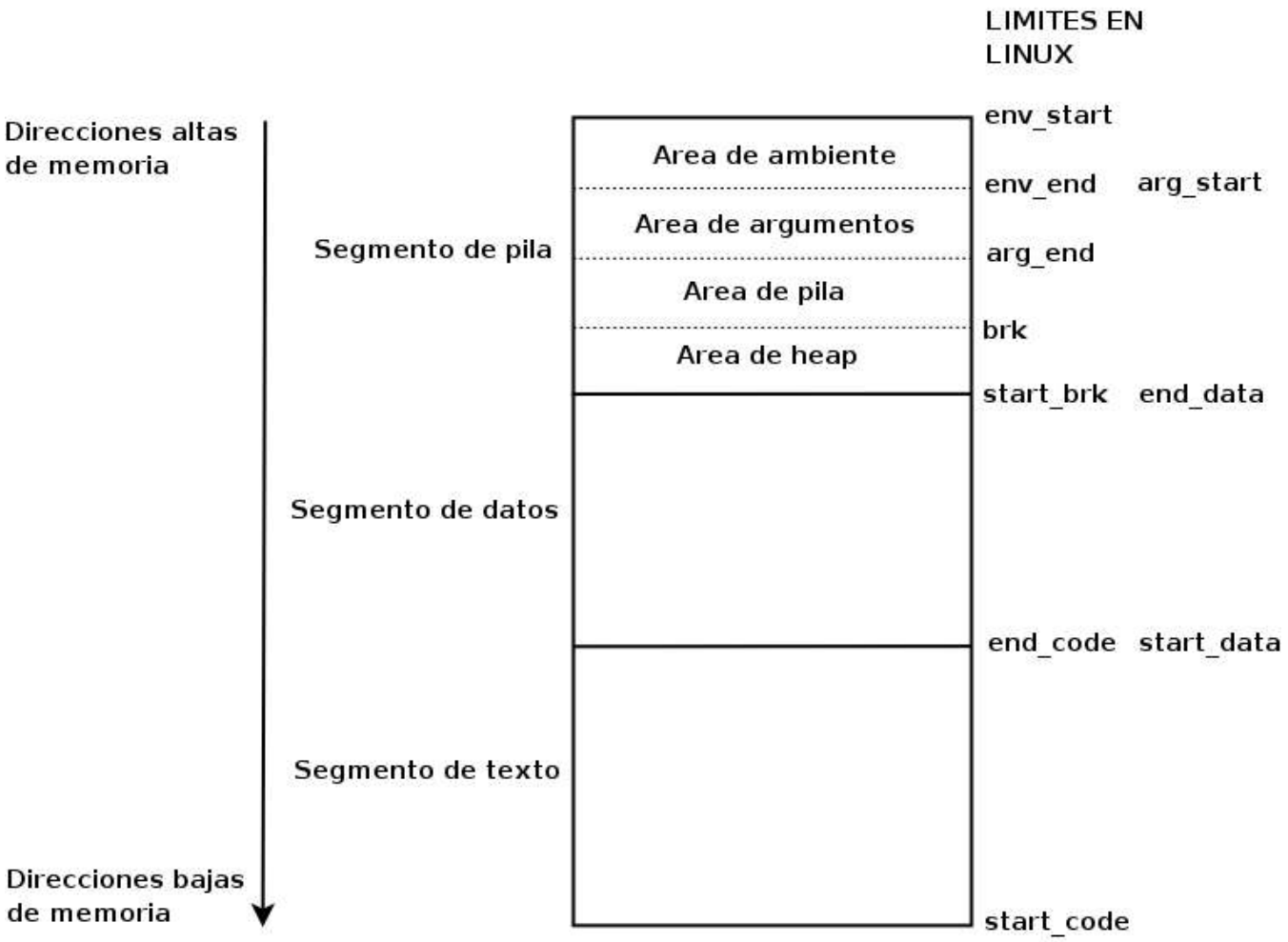
ORGANIZACIÓN DE UN PROCESO EN MEMORIA

- Segmento de texto
  - Código
- Segmento de datos
  - Variables globales
  - Variables estáticas
- Segmento de pila

AREAS DE LA PILA

- Heap
  - Memoria dinámica
- Pila
  - Direcciones de retorno
  - Variables locales
  - Parámetros
- Argumentos
- Ambiente

MEMORIA


LIMITES EN
LINUX
Direcciones altas
de memoria
env_start
Area de ambiente
env_end
arg_start
Area de argumentos
Segmento de pila
arg_end
Area de pila
brk
Area de heap
start_brk
end_data
Segmento de datos
end_code
start_data
Segmento de texto
Direcciones bajas
de memoria
start_code

SEPARATION OF CONCERNS

- Edsger Dijkstra
- Programación Estructurada
  - **Funciones, Procedimientos**
- Programación Orientada a Objetos
  - Objetos
- Programación Orientada a Aspectos
  - Aspecto

INVOCACIÓN DE FUNCIONES

- ¿Que problemas hay con las funciones?
  - Almacenamiento temporal de variables
    - Solo deben existir por invocación
  - Arbol de invocaciones desconocido en tiempo de compilación
  - Devolver el control a la función invocadora

ETAPAS DE INVOCACIÓN

- Etapa 1
  - Invocación
    - Almacenamiento de parámetros
    - Almacenamiento de dirección de retorno
- Etapa 2
  - Prologo
    - Almacenamiento de dirección de variables anteriores
    - Separar espacio para almacenar variables locales
- Etapa 3
  - Epilogo
    - Disponibilidad de variables locales anteriores
    - Retornar el control a la función anterior

EJEMPLO

```
unsigned int factorial(unsigned int n)
{
	unsigned int resultado;
	if (n == 0)
	{
		return 1;
	}
	resultado = n;
	while (n > 1)
	{
		resultado *= --n;
	};
	return resultado;
}
```

```
int main()
{
	unsigned int resultado = factorial(3);
}
```

OPERACIONES CON LA PILA

- push
- pop
- Ejemplo
  - push 1 [1]
  - push 4 [1,4]
  - push 3 [1,4,3]
  - pop [1,4]
  - pop [1]

EJEMPLO

```
(gdb) disassemble main
Dump of assembler code for function main:
0x08048391 <main+0>:    push   %ebp
0x08048392 <main+1>:    mov    %esp,%ebp
0x08048394 <main+3>:    sub    $0x8,%esp
0x08048397 <main+6>:    and    $0xfffffff0,%esp
0x0804839a <main+9>:    mov    $0x0,%eax
0x0804839f <main+14>:   sub    %eax,%esp
0x080483a1 <main+16>:   movl   $0x3,(%esp)
0x080483a8 <main+23>:   call   0x8048354 <factorial>
0x080483ad <main+28>:   mov    %eax,0xfffffffc(%ebp)
0x080483b0 <main+31>:   leave
0x080483b1 <main+32>:   ret
0x080483b2 <main+33>:   nop
```

EJEMPLO

```
(gdb) disas factorial
Dump of assembler code for function factorial:
0x08048354 <factorial+0>:     push   %ebp
0x08048355 <factorial+1>:     mov    %esp,%ebp
0x08048357 <factorial+3>:     sub    $0x8,%esp
0x0804835a <factorial+6>:     cmpl   $0x0,0x8(%ebp)
...
0x08048384 <factorial+48>:     jmp    0x804836f <factorial+27>
0x08048386 <factorial+50>:     mov    0xfffffffc(%ebp),%eax
0x08048389 <factorial+53>:     mov    %eax,0xfffffff8(%ebp)
0x0804838c <factorial+56>:     mov    0xfffffff8(%ebp),%eax
0x0804838f <factorial+59>:     leave
0x08048390 <factorial+60>:     ret
End of assembler dump.
```

¿QUE SE ALMACENA EN LA PILA?

- Parámetros
- Dirección de retorno
- Dirección de las variables anteriores
- Variables locales

ARREGLOS

- Agrupación
- Consecutiva
- Mismo tipo

- Recorrido por medio de
  - Apuntadores
  - ¿Referencias?

- ¿Verificación de limites?

DESBORDAMIENTO DE BUFFER

Escritura sobre posiciones adyacentes al buffer, es decir, fuera de los limites

EJEMPLO

```
static char global_buffer_1[10];
static char global_buffer_2[10];

int main () {
  int i;
  for (i=0; i<12; i++)  {
    global_buffer_1[i] = 'A';
  }
  printf ("Address of global_buffer_1: 0x%x\n", global_buffer_1);
  printf ("Content of global_buffer_1: \"%s\"\n", global_buffer_1);
  printf ("Address of global_buffer_2: 0x%x\n", global_buffer_2);
  printf ("Content of global_buffer_2: \"%s\"\n", global_buffer_2);
}
```

EJEMPLO

Address of global_buffer_1: 0x804972c
Content of global_buffer_1: "AAAAAAAAAAAAA"
Address of global_buffer_2: 0x8049736
Content of global_buffer_2: "AA"

PREGUNTAS

- ¿Donde están las variables globales?
- ¿Donde están las variables locales?
- ¿Que pasaría si el buffer que se desborda fuera local?
- ¿Que se sobreescribiria?

RESPUESTA

LA DIRECCIÓN DE RETORNO

¿y esto que significa?

RESPUESTA

Desbordando un buffer puedo cambiar el flujo de ejecución de un programa

¿pero a donde?

RESPUESTA

A instrucciones maliciosas que se encuentran en el mismo buffer

¿y cual es el impacto?

RESPUESTA

Se pueden ejecutar instrucciones arbitrarias dentro de un proceso

TECNICAS DE PROTECCIÓN

- Segmentos de pila no ejecutables
- Verificación de limites
- Aleatorización de direcciones de memoria

Nombre: Juan Rafael Álvarez Correa
Email: juan.alvarez@fluidsignal.com
Web: http://people.fluidsignal.com/~jalvarez
Telefono celular: 3006551750
Telefono oficina: 574-3522627