jtsec

BEYOND IT SECURITY

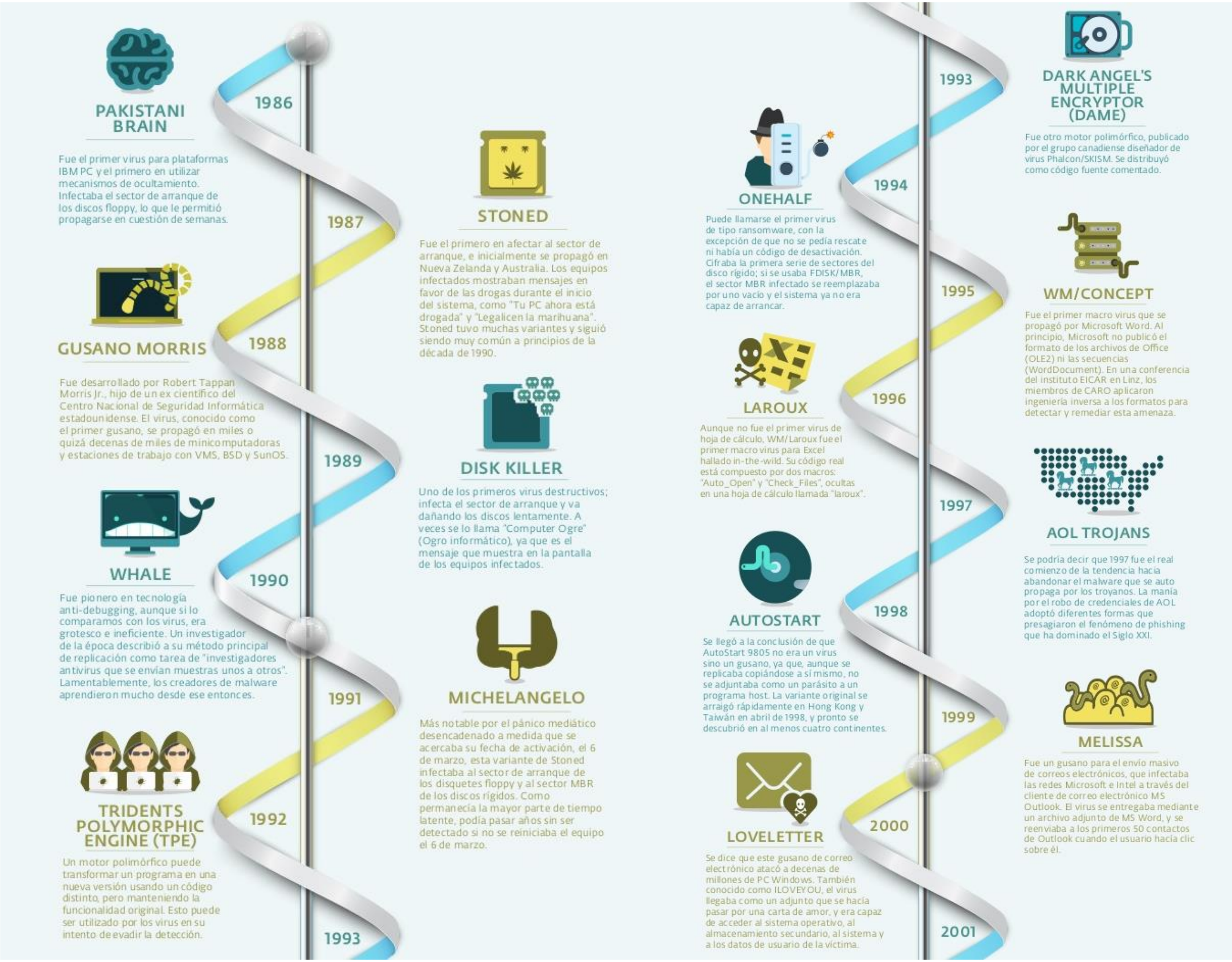
Análisis de Malware

Índice

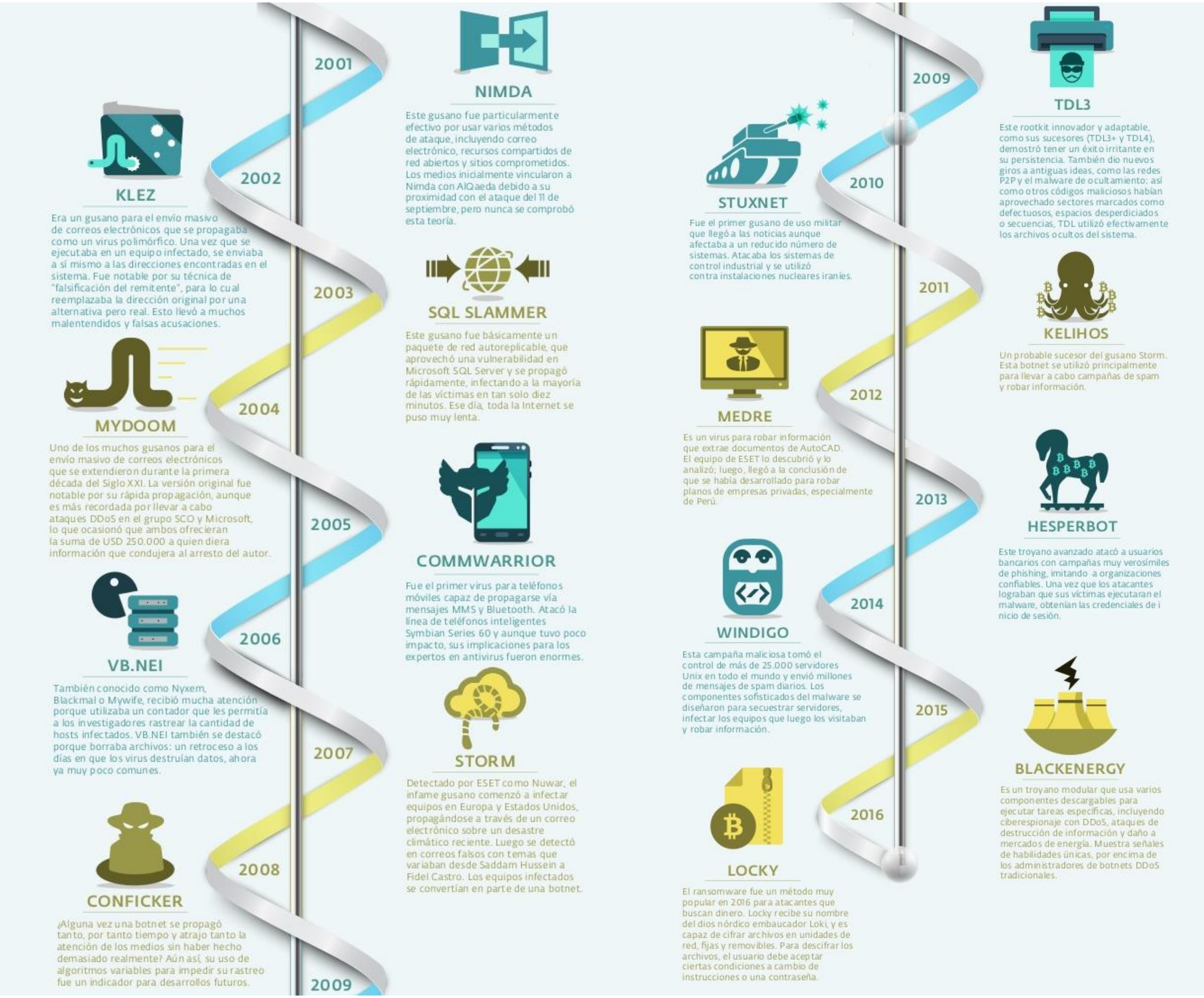
Introducción e historia

- Terminología
  - Bomba lógica
  - Troyano
  - Backdoor (Botnet & RATs & CC)
  - Virus
  - Worm
  - Rabbit
  - Spyware
  - Adware
  - Downloader / Launcher
  - Rootkit
  - Spammer
  - Scareware

Introducción e historia



Introducción e historia

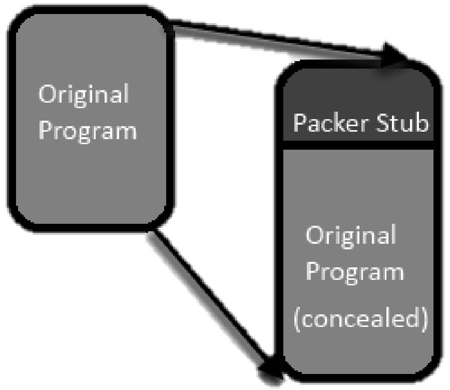


Introducción e historia

1. Introducción e historia
2. **Análisis estático básico**
3. Análisis dinámico básico
4. Análisis estático avanzado

Análisis estático básico

- VirusTotal
- Hashing
- Strings
    - Unicode
- Cabeceras
- Análisis de Packer (PEiD)



Análisis estático básico

- Bibliotecas enlazadas
  - Estáticas
  - Dinámicas (imports) → Dependency Walker
  - En tiempo de ejecución
    - LoadLibrary
    - GetProcAddress

Análisis estático básico

- Librerías comunes de Windows
  - Kernel32.dll
  - Advapi32.dll
  - User32.dll
  - Gdi32.dll
  - Ntdll.dll
  - Wsock32.dll / Ws2_32.dll
  - Wininet.dll

Análisis estático básico

- Nombres de funciones
  - CreateWindow / CreateWindowEx
  - CreateDirectory
    - CreateDirectoryA
    - CreateDirectoryW

Análisis estático básico

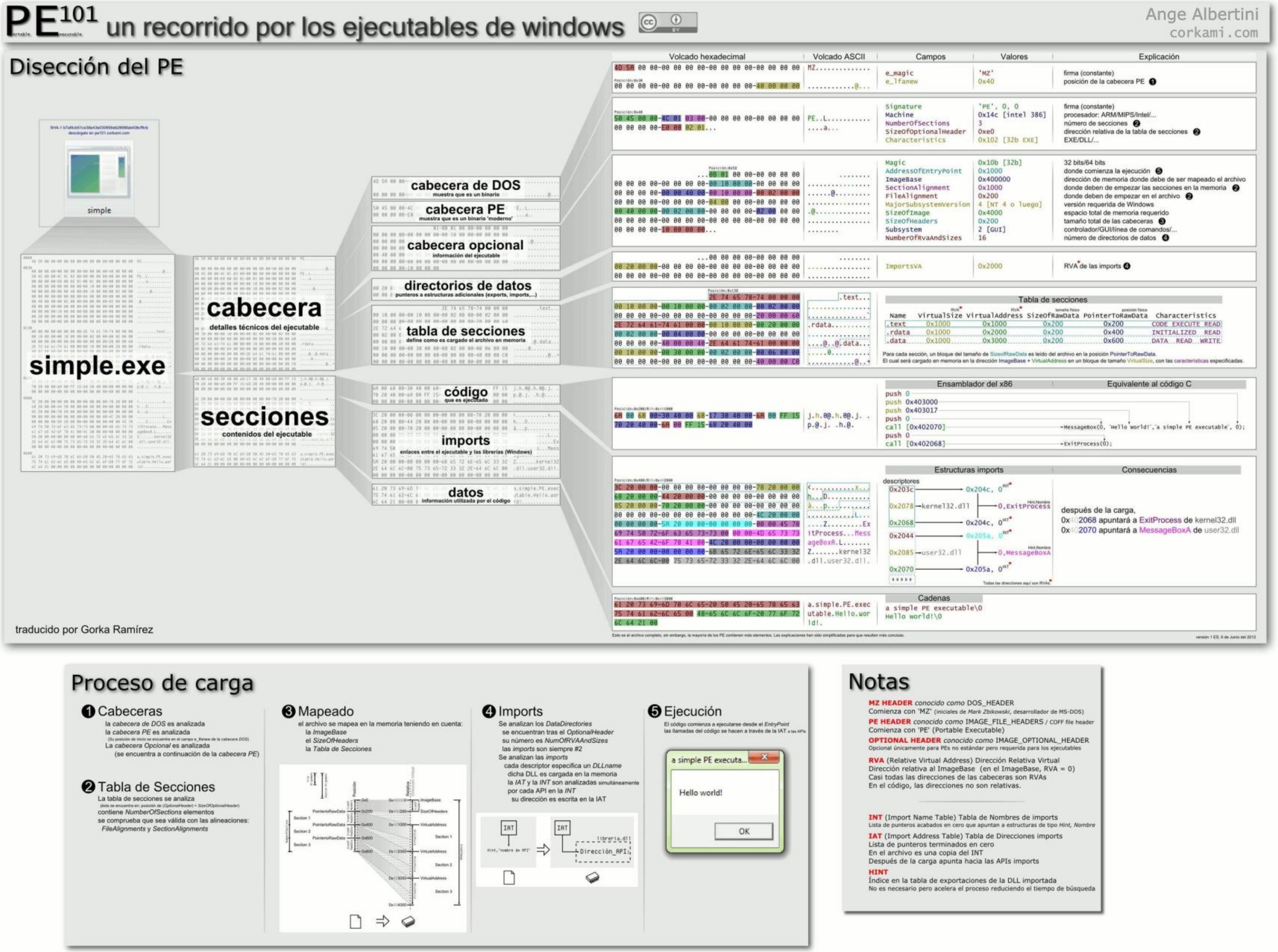
- Ficheros PE
  - IMAGE_DOS_HEADER → Historical
  - MS-DOS Stub Program → Historical
  - IMAGE_NT_HEADERS
    - IMAGE_FILE_HEADER → Información sobre compilación
    - IMAGE_OPTIONAL_HEADER →
      - Subsystem description: GUI / Consola
  - IMAGE_SECTION_HEADER → Secciones del ejecutable, nombres elegidos por el compilador
    - Virtual Size: lo que ocupa en memoria
    - Size of Raw data: lo que ocupa en disco

Análisis estático básico

- Ficheros PE

Nombre	Descripción
.text / .code	Código ejecutable
.rdata / .const	Datos read-only
.data	Datos
.idata	Funciones importadas (a veces se omite y están en .rdata)
.edata	Funciones exportadas (a veces se omite y están en .rdata)
.pdata	Información de EH en 64 bits
.rsrc	Resources
.reloc	Información para relocar bibliotecas
.bss	Uninitialized data
…	…

Análisis estático básico



Análisis estático básico

- Resources
  - Iconos
  - Menús
  - Diálogos
  - Cadenas

Análisis estático básico

- ¡Probemos! Práctica, 3.1

Índice

1. Introducción e historia
2. Análisis estático básico
3. **Análisis dinámico básico**
4. Análisis estático avanzado

Análisis dinámico básico

- Máquinas virtuales
  - Pros
    - Aisladas
    - Snapshots
    - Record/Replay
  - Contras
    - Detectables)
    - Vulnerabilidades

Análisis dinámico básico

- Warning! No sabemos qué tenemos entre manos y vamos a ejecutar cosas! Usa un entorno aislado!

Análisis dinámico básico

- Sandboxes
  - Hybrid Analysis
    - https://www.hybrid-analysis.com/
  - Any Run
    - https://app.any.run/
    - https://www.threattrack.com/online-malware-sandbox.aspx
  - Cuckoo / VMCloak
    - https://cuckoo.cert.ee/
- Online / Offline

Análisis dinámico básico

- Sandboxes, problemas
  - Parámetros
  - Funciones de arranque
  - Detección de VM
  - Malware en DLLs

Análisis dinámico básico

- Malware en DLLs

  - rundll32.exe DLLname, Export arguments
  - rundll32.exe xyzzy.dll, #5
  - rundll32 ipr32x.dll,InstallService ServiceName

    net start ServiceName

Análisis dinámico básico

- Process Monitor
  - Registro
  - Filesystem
  - Procesos y hebras
  - Red (no recomendado)

Análisis dinámico básico

- Process Explorer
  - Task manager
  - Verifica si DLL está firmada por Microsoft en runtime
  - Comprobar strings en disco y en memoria
    - Algunas diferencias son normales
  - Análisis rápido de malware en documentos

Análisis dinámico básico

- Regshot
  - Permite hacer dos snapshots del registro y compararlas

Análisis dinámico básico

- Comportamiento de red
  - Fakenet / fakenet-ng
    - Redirige el tráfico DNS/HTTP/SSL
    - Inteligente, si piden jpg sirve jpg
    - Saca un pcap
    - Muy flexible

Análisis dinámico básico

- Comportamiento de red
  - Wireshark
  - InetSim (Linux)
  - Nc
  - ApateDNS

Análisis dinámico básico

- ¡Probemos! Práctica, 3.2

Índice

1. Introducción e historia
2. Análisis estático básico
3. Análisis dinámico básico
4. **Análisis estático avanzado**

Análisis estático avanzado

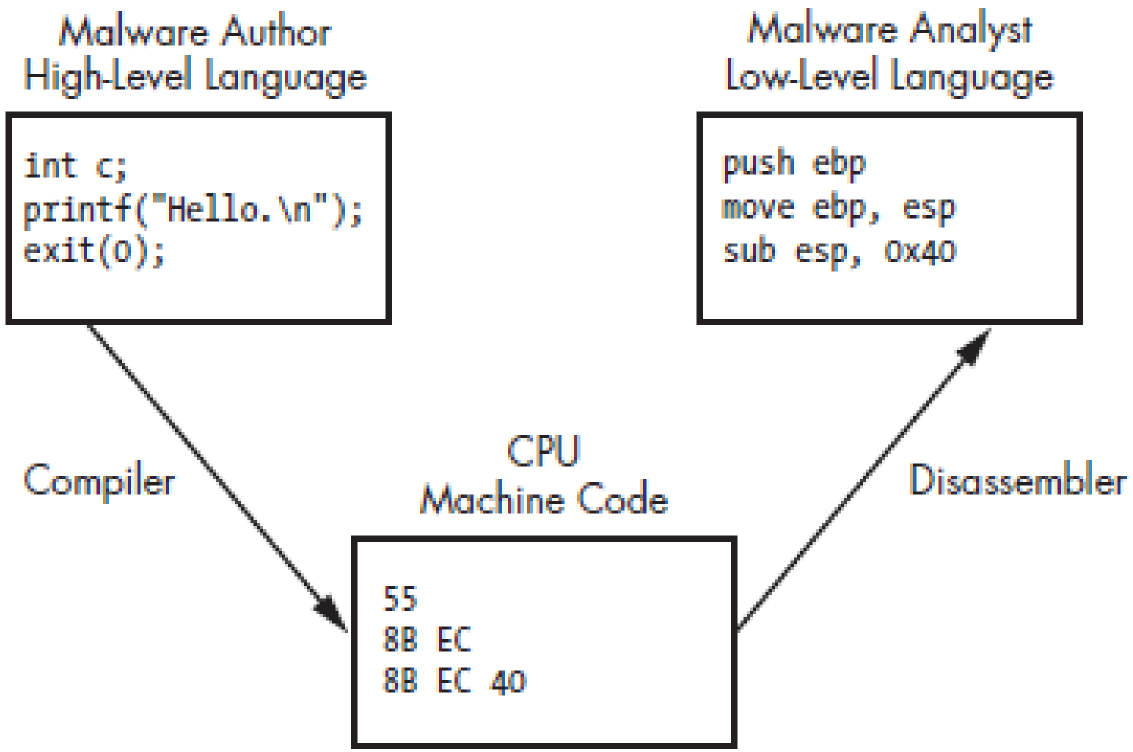
- El análisis estático básico nos permite saber qué funciones se importan, pero no si se usan ni como se usan.
- El análisis dinámico básico nos puede dar una visión a nivel de interfaz, pero se acaba ahí

Análisis estático avanzado

- Niveles de abstracción en programación
  - Hardware (lógica digital)
  - Microcódigo / Firmware → 10101001…
  - Código máquina (opcodes) → 26h32h…
  - Lenguajes de bajo nivel (ensamblador)
  - Lenguajes de alto nivel (C, C++, …)
  - Lenguajes interpretados (bytecode)
- ¿En qué nivel trabaja el analista de malware?

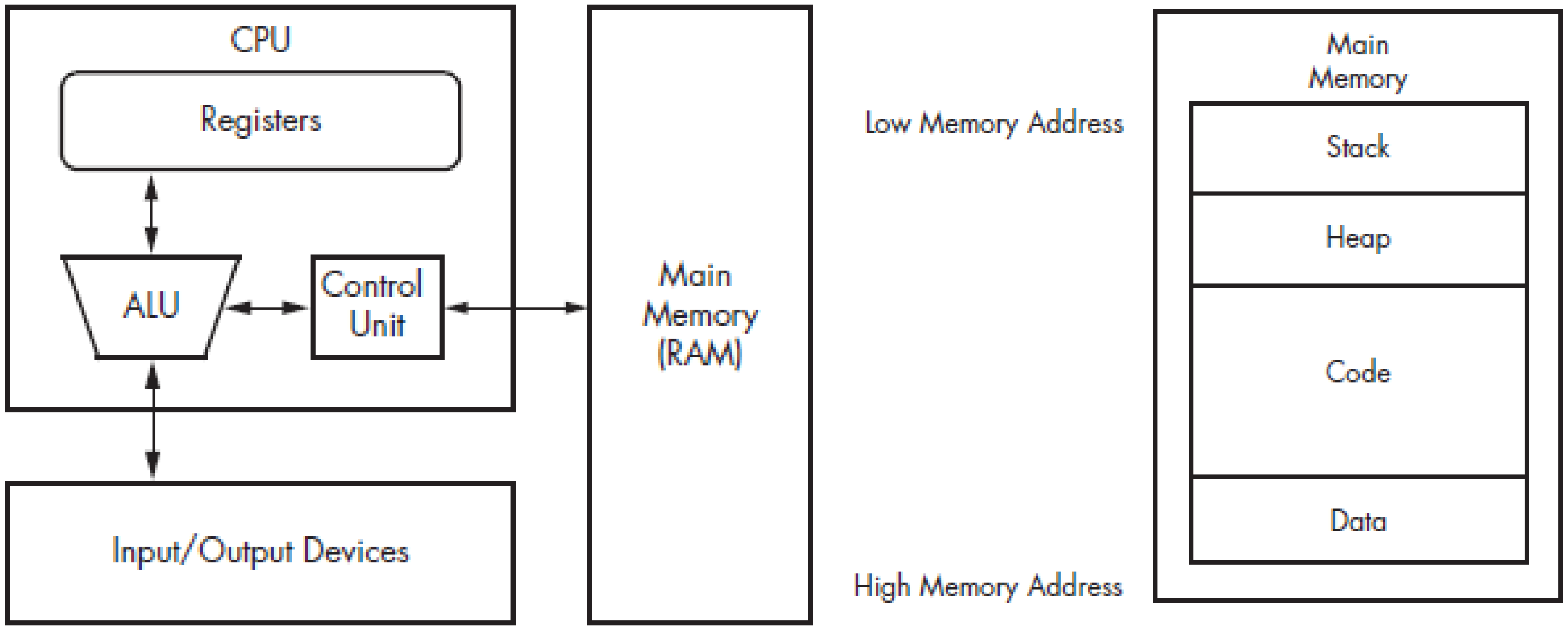
Análisis estático avanzado

- ¿En qué nivel trabaja el analista de malware?



Análisis estático avanzado

- Arquitectura X86



Análisis estático avanzado

- Instrucciones ASM

  - Mnemonic
  - Mnemonic [destination]
  - Mnemonic [destination], [source]
  - Mnemonic [destination], [source1], [source2]

Análisis estático avanzado

- Disassembler

B9 42 00 00 00 → mov ecx, 0x42

Análisis estático avanzado

- Tipos de operando

    - Inmediato: 0x42
    - Registro: ecx
    - Dirección de memoria: [ecx]

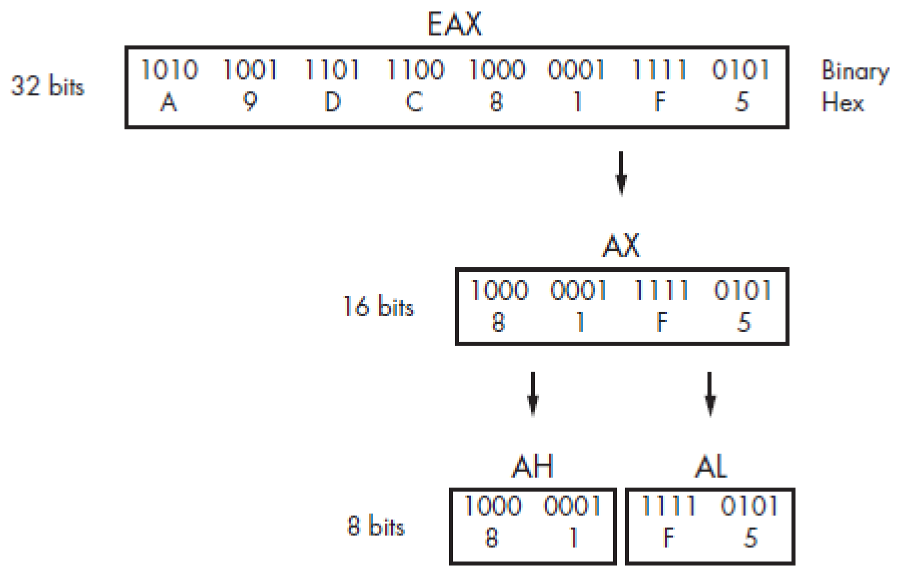
Análisis estático avanzado

- Registros

General registers	Segment registers	Status register	Instruction pointer
EAX (AX, AH, AL)	CS	EFLAGS	EIP
EBX (BX, BH, BL)	SS		
ECX (CX, CH, CL)	DS		
EDX (DX, DH, DL)	ES		
EBP (BP)	FS		
ESP (SP)	GS		
ESI (SI)			

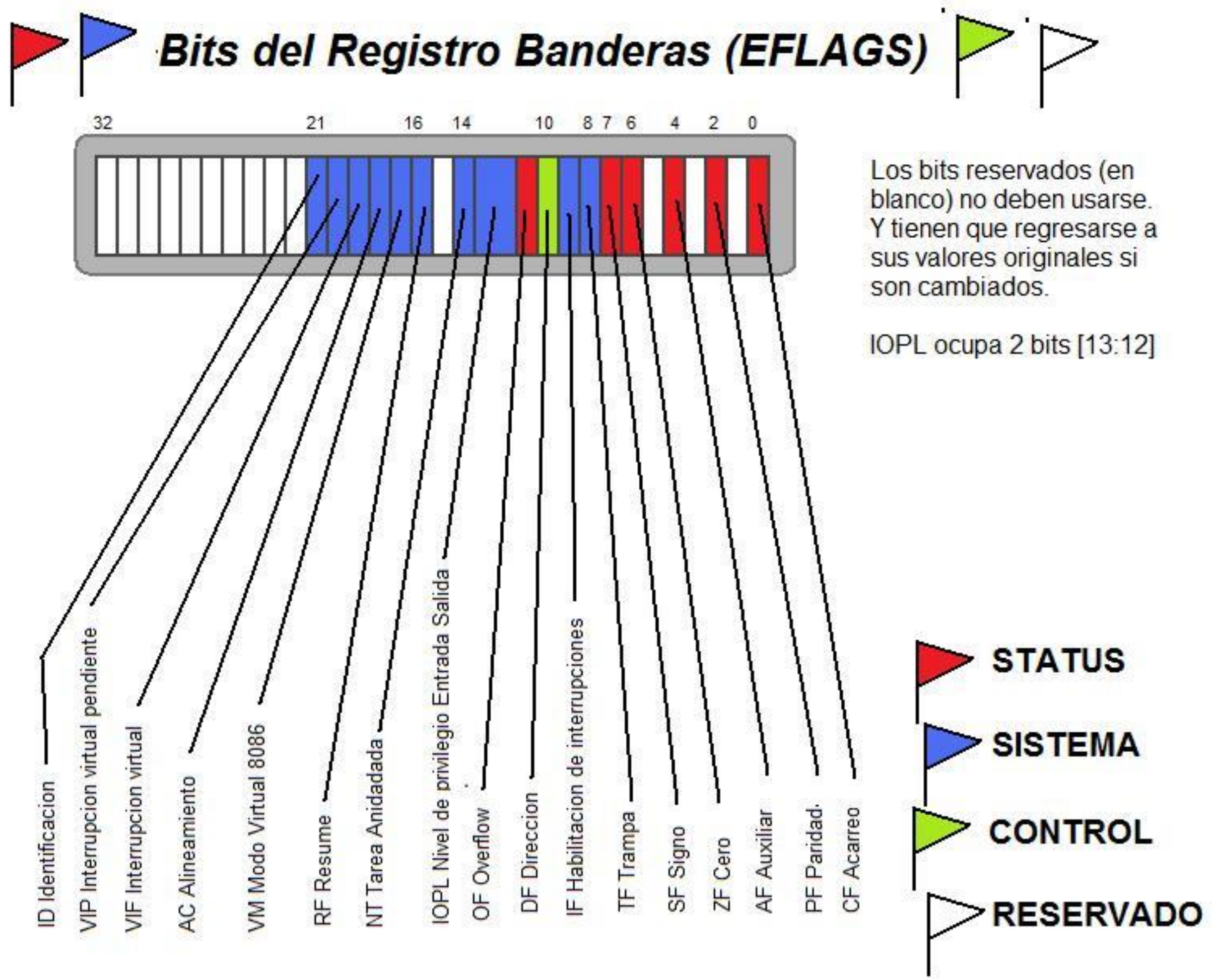
Análisis estático avanzado

- Registros



Análisis estático avanzado

- EFLAGS



Análisis estático avanzado

- EFLAGS
  - ZF: Zero Flag
  - CF: Carry Flag
  - SF: Sign Flag
  - TF: Trap Flag, Modo single step

- Instrucciones

Instruction	Description
`mov eax, ebx`	Copies the contents of EBX into the EAX register
`mov eax, 0x42`	Copies the value 0x42 into the EAX register
`mov eax, [0x4037C4]`	Copies the 4 bytes at the memory location 0x4037C4 into the EAX register
`mov eax, [ebx]`	Copies the 4 bytes at the memory location specified by the EBX register into the EAX register
`mov eax, [ebx+esi*4]`	Copies the 4 bytes at the memory location specified by the result of the equation `ebx+esi*4` into the EAX register

Análisis estático avanzado

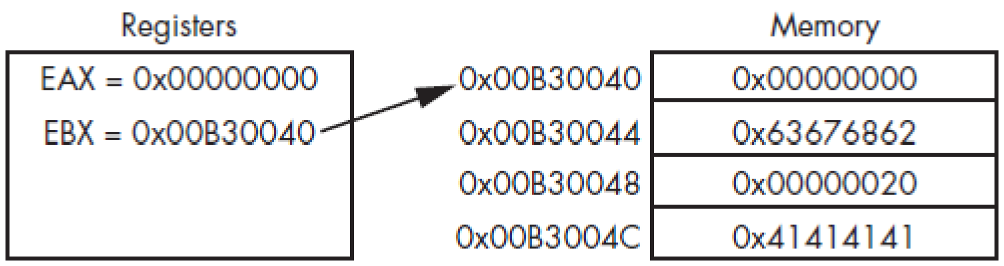
- Instrucciones

Instruction	Description
mov eax, ebx	Copies the contents of EBX into the EAX register
mov eax, 0x42	Copies the value 0x42 into the EAX register
mov eax, [0x4037C4]	Copies the 4 bytes at the memory location 0x4037C4 into the EAX register
mov eax, [ebx]	Copies the 4 bytes at the memory location specified by the EBX register into the EAX register
mov eax, [ebx+esi*4]	Copies the 4 bytes at the memory location specified by the result of the equation ebx+esi*4 into the EAX register

- ~~mov eax, ebx+8~~ → lea eax, [ebx+8]

Análisis estático avanzado

- Instrucciones



- mov eax, [ebx+8] → eax = ?
- lea eax, [ebx+8] → eax = ?

Análisis estático avanzado

- Instrucciones

Instruction	Description
`sub eax, 0x10`	Subtracts 0x10 from EAX
`add eax, ebx`	Adds EBX to EAX and stores the result in EAX
`inc edx`	Increments EDX by 1
`dec ecx`	Decrements ECX by 1

Instruction	Description
`mul 0x50`	Multiplies EAX by 0x50 and stores the result in EDX:EAX
`div 0x75`	Divides EDX:EAX by 0x75 and stores the result in EAX and the remainder in EDX

Instruction	Description
`xor eax, eax`	Clears the EAX register
`or eax, 0x7575`	Performs the logical or operation on EAX with 0x7575
`mov eax, 0xA` `shl eax, 2`	Shifts the EAX register to the left 2 bits; these two instructions result in EAX = 0x28, because 1010 (0xA in binary) shifted 2 bits left is 101000 (0x28)
`mov bl, 0xA` `ror bl, 2`	Rotates the BL register to the right 2 bits; these two instructions result in BL = 10000010, because 1010 rotated 2 bits right is 10000010

Análisis estático avanzado

- La pila

  - ESP: Stack pointer (top)
  - EBP: Base pointer, comienzo de parámetros de función y variables locales (suelen referenciarse con respecto a ebp)

  - push, pop, pusha, popa, pushad, popad
  - enter / leave
  - call / ret

Análisis estático avanzado

- Convenciones
  - Cosas que un compilador hace siempre igual
  - Uso de los registros
  - Forma de llamar a una función

  - cdecl, stdcall, fastcall …

Análisis estático avanzado

- cdecl
  - Visual C++ y gcc
  - Return en eax
  - Argumentos RTL por pila
  - Caller stack cleanup
  - Registers EAX, ECX, and EDX are caller-saved, and the rest are callee-saved (esi, edi, ebx, ebp).

Análisis estático avanzado

- stdcall
  - Microsoft Win32 API
  - Argumentos RTL por pila
  - Callee stack cleanup

Análisis estático avanzado

- La pila: llamadas a funciones
  - Prólogo

```
push   ebp            ; Save the stack-frame base pointer (of the calling function).
mov    ebp, esp       ; Set the stack-frame base pointer to be the current
                      ; location on the stack.
sub    esp, N         ; Grow the stack by N bytes to reserve space for local variables
```

  - En este momento tenemos…

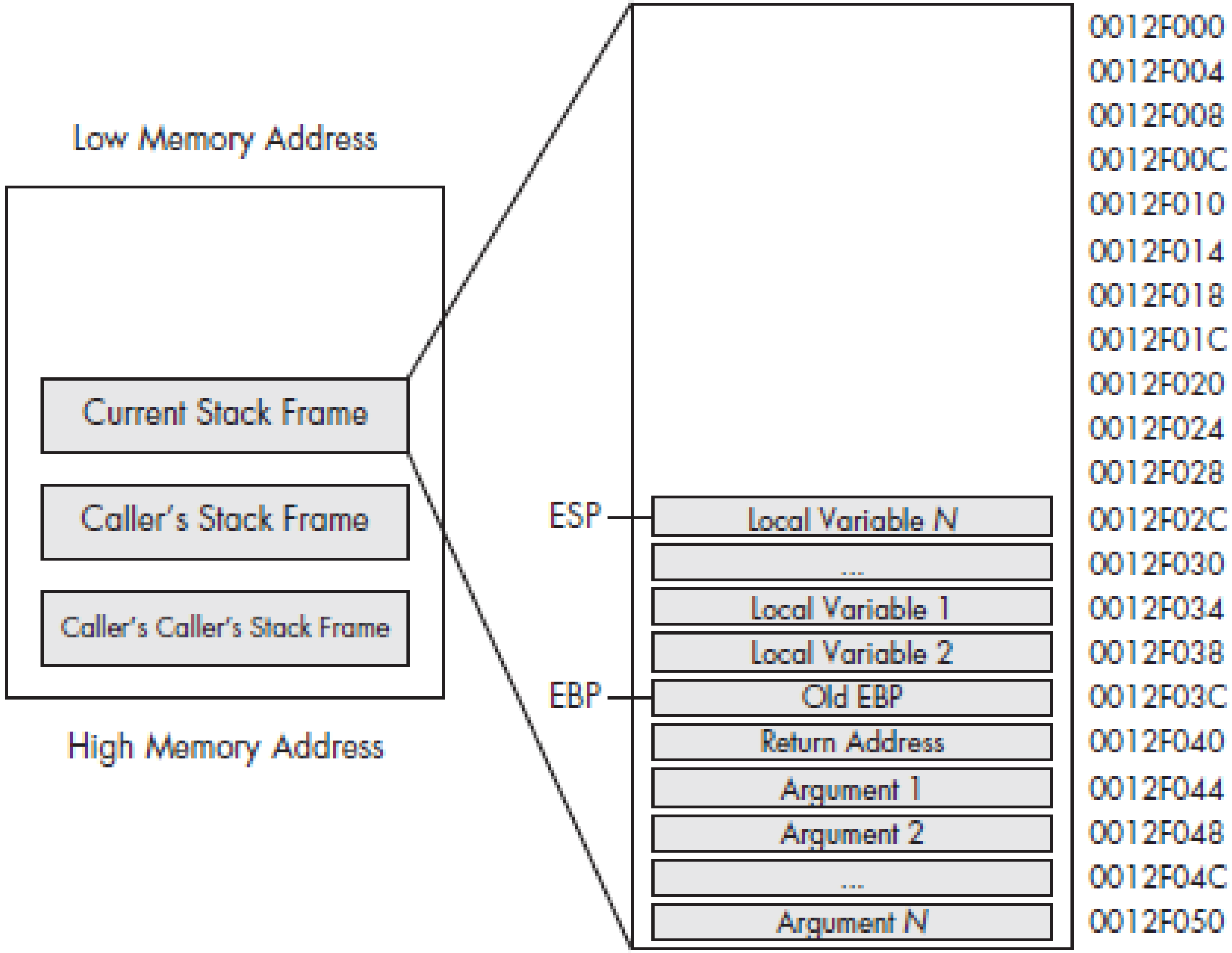
```
...
ebp + 4:     Return address
ebp + 0:     Calling function's old ebp value
ebp - 4:     (local variables)
...
```

  - Epílogo

```
mov    esp, ebp       ; Put the stack pointer back where it was when this function
                      ; was called.
pop    ebp            ; Restore the calling function's stack frame.
ret                   ; Return to the calling function.
```

Análisis estático avanzado

- La pila: disposición



Análisis estático avanzado

- La pila: llamadas a funciones

```c
int callee(int, int, int);

int caller(void)
{
    return callee(1, 2, 3) + 5;
}
```

```asm
caller:
    ; make new call frame
    push    ebp
    mov     ebp, esp
    ; push call arguments
    push    3
    push    2
    push    1
    ; call subroutine 'callee'
    call    callee
    ; remove arguments from frame
    add     esp, 12
    ; use subroutine result
    add     eax, 5
    ; restore old call frame
    pop     ebp
    ; return
    ret
```

Análisis estático avanzado

- La pila: llamadas a funciones

b	
a	
return address	← esp

add: # prologo
push ebp

eip →

```
#include <stdio.h>

int add(int a, int b);

int main(){
  int a = 10;
  int b = 6;
  int sum;
  sum = add(a,b);
  printf("Sum = %d",sum);
  return(0);
}
```

```
int add(int a, int b)
    int c;
    c = a + b;
    return c;
}
```

Análisis estático avanzado

- La pila: llamadas a funciones

b	
a	
return address	
old ebp	← esp

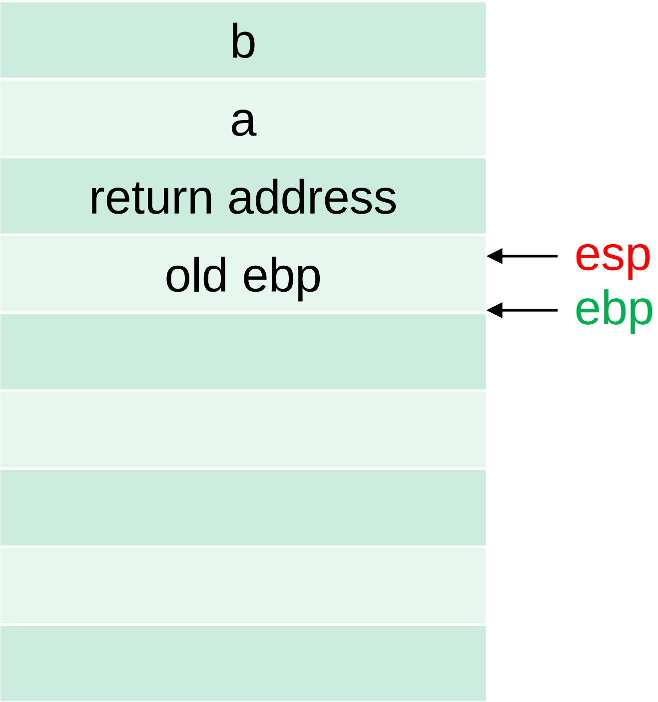
```
add: # prologo
     push ebp      ← eip
```

```c
#include <stdio.h>

int add(int a, int b);

int main(){
  int a = 10;
  int b = 6;
  int sum;
  sum = add(a,b);
  printf("Sum = %d",sum);
  return(0);
}
```

```c
int add(int a, int b)
    int c;
    c = a + b;
    return c;
}
```

Análisis estático avanzado

- La pila: llamadas a funciones



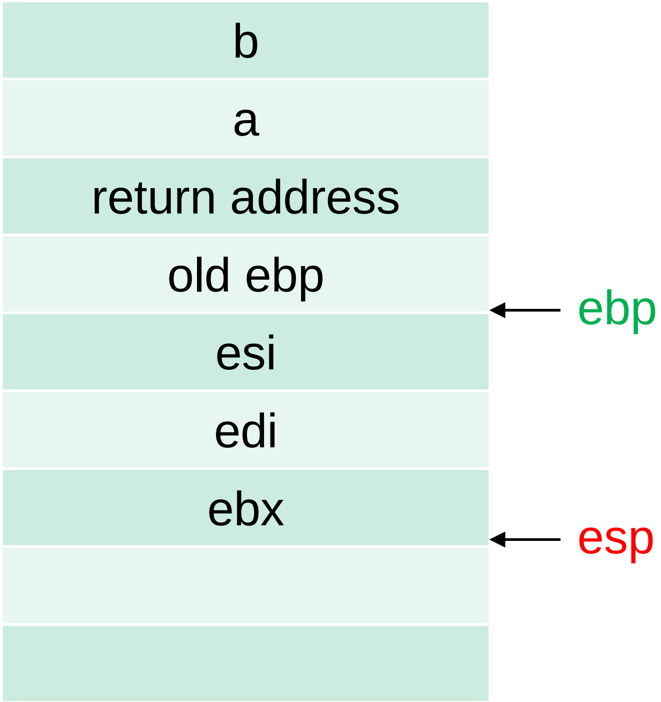

```
add: # prologo
    push ebp
    mov ebp,esp    ⟵ eip
```

```c
#include <stdio.h>

int add(int a, int b);

int main(){
  int a = 10;
  int b = 6;
  int sum;
  sum = add(a,b);
  printf("Sum = %d",sum);
  return(0);
}
```

```c
int add(int a, int b)
    int c;
    c = a + b;
    return c;
}
```

Análisis estático avanzado

- La pila: llamadas a funciones




```
add: # prologo
        push ebp
        mov ebp,esp
# guardamos registros
# según cdecl
        push esi
        push edi
        push ebx      ← eip
```

```c
#include <stdio.h>

int add(int a, int b);

int main(){
  int a = 10;
  int b = 6;
  int sum;
  sum = add(a,b);
  printf("Sum = %d",sum);
  return(0);
}
```

```c
int add(int a, int b)
    int c;
    c = a + b;
    return c;
}
```

Análisis estático avanzado

- La pila: llamadas a funciones

b	
a	
return address	
old ebp	← ebp
esi	
edi	
ebx	
c	← esp

```
add: # prologo
        push ebp
        mov ebp,esp
# guardamos registros
# según cdecl
        push esi
        push edi
        push ebx
# hacer sitio para
variable local c
        sub esp,4      ← eip
```

```c
#include <stdio.h>

int add(int a, int b);

int main(){
  int a = 10;
  int b = 6;
  int sum;
  sum = add(a,b);
  printf("Sum = %d",sum);
  return(0);
}
```

```c
int add(int a, int b)
    int c;
    c = a + b;
    return c;
}
```

Análisis estático avanzado

- La pila: llamadas a funciones

b	
a	
return address	
old ebp	← ebp
esi	
edi	
ebx	
c	← esp

```
add: # prologo
        push ebp
        mov ebp,esp
# guardamos registros
# según cdecl
        push esi
        push edi
        push ebx
# hacer sitio para
variable local c
        sub esp,4
# meto b en ebx
        mov ebx, [ebp+12]
        add ebx, [ebp+8]←  eip
```

```c
#include <stdio.h>

int add(int a, int b);

int main(){
  int a = 10;
  int b = 6;
  int sum;
  sum = add(a,b);
  printf("Sum = %d",sum);
  return(0);
}
```

```c
int add(int a, int b)
    int c;
    c = a + b;
    return c;
}
```

Análisis estático avanzado

- La pila: llamadas a funciones

b	
a	
return address	
old ebp	← ebp
esi	
edi	
ebx	
c	← esp

```
add: # prologo
        push ebp
        mov ebp,esp
# guardamos registros
# según cdecl
        push esi
        push edi
        push ebx
# hacer sitio para
variable local c
        sub esp,4
# meto b en ebx
        mov ebx, [ebp+12]
        add ebx, [ebp+8]
# meto a+b en c
        mov [ebp-16], ebx    ← eip
```

```c
#include <stdio.h>

int add(int a, int b);

int main(){
  int a = 10;
  int b = 6;
  int sum;
  sum = add(a,b);
  printf("Sum = %d",sum);
  return(0);
}
```

```c
int add(int a, int b)
    int c;
    c = a + b;
    return c;
}
```

Análisis estático avanzado

- La pila: llamadas a funciones

b	
a	
return address	
old ebp	← ebp
esi	
edi	
ebx	
c	← esp

```
add: # prologo
        push ebp
        mov ebp,esp
# guardamos registros
# según cdecl
        push esi
        push edi
        push ebx
# hacer sitio para
variable local c
        sub esp,4
# meto b en ebx
        mov ebx, [ebp+12]
        add ebx, [ebp+8]
# meto a+b en c
        mov [ebp-16], ebx
# meto c en eax
        mov eax, [ebp-16]      ← eip
```

```c
#include <stdio.h>

int add(int a, int b);

int main(){
  int a = 10;
  int b = 6;
  int sum;
  sum = add(a,b);
  printf("Sum = %d",sum);
  return(0);
}
```

```c
int add(int a, int b)
    int c;
    c = a + b;
    return c;
}
```

Análisis estático avanzado

- La pila: llamadas a funciones

b	
a	
return address	
old ebp	
esi	← ebp
edi	
ebx	← esp
c	

```
# limpiar pila
    add esp, 4   ← eip
```

```c
#include <stdio.h>

int add(int a, int b);

int main(){
  int a = 10;
  int b = 6;
  int sum;
  sum = add(a,b);
  printf("Sum = %d",sum);
  return(0);
}
```

```c
int add(int a, int b)
    int c;
    c = a + b;
    return c;
}
```

Análisis estático avanzado

- La pila: llamadas a funciones

b	
a	
return address	
old ebp	← ebp
esi	
edi	← esp
ebx	
c	

```
# limpiar pila
        add esp, 4
        pop ebx   ← eip
```

```c
#include <stdio.h>

int add(int a, int b);

int main(){
  int a = 10;
  int b = 6;
  int sum;
  sum = add(a,b);
  printf("Sum = %d",sum);
  return(0);
}
```

```c
int add(int a, int b)
    int c;
    c = a + b;
    return c;
}
```

Análisis estático avanzado

- La pila: llamadas a funciones

b	
a	
return address	
old ebp	
	← ebp
esi	← esp
edi	
ebx	
c	

```
# limpiar pila
        add esp, 4
        pop ebx
        pop edi     ← eip
```

```c
#include <stdio.h>

int add(int a, int b);

int main(){
  int a = 10;
  int b = 6;
  int sum;
  sum = add(a,b);
  printf("Sum = %d",sum);
  return(0);
}
```

```c
int add(int a, int b)
    int c;
    c = a + b;
    return c;
}
```

Análisis estático avanzado

- La pila: llamadas a funciones

b
a
return address
old ebp ← esp ← ebp
esi
edi
ebx
c

```
# limpiar pila
        add esp, 4
        pop ebx
        pop edi
        pop esi    ← eip
```

```c
#include <stdio.h>

int add(int a, int b);

int main(){
  int a = 10;
  int b = 6;
  int sum;
  sum = add(a,b);
  printf("Sum = %d",sum);
  return(0);
}
```

```c
int add(int a, int b)
    int c;
    c = a + b;
    return c;
}
```

Análisis estático avanzado

- La pila: llamadas a funciones

b	
a	
return address	
old ebp	← esp ← ebp
esi	
edi	
ebx	
c	

```
# limpiar pila
        add esp, 4
        pop ebx
        pop edi
        pop esi
# epílogo
        mov esp, ebp      ← eip
```

```c
#include <stdio.h>

int add(int a, int b);

int main(){
  int a = 10;
  int b = 6;
  int sum;
  sum = add(a,b);
  printf("Sum = %d",sum);
  return(0);
}
```

```c
int add(int a, int b)
    int c;
    c = a + b;
    return c;
}
```

Análisis estático avanzado

- La pila: llamadas a funciones

b
a
return address ← esp
old ebp
esi
edi
ebx
c

```
# limpiar pila
        add esp, 4
        pop ebx
        pop edi
        pop esi
# epílogo
        mov esp, ebp
        pop ebp    ← eip
```

```c
#include <stdio.h>

int add(int a, int b);

int main(){
  int a = 10;
  int b = 6;
  int sum;
  sum = add(a,b);
  printf("Sum = %d",sum);
  return(0);
}
```

```c
int add(int a, int b)
    int c;
    c = a + b;
    return c;
}
```

Análisis estático avanzado

- La pila: llamadas a funciones

b	
a	← esp
return address	
old ebp	
esi	
edi	
ebx	
c	

```
# limpiar pila
        add esp, 4
        pop ebx
        pop edi
        pop esi
# epílogo
        mov esp, ebp
        pop ebp
        ret         ← eip
```

```c
#include <stdio.h>

int add(int a, int b);

int main(){
  int a = 10;
  int b = 6;
  int sum;
  sum = add(a,b);        // eip →
  printf("Sum = %d",sum);
  return(0);
}
```

```c
int add(int a, int b)
    int c;
    c = a + b;
    return c;
}
```

Análisis estático avanzado

- Comparaciones

  - tst → equivale a and
  - cmp → equivale a sub

  - No alteran los operandos solo las flags

cmp dst, src	ZF	CF
dst = src	1	0
dst < src	0	1
dst > src	0	0

Análisis estático avanzado

- Branching
  - Incondicional → jmp
  - Condicional → más de 30 tipos

Instruction	Description
jz loc	Jump to specified location if ZF = 1.
jnz loc	Jump to specified location if ZF = 0.
je loc	Same as jz, but commonly used after a cmp instruction. Jump will occur if the destination operand equals the source operand.
jne loc	Same as jnz, but commonly used after a cmp. Jump will occur if the destination operand is not equal to the source operand.
jg loc	Performs signed comparison jump after a cmp if the destination operand is greater than the source operand.
jge loc	Performs signed comparison jump after a cmp if the destination operand is greater than or equal to the source operand.
ja loc	Same as jg, but an unsigned comparison is performed.
jae loc	Same as jge, but an unsigned comparison is performed.
jl loc	Performs signed comparison jump after a cmp if the destination operand is less than the source operand.
jle loc	Performs signed comparison jump after a cmp if the destination operand is less than or equal to the source operand.
jb loc	Same as jl, but an unsigned comparison is performed.
jbe loc	Same as jle, but an unsigned comparison is performed.
jo loc	Jump if the previous instruction set the overflow flag (OF = 1).
js loc	Jump if the sign flag is set (SF = 1).
jecxz loc	Jump to location if ECX = 0.

Análisis estático avanzado

- Instrucciones Rep
  - movsX, cmpsX, stosX, scasX
    - X = (b)yte, (w)ord, (d)ouble Word

  - esi → source index (++)
  - edi → destination index (++)
  - ecx → counter (--)

Depende de DF

Instruction	Description
rep	Repeat until ECX = 0
repe, repz	Repeat until ECX = 0 or ZF = 0
repne, repnz	Repeat until ECX = 0 or ZF = 1

Análisis estático avanzado

- Instrucciones Rep

Instruction	Description
`repe cmpsb`	Used to compare two data buffers. EDI and ESI must be set to the two buffer locations, and ECX must be set to the buffer length. The comparison will continue until ECX = 0 or the buffers are not equal.
`rep stosb`	Used to initialize all bytes of a buffer to a certain value. EDI will contain the buffer location, and AL must contain the initialization value. This instruction is often seen used with `xor eax, eax`.
`rep movsb`	Typically used to copy a buffer of bytes. ESI must be set to the source buffer address, EDI must be set to the destination buffer address, and ECX must contain the length to copy. Byte-by-byte copy will continue until ECX = 0.
`repne scasb`	Used for searching a data buffer for a single byte. EDI must contain the address of the buffer, AL must contain the byte you are looking for, and ECX must be set to the buffer length. The comparison will continue until ECX = 0 or until the byte is found.

Análisis estático avanzado

- El main en C

```
int main(int argc, char* argv[])
{
        if (argc != 3) {return 0;}

        if (strncmp(argv[1], "-r", 2) == 0){

            DeleteFileA(argv[2]);

        }
        return 0;
}
```

```
004113CE  cmp    [ebp+argc], 3
004113D2  jz     short loc_4113D8
004113D4  xor    eax, eax
004113D6  jmp    short loc_411414
004113D8  mov    esi, esp
004113DA  push   2                 ; MaxCount
004113DC  push   offset Str2       ; "-r"
004113E1  mov    eax, [ebp+argv]
004113E4  mov    ecx, [eax+4]
004113E7  push   ecx               ; Str1
004113E8  call   strncmp
004113F8  test   eax, eax
004113FA  jnz    short loc_411412
004113FC  mov    esi, esp
004113FE  mov    eax, [ebp+argv]
00411401  mov    ecx, [eax+8]
00411404  push   ecx               ; lpFileName
00411405  call   DeleteFileA
```

Análisis estático avanzado

- Llamadas al sistema

  - En Linux son usuales
    - eax = syscall number
    - Args → ebx, ecx, edx, esi, edi, ebp
    - int 0x80
  - http://syscalls.kernelgrok.com/

Análisis estático avanzado

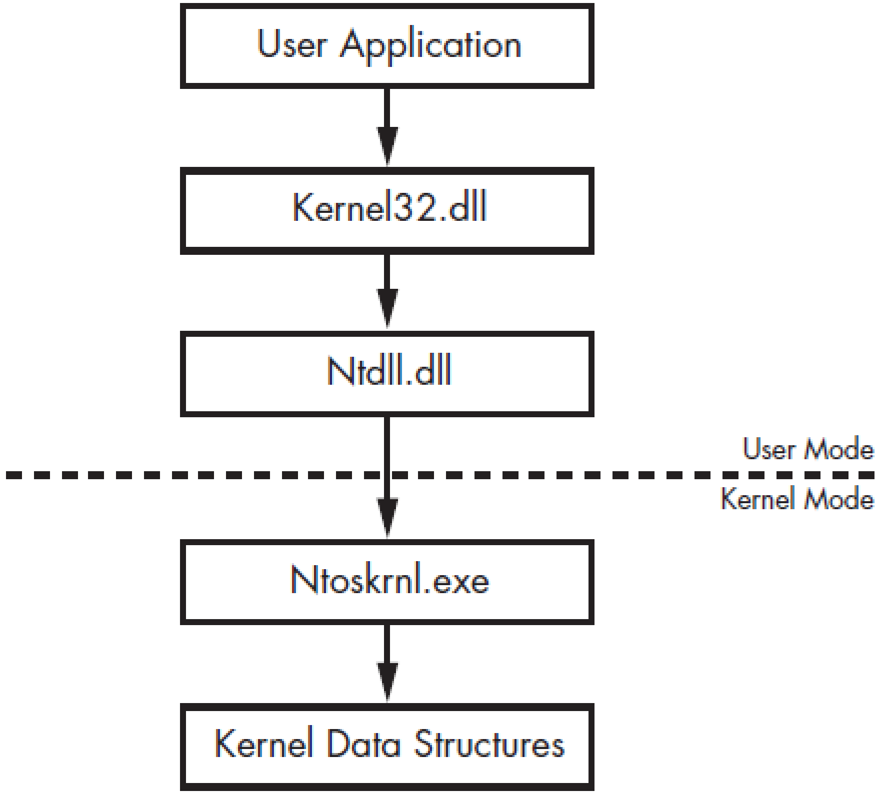
- Llamadas al sistema

  - En Windows están “escondidas” bajo la API
    - syscall, sysenter, int 0x2e

  - Para hacer ciertas cosas (rootkits) hay que programar en kernel mode (drivers) (acceso directo al hardware)

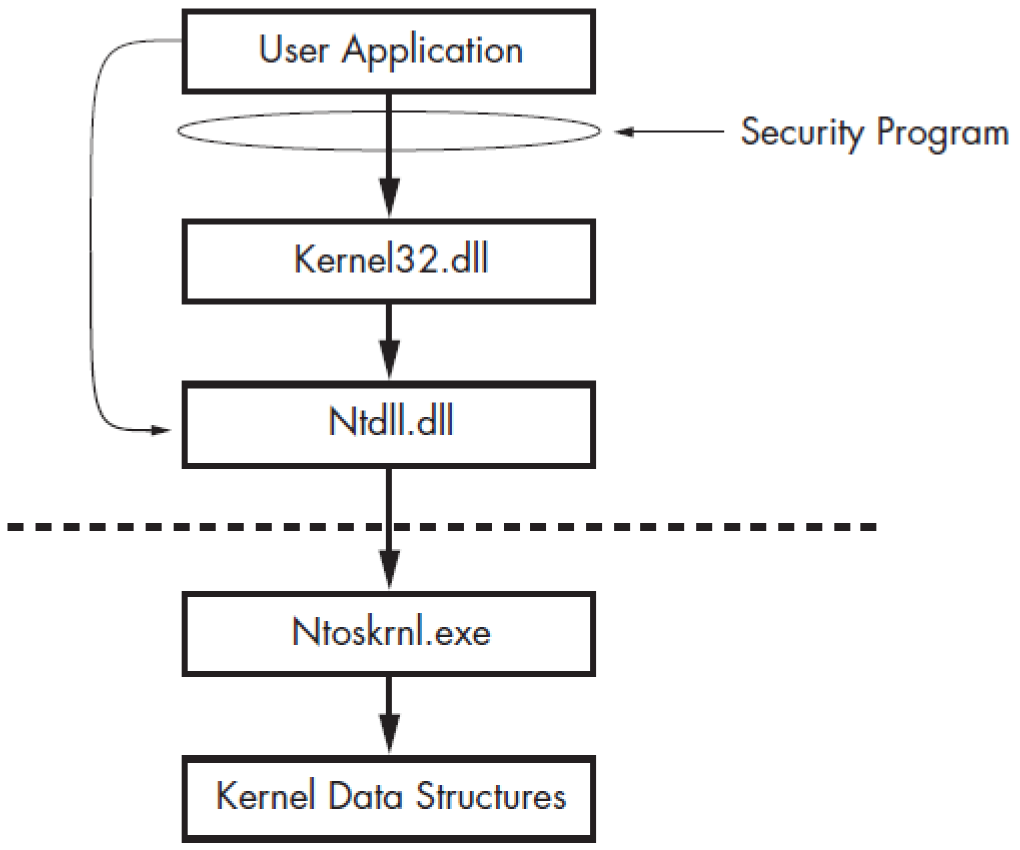
Análisis estático avanzado

- Llamadas al sistema



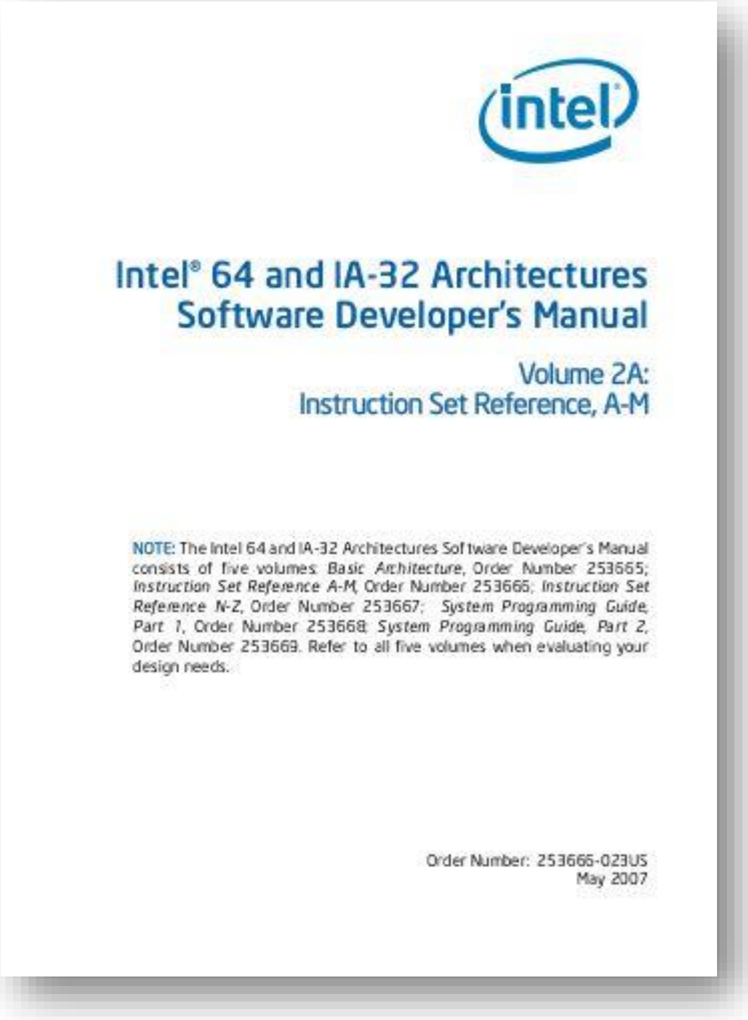
Análisis estático avanzado

- Llamadas al sistema



Análisis estático avanzado

- Más información

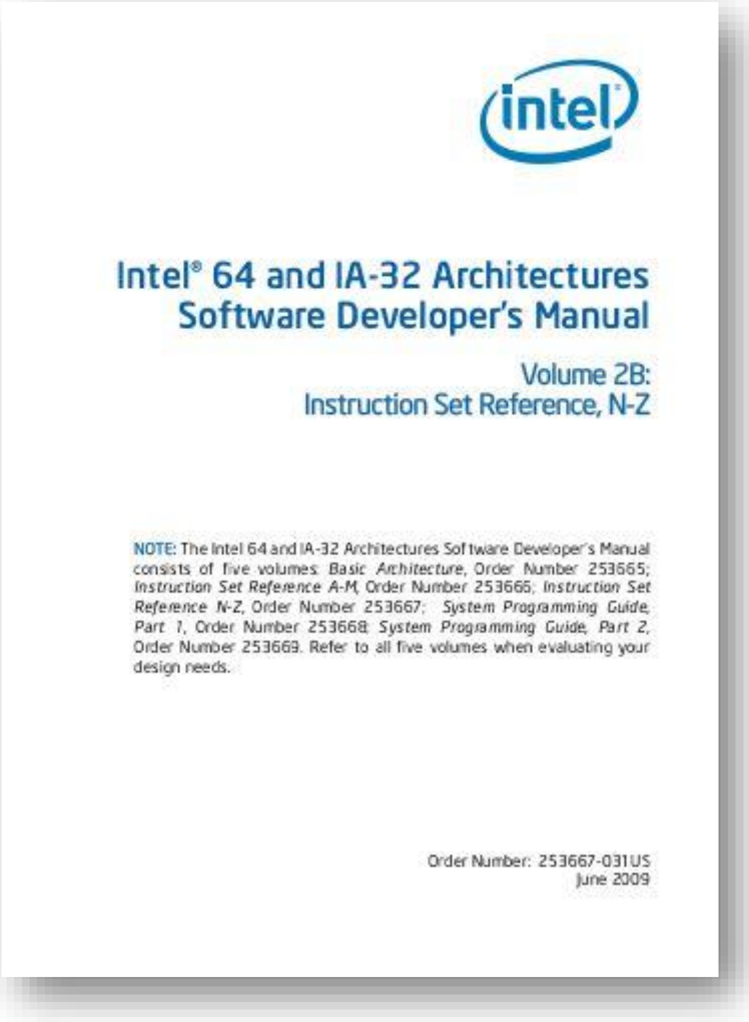


Intel® 64 and IA-32 Architectures
Software Developer's Manual

Volume 2A:
Instruction Set Reference, A-M

NOTE: The Intel 64 and IA-32 Architectures Software Developer's Manual consists of five volumes: *Basic Architecture*, Order Number 253665; *Instruction Set Reference A-M*, Order Number 253666; *Instruction Set Reference N-Z*, Order Number 253667; *System Programming Guide, Part 1*, Order Number 253668; *System Programming Guide, Part 2*, Order Number 253669. Refer to all five volumes when evaluating your design needs.

Order Number: 253666-023US
May 2007



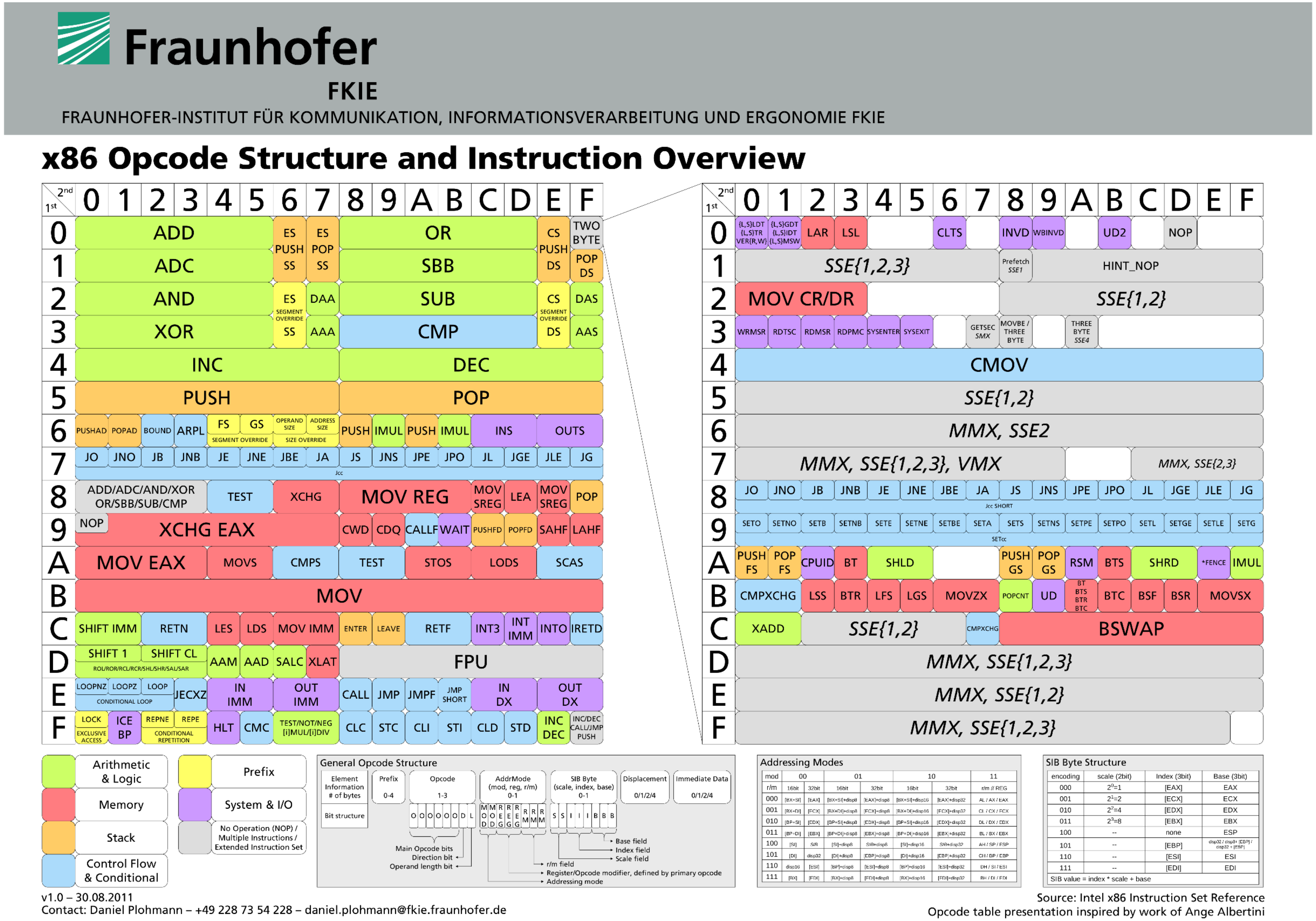
Intel® 64 and IA-32 Architectures
Software Developer's Manual

Volume 2B:
Instruction Set Reference, N-Z

NOTE: The Intel 64 and IA-32 Architectures Software Developer's Manual consists of five volumes: *Basic Architecture*, Order Number 253665; *Instruction Set Reference A-M*, Order Number 253666; *Instruction Set Reference N-Z*, Order Number 253667; *System Programming Guide, Part 1*, Order Number 253668; *System Programming Guide, Part 2*, Order Number 253669. Refer to all five volumes when evaluating your design needs.

Order Number: 253667-031US
June 2009

Análisis estático avanzado



- ¡Probemos! Práctica, 3.3

Análisis estático avanzado: IDA Pro

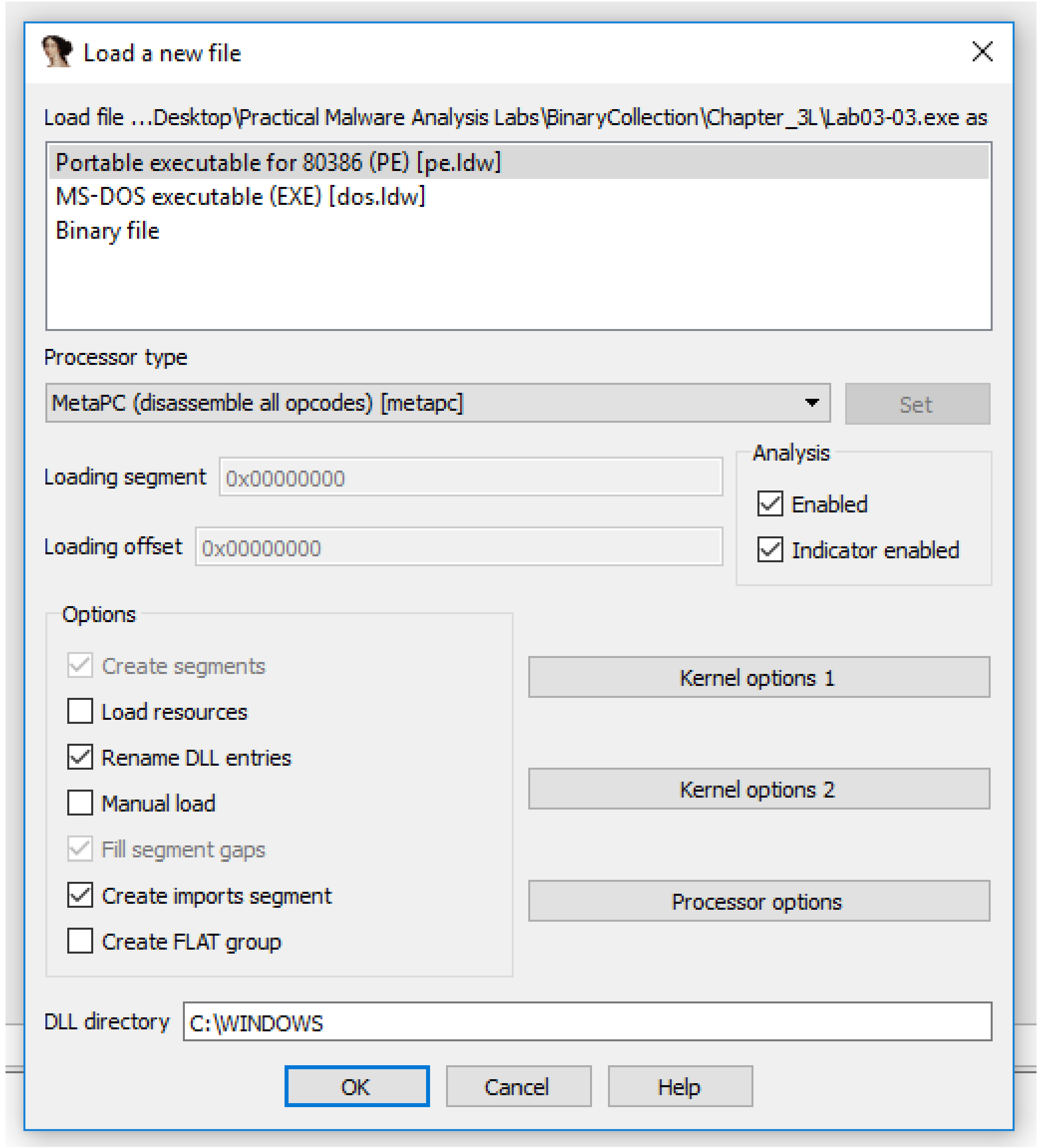
- IDA Pro

  - Estándar de facto
  - Hay otras herramientas pero no tan buenas
  - Es comercial (1500€ aprox)
    - Versión 7.0 free

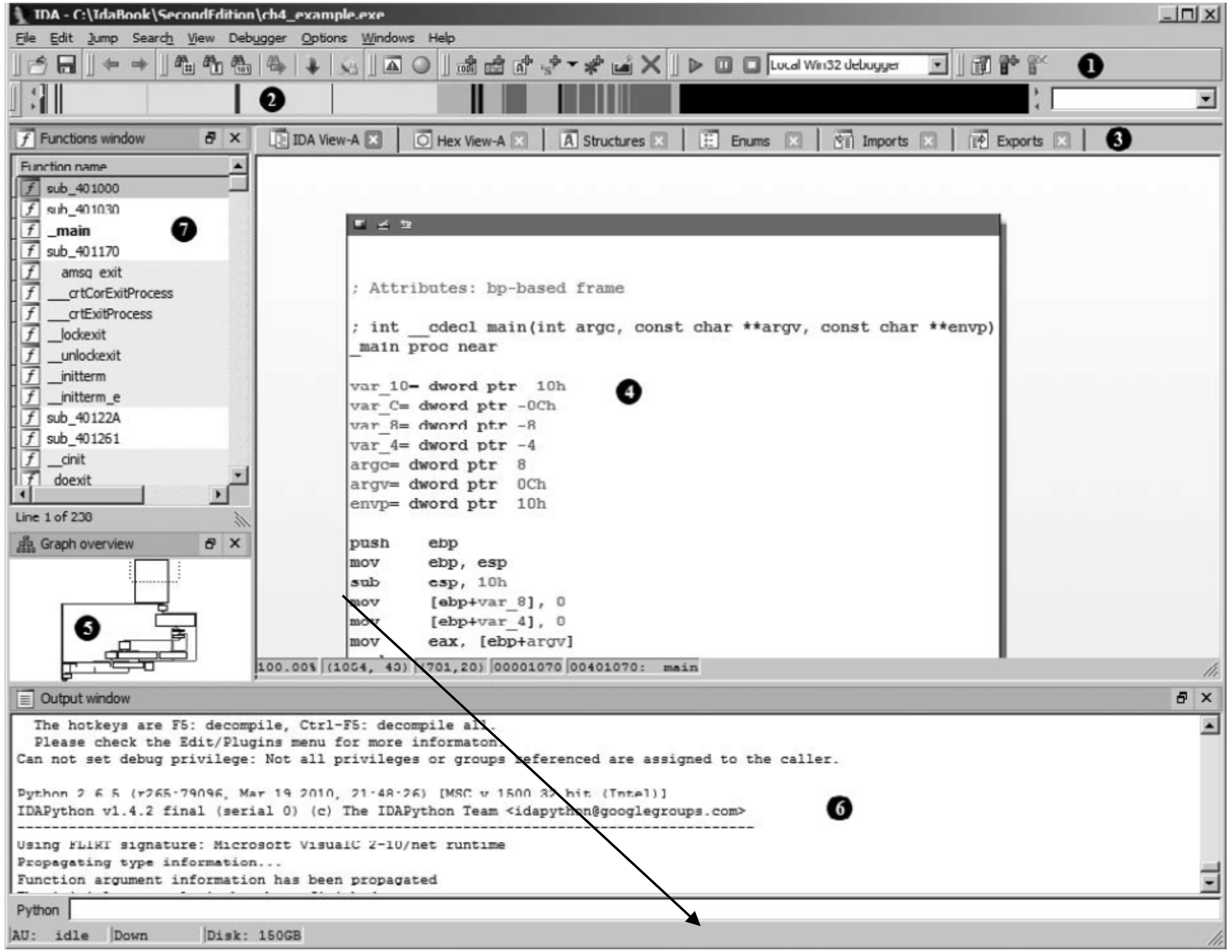
Análisis estático avanzado: IDA Pro

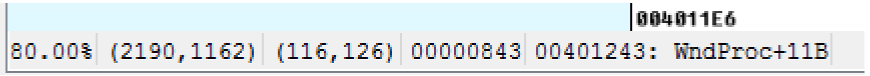
- IDA Pro
  - Descubre funciones
  - Analiza la pila
  - Identifica variables locales
  - Fast Library Identification and Recognition Technology (FLIRT)
  - Plugins
    - Hex-Rays decompiler 2500€
    - Zynamics BinDiff (free desde 2016)

Análisis estático avanzado: IDA Pro

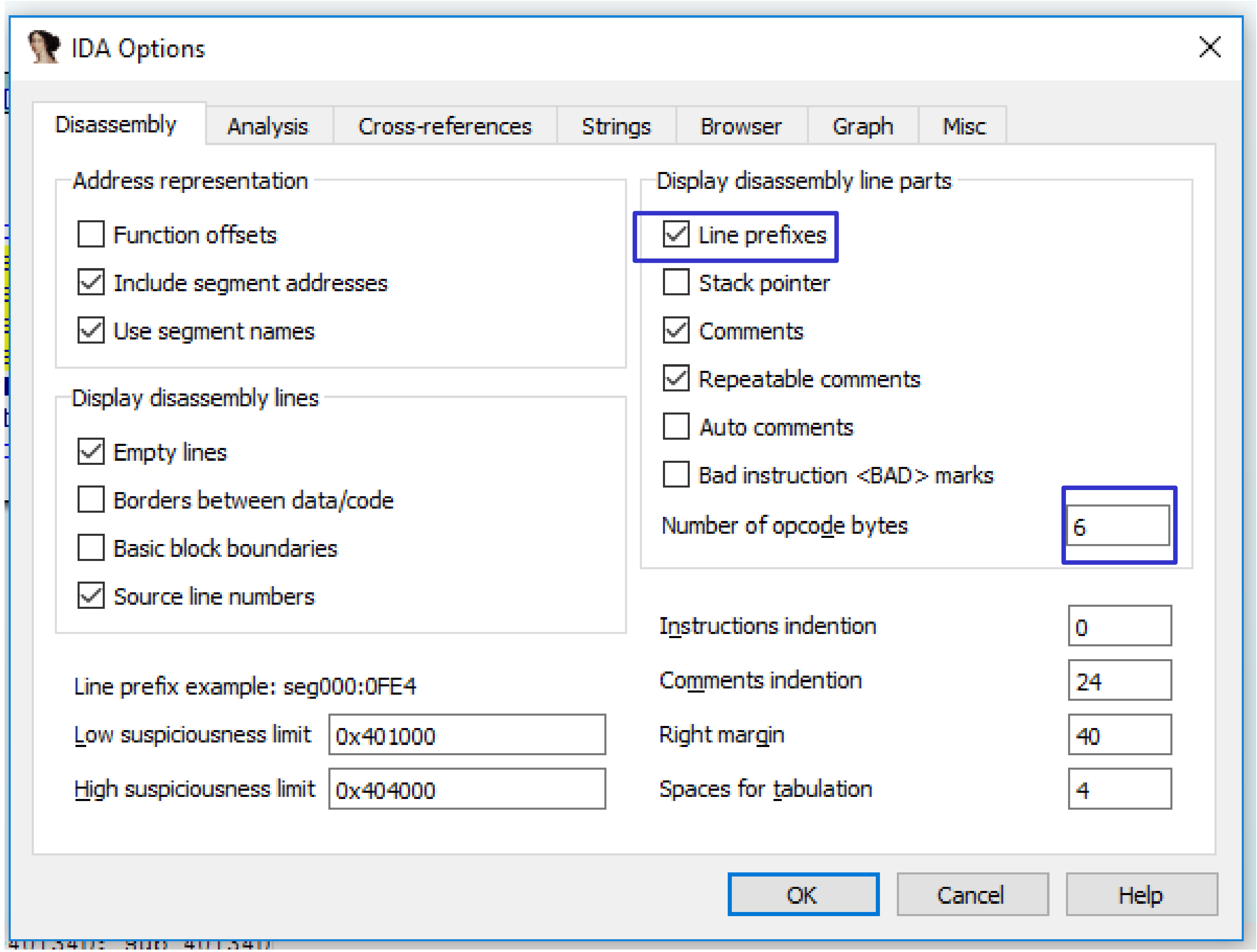


Análisis estático avanzado: IDA Pro





Análisis estático avanzado: IDA Pro



Análisis estático avanzado: IDA Pro

- Navegando por el código
  - Links
    - *Sub links* comienzo de funciones
    - *Loc links* destino de saltos.
    - *Offset links* posiciones de memoria
  - Cross references
    - CODE XREF
    - DATA XREF
  - Botones de History Adelante / Atrás

Análisis estático avanzado: IDA Pro

- Navegando por el código
  - Jump
    - to address
    - to file offset
  - Searching

Análisis estático avanzado: IDA Pro

- Análisis de funciones

  - IDA Pro es capaz de reconocer funciones, etiquetarlas y deducir
    - parámetros (prefijo arg_)
    - y variables locales (prefijo var_)

  - Permite un mayor nivel de abstracción
  - A veces puede omitir funciones
    - Podemos especificarlas a mano

Análisis estático avanzado: IDA Pro

- Gráficos útiles

  - Xrefs to: different paths that a program can take to reach a particular function.
  - Xrefs from: This view can quickly tell you what a function does and what the functions do underneath it. This is the easiest way to get a quick overview of the function.

Análisis estático avanzado: IDA Pro

- Mejorando el resultado

  - Podemos:
    - Renombrar funciones / locations
    - Poner comentarios
      - : → un comentario
      - ; → un comentario que se repite en cada xref
    - Cambiar el tipo de una referencia (dec/hex/bin…)

Análisis estático avanzado: IDA Pro

- Mejorando el resultado

  - A veces el analizador puede equivocarse…

```
mov eax, loc_410000
add ebx, eax
mul ebx
```

Análisis estático avanzado: IDA Pro

- Mejorando el resultado

  - Constantes
    - ¿Cómo tratamos un #DEFINE ?
    - IDA Pro trae un catálogo de los usuales en la librería estándar de C y en el SDK de Windows (Type libraries subview)

Análisis estático avanzado: IDA Pro

- Scripting
  - IDC
  - IDAPython

Análisis estático avanzado: IDA Pro

- ¡Probemos! Práctica, 3.4

¡Gracias por atenderme!


JOIN OUR TEAM
careers@jtsec.es
jtsec
BEYOND IT SECURITY

Contacto

jtsec: Beyond IT Security

Granada & Madrid

hola@jtsec.es

@jtsecES

www.jtsec.es

"Cualquier loco puede realizar algo complicado. Se necesita un genio para hacerlo simple."

Woody Guthrie