

Introducción a la Ingeniería Inversa

Diseño de Aplicaciones Seguras

Ricardo J. Rodríguez

rjrodriguez@unizar.es



Universidad
Zaragoza

19 de Enero de 2012

Grupo de Ingeniería de Sistemas de Eventos Discretos
Universidad de Zaragoza

Outline

- 1 Introducción a la Ingeniería Inversa
 - Qué es la Ingeniería Inversa
 - Motivación
 - Aproximaciones a la Ingeniería Inversa
- 2 Conocimientos previos
- 3 Refrescando la memoria...
 - Conocimientos de manejo de debuggers
 - Conocimientos de arquitectura x86 básica
 - Conocimientos de las estructuras PE
- 4 Herramientas útiles
- 5 Técnicas de análisis
 - Código muerto
 - Código 'vivo'
- 6 Técnicas de *reversing*
 - CD Check
 - *Patching* y *loaders*
 - *Time-trials* y Registro de Windows
 - Captura del *serial* y *Keygenning*
 - Archivos de licencia
 - Desempacado (*unpacking*)
- 7 Algunos métodos *antireversing*
- 8 Ejemplo práctico
 - Estudio del crackME
 - Algoritmo de generación
- 9 Conclusiones
- 10 Investigación en Ingeniería Inversa

1 Introducción a la Ingeniería Inversa

- Qué es la Ingeniería Inversa
- Motivación
- Aproximaciones a la Ingeniería Inversa

2 Conocimientos previos

3 Refrescando la memoria...

- Conocimientos de manejo de debuggers
- Conocimientos de arquitectura x86 básica
- Conocimientos de las estructuras PE

4 Herramientas útiles

5 Técnicas de análisis

- Código muerto
- Código 'vivo'

6 Técnicas de *reversing*

- CD Check
- *Patching* y loaders
- *Time-trials* y Registro de Windows
- Captura del *serial* y *Keygenning*
- Archivos de licencia
- Desempacado (*unpacking*)

7 Algunos métodos *antireversing*

8 Ejemplo práctico

- Estudio del crackME
- Algoritmo de generación

9 Conclusiones

10 Investigación en Ingeniería Inversa

Introducción a la Ingeniería Inversa (I)

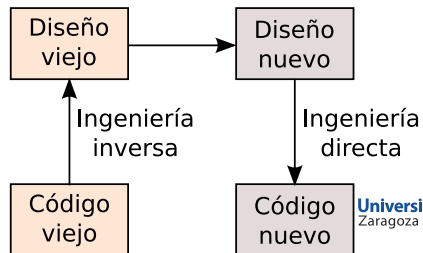
Ingeniería inversa (*reverse engineering*)

- **Descubrir cómo funciona** (algo) a partir de un análisis exhaustivo
- **Mejora de productos/sistemas viejos**
- **Diferentes dominios** de aplicación
 - *Hardware (legacy hardware)*
 - *Software (e.g. Samba)*

Introducción a la Ingeniería Inversa (I)

Ingeniería inversa (*reverse engineering*)

- Descubrir cómo funciona (algo) a partir de un análisis exhaustivo
- Mejora de productos/sistemas viejos
- Diferentes dominios de aplicación
 - Hardware (*legacy hardware*)
 - Software (e.g. Samba)
- Ir hacia atrás en el ciclo de desarrollo



Introducción a la Ingeniería Inversa (II)

Motivación

- Interoperabilidad
- Documentación inexistente
- Análisis de productos finales
- Auditoría de seguridad
- Espionaje industrial o militar (e.g. Segunda GM)
- Eliminación de anticopias o limitaciones de uso
- Creación de duplicados sin licencia
- Académicos
- Curiosidad innata
- Para aprender de los errores de otros

Introducción a la Ingeniería Inversa (II): Motivación (2)

Encontrar vulnerabilidades en el software

- Chequeo de cotas de manera incorrecta (*buffer overflow*)
- Uso de entradas sin validación
- Rutinas cíclicas para entrada de datos
- Operaciones de copia a nivel de byte
- Aritmética de punteros basada en entradas dadas del usuario
- “Confianza” en sistemas seguros con entradas dinámicas

Introducción a la Ingeniería Inversa (III)

Aproximaciones

● Caja blanca

- Analizar y entender el código fuente (o binario desensamblado)
- Encontrar errores de programación y/o implementación
- Analizador estático: búsqueda de patrones (¿falso positivo?)
- Ejemplos: *WhiteBox SecureAssistant*, *IDAPro*, *SourceScope*...

● Caja negra

- Analizar programa según diferentes entradas → software en ejecución
- No es tan efectivo, pero requiere menos experiencia
- Método habitual para detectar exploits

● Caja gris

- Combinación de las dos anteriores
- Ejecución de programa mediante debug...
- ...alimentándolo con diferentes entradas
- Ejemplos: Rational's Purify (análisis uso/consumo memoria), Valgrind

Introducción a la Ingeniería Inversa (III)

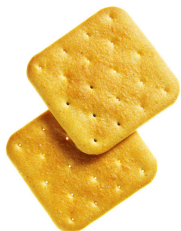
Reverse engineering code

- También conocida como *cracking*
- **Eliminar protecciones de código** (*copyrights*)
- NO siempre es malo: detección de *bugs*, potenciales *exploits*, ...
- Lucha **contra el malware**

Introducción a la Ingeniería Inversa (III)

Reverse engineering code

- También conocida como *cracking*
- Eliminar protecciones de código (*copyrights*)
- NO siempre es malo: detección de *bugs*, potenciales *exploits*, ...
- Lucha *contra el malware*
- **Crackers**: algo más que unas galletas...
 - **NO CONFUNDIR** con los *criminal hackers*



- 1 Introducción a la Ingeniería Inversa
 - Qué es la Ingeniería Inversa
 - Motivación
 - Aproximaciones a la Ingeniería Inversa
- 2 **Conocimientos previos**
- 3 Refrescando la memoria...
 - Conocimientos de manejo de debuggers
 - Conocimientos de arquitectura x86 básica
 - Conocimientos de las estructuras PE
- 4 Herramientas útiles
- 5 Técnicas de análisis
 - Código muerto
 - Código 'vivo'
- 6 Técnicas de *reversing*
 - CD Check
 - *Patching* y *loaders*
 - *Time-trials* y Registro de Windows
 - Captura del *serial* y *Keygenning*
 - Archivos de licencia
 - Desempacado (*unpacking*)
- 7 Algunos métodos *antireversing*
- 8 Ejemplo práctico
 - Estudio del crackME
 - Algoritmo de generación
- 9 Conclusiones
- 10 Investigación en Ingeniería Inversa

Conocimientos previos (I)

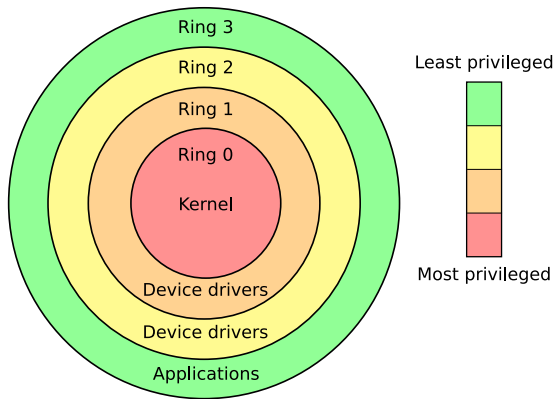
- **Ensamblador** (<http://www.intel.com/products/processor/manuals/>)

Conocimientos previos (I)

- **Ensamblador** (<http://www.intel.com/products/processor/manuals/>)
- **Funcionamiento interno SS.OO.**
 - *¿Qué ocurre al pulsar un botón?*
 - *¿Y al aceptar un checkbox?*
 - La biblia de APIs de Windows: WinXXAPI (32 o 64 bits)
 - <http://msdn.microsoft.com/en-us/library/Aa383723>
 - API: Application Programming Interface
- **Estructura interna de un PE (*Portable Executable*)**
 - <http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.msp>
- **Debuggear programas**
 - *Breakpoints*
 - *Step into, step over, ...*

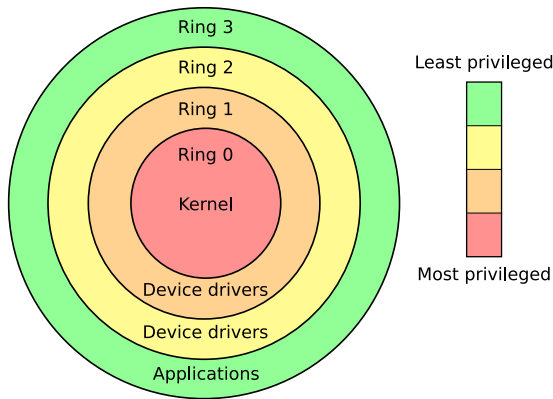
Conocimientos previos (II)

- Anillos de privilegio $R0 \dots R3$
- Más privilegios (*kernel*) a menos privilegios (aplicaciones)



Conocimientos previos (II)

- Anillos de privilegio $R0 \dots R3$
- Más privilegios (*kernel*) a menos privilegios (aplicaciones)



“When a program runs, then it can be cracked”

- 1 Introducción a la Ingeniería Inversa
 - Qué es la Ingeniería Inversa
 - Motivación
 - Aproximaciones a la Ingeniería Inversa
- 2 Conocimientos previos
- 3 Refrescando la memoria. . .
 - Conocimientos de manejo de debuggers
 - Conocimientos de arquitectura x86 básica
 - Conocimientos de las estructuras PE
- 4 Herramientas útiles
- 5 Técnicas de análisis
 - Código muerto
 - Código 'vivo'
- 6 Técnicas de *reversing*
 - CD Check
 - *Patching* y loaders
 - *Time-trials* y Registro de Windows
 - Captura del *serial* y *Keygenning*
 - Archivos de licencia
 - Desempacado (*unpacking*)
- 7 Algunos métodos *antireversing*
- 8 Ejemplo práctico
 - Estudio del crackME
 - Algoritmo de generación
- 9 Conclusiones
- 10 Investigación en Ingeniería Inversa

Conocimientos de manejo de debuggers

Nociones de debugger

- Código ensamblador
- Ejecución paso a paso
- Útil para detectar fallos en programas
- Comandos típicos:
 - *Breakpoint*: punto de ruptura
 - *Step into / Step over*
 - *Animate into / Animate over*
 - Ejecución hasta RET

Conocimientos de arquitectura x86 básicos (I)

Registros

- Diferentes **tipos de registros, según uso**
 - Propósito general
 - Segmento
 - Apuntador de instrucciones
 - Índice
 - Bandera (*flags*)

Conocimientos de arquitectura x86 básicos (II)

Registros de propósito general

- *EAX*: registro acumulador
 - Operaciones aritméticas y lógicas
 - Recoge resultado después de operaciones (habitualmente)
- *EBX*: registro base
 - Transferencias de datos entre memoria y procesador
- *ECX*: registro contador
 - Contador en bucles (**loop**), cadenas (**rep**), desplazamientos (**shX**)
- *EDX*: registro datos
 - Multiplicación y división (junto con *EAX*)
 - Entrada y salida de puertos

Conocimientos de arquitectura x86 básicos (III) (1)

Registros de **bandera** (*flags*)

- Cada bit significa una cosa diferente

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Flag	-	-	-	-	<i>OF</i>	<i>DF</i>	<i>IF</i>	<i>TF</i>	<i>SF</i>	<i>ZF</i>	-	<i>AF</i>	-	<i>PF</i>	-	<i>CF</i>

- *CF*: acarreo (arrastre tras operaciones aritméticas)
- *OF*: desbordamiento
- *ZF*: cero (se pone a 1 si el resultado de la última operación es 0)
- *SF*: signo (se pone a 1 si el resultado de la última operación es negativo)
- *PF*: paridad (se pone a 1 si el resultado de la última operación es par)

Conocimientos de arquitectura x86 básicos (III) (2)

Registros de **bandera** (*flags*)

- Cada bit significa una cosa diferente

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Flag	-	-	-	-	OF	DF	IF	TF	SF	ZF	-	AF	-	PF	-	CF

- *AF*: auxiliar (operaciones BCD)
- *DF*: dirección (ops. de cadena, dirección ascendente o descendente en recorrido)
- *IF*: interrupciones (0 → “enmascaramiento” de ints)
- *TF*: *trap flag* (ejecución paso a paso, ya veremos...)

Conocimientos de arquitectura x86 básicos (IV) (1)

Tipos de instrucciones

- **Codificación variable y alineación independiente**
- INSTRUCCIÓN DESTINO, ORIGEN
- Puede aparecer **un registro, dos o ninguno** en una instrucción (registros implícitos)
- **Referencias a memoria** en cualquiera de los lados (NO a la vez!)
- **Cada operación puede modificar registros** → consultar manual ASM

Conocimientos de arquitectura x86 básicos (IV) (2)

Tipos de instrucciones

- **Flags** recogen resultado de operaciones ALU
- Direccionamiento: inmediato, offset o con índice escalado (no relativo al PC)
- ... excepto saltos (instrucción **jmp**)!
- Instrucciones atómicas (**xchg**, **cmpxchg/cmpxchg8b**, **xadd**, prefijo **LOCK**)
- **Registros para coma flotante** (ops. especiales)
- **Instrucciones SIMD** (*Single Instruction, Multiple Data*): paralelismo a nivel de datos

Conocimientos de arquitectura x86 básicos (V) (3)

Flujo del programa

- **jmp**
- **call** → **ret**
- Pueden ser cercanos o lejanos, según destino

La pila

- LIFO (*Last In First Out*)
- Paso de parámetros a procedimientos
- **push** / **pop**
- **call**
- **ret**

```
push ebp
mov  ebp, esp
sub  esp, SIZE_LOCAL_VARIABLES
```

Código de la subrutina

```
mov  esp, ebp
pop  ebp
ret  SIZE_INPUT_PARAMS; 4 by default
```



Conocimientos de las estructuras PE (I)

Encabezado DOS
Stub DOS
Encabezado NT
Encabezado secc.
Stub NT

- **Encabezados:** tamaño constante
- EXEs, DLLs, OBJs
- **Cabecera MZ** (DOS): código a ejecutar si no es compatible con MS-DOS
 - Tamaño 0x40, últimos 4 @cabecera PE
- **Cabecera PE** (Portable Executable)
 - 04h: tipo de máquina compilado
 - 06h: número de secciones
 - 14h: tamaño de la cabecera opcional
 - 16h: características del fichero
 - 18h: comienzo cabecera opcional
 - 01Ch: tamaño del código
 - 028h: Entry Point
 - 034h: dirección base del fichero

Conocimientos de las estructuras PE (II)

Encabezado DOS
Stub DOS
Encabezado NT
Encabezado secc.
Stub NT

- Recuerda: **secciones alineadas en memoria durante ejecución**
- **Tablas de secciones:**
 - @PE header+tamaño de PE header+tamaño de cabecera opcional
 - 00h: nombre de la sección
 - 08h: tamaño virtual
 - 0ch: dirección virtual
 - 024h: flags (lectura, escritura, ejecución...)
- **Secciones:** división del código
 - .text, .idata, .bss, .data, .reloc
 - Nombre de la sección irrelevante para el funcionamiento

Conocimientos de las estructuras PE (II)

Referencias

- Wikipedia
(http://en.wikipedia.org/wiki/Portable_Executable)
- Peering Inside the PE: A Tour of the Win32 Portable Executable File Format
(<http://msdn.microsoft.com/en-us/library/ms809762.aspx>)
- Microsoft PE and COFF Specification
(<http://msdn.microsoft.com/en-us/windows/hardware/gg463119>)
- The .NET File Format
(<http://ntcore.com/files/dotnetformat.htm>)

- 1 Introducción a la Ingeniería Inversa
 - Qué es la Ingeniería Inversa
 - Motivación
 - Aproximaciones a la Ingeniería Inversa
- 2 Conocimientos previos
- 3 Refrescando la memoria...
 - Conocimientos de manejo de debuggers
 - Conocimientos de arquitectura x86 básica
 - Conocimientos de las estructuras PE
- 4 **Herramientas útiles**
- 5 Técnicas de análisis
 - Código muerto
 - Código 'vivo'
- 6 Técnicas de *reversing*
 - CD Check
 - *Patching* y *loaders*
 - *Time-trials* y Registro de Windows
 - Captura del *serial* y *Keygenning*
 - Archivos de licencia
 - Desempacado (*unpacking*)
- 7 Algunos métodos *antireversing*
- 8 Ejemplo práctico
 - Estudio del crackME
 - Algoritmo de generación
- 9 Conclusiones
- 10 Investigación en Ingeniería Inversa

Herramientas útiles (I)

Básicas

- **Desensambladores**
 - General: W32Dasm, IDA Pro, ...
 - Específico: p32Dasm (VBasic), Reflector (.NET), ...
- **Editor hexadecimal** (e.g., HexWorkshop)
- **Debuggers**
 - Soft ICE
 - OllyDBG
 - IDA Pro
 - ...

Herramientas útiles (II)

Otras...

- **Identificadores y editores PE** (PEiD, PEEditor, ...)
- **Visores/editores de recursos** (XNResource Editor, Resorce Hacker, ...)
- **Volcadores de memoria** (LordPE Deluxe, ProcDump, SirPE, ...)

Herramientas útiles (II)

Otras...

- **Identificadores y editores PE** (PEiD, PEEditor, ...)
- **Visores/editores de recursos** (XNResource Editor, Resorce Hacker, ...)
- **Volcadores de memoria** (LordPE Deluxe, ProcDump, SirPE, ...)
- **Emuladores** (HASP, Sentinel, ...)
- **Monitores de API** (KaKeeware Application Monito, Event2Address, ...)
- **Reparadores de IAT** (ImportREC, ReVirgin)

Herramientas útiles (III)

Documentación: manuales y tutoriales

- La más importante → **hay que leer para aprender**
- **Internet**, una herramienta útil y al alcance de cualquiera
 - **Grupos de cracking**
 - Hispano-hablantes (WkT, CLS, eCh, ...)
 - Extranjeros (RZR, TNT!, ARTeam, RE, ...)
 - **Foros**
 - elHacker (sección 'Programación→Ingeniería Inversa')
 - ExeTools
 - WoodMan
 - **Páginas personales** (Karpoff, Shoulck, Saccopharynx, +NCR, AbsSha, ...)
 - Tuts4You (<http://www.tuts4you.com/>)

Herramientas útiles (III)

Documentación: manuales y tutoriales

- La más importante → **hay que leer para aprender**
- **Internet**, una herramienta útil y al alcance de cualquiera
 - **Grupos de cracking**
 - Hispano-hablantes (WkT, CLS, eCh, ...)
 - Extranjeros (RZR, TNT!, ARTeam, RE, ...)
 - **Foros**
 - elHacker (sección 'Programación→Ingeniería Inversa')
 - ExeTools
 - WoodMan
 - **Páginas personales** (Karpoff, Shoulck, Saccopharynx, +NCR, AbsSha, ...)
 - Tuts4You (<http://www.tuts4you.com/>)
- **Práctica, práctica y (un poco más de) práctica**
 - Cualquier (pobre) programa que caiga en nuestras manos
 - Crackmes (<http://www.crackmes.us/>)

- 1 Introducción a la Ingeniería Inversa
 - Qué es la Ingeniería Inversa
 - Motivación
 - Aproximaciones a la Ingeniería Inversa
- 2 Conocimientos previos
- 3 Refrescando la memoria...
 - Conocimientos de manejo de debuggers
 - Conocimientos de arquitectura x86 básica
 - Conocimientos de las estructuras PE
- 4 Herramientas útiles
- 5 **Técnicas de análisis**
 - Código muerto
 - Código 'vivo'
- 6 Técnicas de *reversing*
 - CD Check
 - *Patching* y *loaders*
 - *Time-trials* y Registro de Windows
 - Captura del *serial* y *Keygenning*
 - Archivos de licencia
 - Desempacado (*unpacking*)
- 7 Algunos métodos *antireversing*
- 8 Ejemplo práctico
 - Estudio del crackME
 - Algoritmo de generación
- 9 Conclusiones
- 10 Investigación en Ingeniería Inversa

Análisis de código muerto: descripción

- Programas **sin protección (o protección mínima)**
- Es **raro** que funcione
- Herramientas necesarias
 - Identificador PE
 - Desensamblador
 - Editor Hexadecimal
 - Cerebro
 - Intuición
 - Lápiz y papel
 - Suerte :)
- Casos típicos
 - Salto JE/JNE (JZ/JNZ) para registro correcto
 - Número de registro embebido en la aplicación
- **Muy sencillo (queremos desafíos!)**

Análisis de código muerto: ejemplos (I)

NOPeo de salto de comprobación

- Una o varias rutinas de comprobación de *serial*
- **NOPeo**: sustituir código máquina por NOP (No OPeration)
 - JE/JNE (74/75) → NOP (90)
 - JE/JNE (74/75) → JMP (EB)^a
 - Variantes: JX/JNX (cualquiera) → NOP (90) ó JMP (EB)

^aSi el salto es largo (destino a más de 32 bits desde el lugar de origen), varía...

```

:004984AF 68488A5300      push 00538A48
:004984B4 E8CCFBFFFF      call 00498085
:004984E9 85C0            test eax, eax
:004984EB 0F8499000000    js 0049855A
:004984C1 BE887C5200      mov esi, 00527C88
:004984C6 BF488A5300      mov edi, 00538A48
:004984C8 33C0            xor eax, eax
:004984CD 83C9FF          or ecx, 0FFFFFFF
:004984D0 F2              repnz

```

Pasos

- 1 Identificar PE y desensamblar
- 2 Buscar mensajes de chico malo
- 3 Analizar camino hasta el mensaje
- 4 NOPear salto/desviar camino

Análisis de código muerto: ejemplos (II)

A la caza del *serial*

- Una o varias rutinas de comprobación de serial
- El **código de registro** (*serial*) es **único** y...
- ...está **embebido** en la aplicación ¡!

```

00424910: PUSH registro.00422790          UNINJUB "english.dll"
00425000: MOV DWORD PTR SS:[EBP-00],registro.0042 UNINJUB "trial version expired"
00425008: MOV DWORD PTR SS:[EBP-00],registro.0042 UNINJUB "This trial version has exp
00425016: MOV DWORD PTR SS:[EBP-00],registro.0042 UNINJUB "Este producto ha caducado!"
0042511E: MOV DWORD PTR SS:[EBP-00],registro.0042 UNINJUB "Este producto ha caducado.
00425187: PUSH registro.00422794          UNINJUB "alt"
00425190: MOV DWORD PTR SS:[EBP-00],registro.0042 UNINJUB "error!"
00425420: PUSH registro.00422E48          UNINJUB "utilities 77 backdoor"
00425454: PUSH registro.00422E14          UNINJUB "FILE=FILE=FILE=9999"
00425508: PUSH registro.00422790          UNINJUB "english.dll"
00425525: MOV DWORD PTR SS:[EBP-00],registro.0042 UNINJUB "Registration"
0042552C: PUSH registro.00422E78          UNINJUB "there is a problem when tr
00425531: PUSH registro.00422E72          UNINJUB "ab"

```

Pasos

- 1 Identificar PE (¿está protegido?)
- 2 Desensamblar
- 3 Buscar mensajes de chico malo
- 4 Husmear la zona
- 5 Comprobar cadenas sospechosas :)

Análisis de código 'vivo': descripción

- Programas **con (o sin) protección**
- Herramientas necesarias
 - Identificador PE
 - Desensamblador
 - Debugger
 - Cualquiera del cinturón (dependerá de la aplicación a crackear)
 - Cerebro
 - Intuición
 - Lápiz y papel
 - Suerte :)
- **Más complicados** (i.e., divertido)
- Cada aplicación es un **reto nuevo y diferente**
- Casos típicos
 - Mmm... ¿cualquiera?

(luego veremos un ejemplo...)

Introducción a la Ingeniería Inversa

Diseño de Aplicaciones Seguras

Ricardo J. Rodríguez

rjrodriguez@unizar.es



Universidad
Zaragoza

19 de Enero de 2012

Grupo de Ingeniería de Sistemas de Eventos Discretos
Universidad de Zaragoza

- 1 Introducción a la Ingeniería Inversa
 - Qué es la Ingeniería Inversa
 - Motivación
 - Aproximaciones a la Ingeniería Inversa
- 2 Conocimientos previos
- 3 Refrescando la memoria...
 - Conocimientos de manejo de debuggers
 - Conocimientos de arquitectura x86 básica
 - Conocimientos de las estructuras PE
- 4 Herramientas útiles
- 5 Técnicas de análisis
 - Código muerto
 - Código 'vivo'
- 6 Técnicas de *reversing*
 - CD Check
 - *Patching* y loaders
 - *Time-trials* y Registro de Windows
 - Captura del *serial* y *Keygenning*
 - Archivos de licencia
 - Desempacado (*unpacking*)
- 7 Algunos métodos *antireversing*
- 8 Ejemplo práctico
 - Estudio del crackME
 - Algoritmo de generación
- 9 Conclusiones
- 10 Investigación en Ingeniería Inversa

Técnicas de *cracking* (I): CD Check

- Verificación del CD presente en la unidad
- Fichero concreto en el CD de la unidad (algunas veces)
- Protecciones más avanzadas: SafeDisc, StarForce
- Uso de unidades virtuales: DaemonTools

```

0041F160 898 06  JMP  EBP
0041F160 <74 00  JNZ  EAX
0041F160 5E      POP  ESI
0041F160 5C80   XOR  ESI, ESI
0041F160 5B      POP  EBX
0041F160 81C4 0C020000  ADD  ESP, 20C
0041F160 C2 0400  RETN
0041F160 > 8D424 148100  LER  ECX, DMORD PTR SS:[ESP+114]
0041F160 6F 00010000  PUSH 100
0041F160 8D4C24 10  LER  ECX, DMORD PTR SS:[ESP+10]
0041F160 50      PUSH EAX
0041F160 8D424 18  LER  ECX, DMORD PTR SS:[ESP+18]
0041F160 51      PUSH ECX
0041F160 8D424 14  LER  ECX, DMORD PTR SS:[ESP+14]
0041F160 52      PUSH EDI
0041F160 50      PUSH EAX
0041F160 8D4C24 20  LER  ECX, DMORD PTR SS:[ESP+20]
0041F160 68 00010000  PUSH 100
0041F160 51      PUSH ECX
0041F160 5E      PUSH ESI
0041F160 FF15 94C16100  JNZ  CD_CHECK_DRIVE_PRESENTATION
0041F160 898 06  JMP  EBP
0041F160 <75 00  JNZ  EAX
0041F160 5E      POP  ESI
0041F160 5B      POP  EBX
0041F160 81C4 0C020000  ADD  ESP, 20C
0041F160 C2 0400  RETN
0041F160 > 8B53 24  MOV  EDI, DMORD PTR DS:[EBX+24]
0041F160 8D424 14  LER  ECX, DMORD PTR SS:[ESP+14]
0041F160 81C2 F0320000  ADD  EDI, 3FD
0041F160 50      PUSH EAX
0041F160 6C      PUSH EDI
0041F160 50      PUSH EAX
0041F160 EB 74021E00  JNE  CD_CHECK_DRIVE_PRESENTATION
0041F160 83C4 00  ADD  ESP, 0
0041F160 F700   NEG  EBX
0041F160 5B80   SBB  EAX, EBX
0041F160 5E      POP  ESI
0041F160 40      INC  ECX
0041F160 5B      POP  EBX
0041F160 81C4 0C020000  ADD  ESP, 20C
0041F160 C2 0400  RETN
0041F160 898 06  JMP  EBP
  
```

APIs típicas

- GetDriveTypeA
 - EAX = 5 si hay CD
- GetVolumeInformationA

Técnicas de *cracking* (II): *Patching* y loaders

Patching

- Objetivo: **cambiar flujo natural de ejecución del programa**
 - Cambio de instrucciones máquina
 - Modificando (tras un CMP o TEST) o insertando saltos
 - Sustituyendo por NOPs
- Métodos habituales: búsqueda de cadenas o chequeo de APIs
- **Cambios estáticos** (i.e., permanentes)

Loaders

- Como el *patching*, pero **“en caliente”** → **más elegante**
- Dos tipos (básicos)
 - Simples
 - *Debuggers* (más complejos): útil para programas empacados
- **Cambios dinámicos** (i.e., temporales)

Técnicas de *cracking* (III): *Time-trials* y Registro

Time-trials

- **Protección por tiempo** (uso limitado X días/minutos)
- APIs típicas de chequeo
 - GetLocalTime
 - GetFileTime
 - GetSystemTime

Registro de Windows

- Guardan **datos en el Registro de Windows**
- APIs típicas
 - RegCloseKey
 - RegCreateKeyEx
 - RegOpenKeyEx
 - RegSetValueEx
 - RegQueryValueEx

Técnicas de *cracking* (V): Captura del *serial* y *Keygenning*

Captura del *serial*

- Objetivo: conseguir número de registro del programa
- Idéntico para todos los usuarios
- Embebido en la aplicación
- Fácil: búsqueda de cadenas con patrones conocidos. . .

Keygenning

- Objetivo: encontrar algoritmo de generación de claves
- Complejidad del algoritmo variable
- Cada usuario tiene un número de registro diferente
- Ingeniería inversa pura y dura

Técnicas de *cracking* (VI): Archivos de licencia

- Se registran mediante **archivos de licencia**
- **Cheques rutinarios contra servidor** de la empresa (a veces)
- APIs típicas
 - Conexión: `connect`, `WSAConnect`
 - Recepción: `recv`, `recvfrom`, `WSARecv`, `WSARecvFrom`, `WSARecvMsg`
 - Envío: `send`, `sendto`, `WSASend`, `WSASendTo`, `WSASendMsg`
- **Algunos usan criptografía** (i.e., licencia codificada)
 - MUY complicados de conseguir licencia correcta
→ Dependerá del algoritmo criptográfico usado
- Solución: **intentar parchear**...

Técnicas de *cracking* (VII): Desempacado (*unpacking*)

- Programas **protegidos**
- Pueden ser muy complicados (*anti-dumps, scrambling, ...*)
- Pasos a realizar
 - 1 Hallar el OEP (*Original Entry Point*)
 - *Stolen bytes*
 - Cambios en la cabecera PE
 - 2 **Dumpear el proceso de memoria** (estará desempacado!)
 - Secciones virtuales
 - Ofuscación de código
 - 3 **Arreglar la IAT** (*Import Address Table*)
 - Emulación de APIs
 - Redireccionamiento de APIs
- Lista:
http://en.wikipedia.org/wiki/Executable_compression
- Existen ***unpackers* automáticos**: *tools* propias o *scripts*

- 1 Introducción a la Ingeniería Inversa
 - Qué es la Ingeniería Inversa
 - Motivación
 - Aproximaciones a la Ingeniería Inversa
- 2 Conocimientos previos
- 3 Refrescando la memoria...
 - Conocimientos de manejo de debuggers
 - Conocimientos de arquitectura x86 bás
 - Conocimientos de las estructuras PE
- 4 Herramientas útiles
- 5 Técnicas de análisis
 - Código muerto
 - Código 'vivo'
- 6 Técnicas de *reversing*
 - CD Check
 - *Patching* y *loaders*
 - *Time-trials* y Registro de Windows
 - Captura del *serial* y *Keygenning*
 - Archivos de licencia
 - Desempacado (*unpacking*)
- 7 Algunos métodos *antireversing*
- 8 Ejemplo práctico
 - Estudio del crackME
 - Algoritmo de generación
- 9 Conclusiones
- 10 Investigación en Ingeniería Inversa

Algunos métodos *antireversing*

- Técnicas **anti-debugging**
 - APIs
 - Windows *internals tricks*
 - Detección de herramientas
 - Lectura recomendada: <http://pferrie.tripod.com/>
- Técnicas **anti-tracing**
- Técnicas **anti-dumping**
- Técnicas de **ocultación de OEP**
- Otras técnicas:
 - **Ofuscamiento de código** (código basura)
 - **Detección de modificaciones** (CRC, APIs)
 - **Criptografía**

- 1 Introducción a la Ingeniería Inversa
 - Qué es la Ingeniería Inversa
 - Motivación
 - Aproximaciones a la Ingeniería Inversa
- 2 Conocimientos previos
- 3 Refrescando la memoria...
 - Conocimientos de manejo de debuggers
 - Conocimientos de arquitectura x86 básica
 - Conocimientos de las estructuras PE
- 4 Herramientas útiles
- 5 Técnicas de análisis
 - Código muerto
 - Código 'vivo'
- 6 Técnicas de *reversing*
 - CD Check
 - *Patching* y *loaders*
 - *Time-trials* y Registro de Windows
 - Captura del *serial* y *Keygenning*
 - Archivos de licencia
 - Desempacado (*unpacking*)
- 7 Algunos métodos *antireversing*
- 8 Ejemplo práctico
 - Estudio del crackME
 - Algoritmo de generación
- 9 Conclusiones
- 10 Investigación en Ingeniería Inversa

Ejemplo práctico (I): estudio del crackME

- CrackMe sencillito de <http://www.crackmes.de>
- Objetivo: **hacer un generador de claves**



Pasos

- 1 Analizar para ver si está limpio
- 2 Insertar nombre y código para ver chico malo
- 3 Cargar en *debugger* (e.g., OllyDBG)

Ejemplo práctico (I): estudio del crackME

- CrackMe sencillito de <http://www.crackmes.de>
- Objetivo: **hacer un generador de claves**



Pasos

- 1 Analizar para ver si está limpio
- 2 Insertar nombre y código para ver chico malo
- 3 Cargar en *debugger* (e.g., OllyDBG)

Ejemplo práctico (II): estudio del crackME (1)

00401000	6A 00	PUSH 0	pModule = NULL
00401002	E8 89020000	CALL <JMP.&kernel32.GetModuleHandleA>	GetModuleHandleA
00401007	R3 94314000	MOV DWORD PTR DS:[403194],EAX	
0040100C	6A 00	PUSH 0	IParam = NULL
0040100E	68 29104000	PUSH Ice9.00401029	DlgProc = Ice9.00401029
00401013	6A 00	PUSH 0	hOwner = NULL
00401015	6A 65	PUSH 65	pTemplate = 65
00401017	FF35 94314000	PUSH DWORD PTR DS:[403194]	hInst = NULL
0040101D	E8 3E020000	CALL <JMP.&user32.DialogBoxParamA>	DialogBoxParamA
00401024	E8 61020000	CALL <JMP.&kernel32.ExitProcess>	ExitCode = 0
00401029	55	PUSH EBP	ExitProcess
0040102A	8BEC	MOV EBP,ESP	
0040102C	8B45 0C	MOV EAX,[ARG.2]	
0040102F	83F9 10	CMF EAX,10	
00401032	75 0F	JNZ SHORT Ice9.00401043	
00401034	6A 00	PUSH 0	Result = 0
00401036	FF75 08	PUSH [ARG.1]	hWnd
00401039	E8 28020000	CALL <JMP.&user32.EndDialog>	EndDialog
0040103E	E9 07010000	JMP Ice9.0040114A	
00401043	> 3D 01020000	CMF EAX,201	
00401048	75 18	JNZ SHORT Ice9.00401062	
0040104A	8B45 14	MOV EAX,[ARG.4]	
0040104D	50	PUSH EAX	
0040104E	6A 02	PUSH 2	IParam = 2
00401050	68 01000000	PUSH 0A1	wParam = 2
00401055	FF75 08	PUSH [ARG.1]	Message = WM_NCLBUTTONDOWN
00401058	E8 21020000	CALL <JMP.&user32.PostMessageA>	PostMessageA
0040105D	E9 E8000000	JMP Ice9.0040114A	
00401062	> 3D 10010000	CMF EAX,110	
00401067	75 38	JNZ SHORT Ice9.004010A1	
00401069	E8 28020000	CALL <JMP.&kernel32.IsDebuggerPresent>	IsDebuggerPresent
0040106E	83F9 01	CMF EAX,1	
00401071	75 0A	JNZ SHORT Ice9.0040107D	
00401073	6A 00	PUSH 0	Result = 0
00401075	FF75 08	PUSH [ARG.1]	hWnd
00401078	E8 E9010000	CALL <JMP.&user32.EndDialog>	EndDialog
0040107D	> 33C9	XOR EAX,EAX	

Pasos

Ejemplo práctico (II): estudio del crackME (1)

00401000	6A 00	PUSH 0	pModule = NULL
00401002	E8 89020000	CALL <JMP.&kernel32.GetModuleHandleA>	GetModuleHandleA
00401007	A3 94314000	MOV DWORD PTR DS:[403194], EAX	
0040100C	6A 00	PUSH 0	IParam = NULL
0040100E	68 29104000	PUSH Ice9.00401029	DlgProc = Ice9.00401029
00401013	6A 00	PUSH 0	hOwner = NULL
00401015	6A 65	PUSH 65	pTemplate = 65
00401017	FF35 94314000	PUSH DWORD PTR DS:[403194]	hInst = NULL
0040101D	E8 3E020000	CALL <JMP.&user32.DialogBoxParamA>	DialogBoxParamA
00401022	6A 00	PUSH 0	ExitCode = 0
00401024	E8 61020000	CALL <JMP.&kernel32.ExitProcess>	ExitProcess
00401029	55	PUSH EBP	
0040102A	8BEC	MOV EBP, ESP	
0040102C	8B45 0C	MOV EAX, [ARG.2]	
0040102F	83F9 10	CMPEAX, 10	
00401032	> 75 0F	JNZ SHORT Ice9.00401043	
00401034	6A 00	PUSH 0	Result = 0
00401036	FF75 08	PUSH [ARG.1]	hWnd
00401039	E8 28020000	CALL <JMP.&user32.EndDialog>	EndDialog
0040103E	E9 07010000	JMP Ice9.0040114A	
00401043	> 3D 01020000	CMPEAX, 201	
00401048	> 75 18	JNZ SHORT Ice9.00401062	
0040104A	8B45 14	MOV EAX, [ARG.4]	
0040104D	50	PUSH EAX	
0040104E	6A 02	PUSH 2	IParam = 2
00401050	68 71000000	PUSH 0A1	wParam = 2
00401055	FF75 08	PUSH [ARG.1]	Message = WM_NCLBUTTONDOWN
00401058	E8 21020000	CALL <JMP.&user32.PostMessageA>	PostMessageA
0040105D	E9 E8000000	JMP Ice9.0040114A	
00401062	> 3D 10010000	CMPEAX, 10	
00401067	> 75 38	JNZ SHORT Ice9.004010A1	
00401069	E8 28020000	CALL <JMP.&kernel32.IsDebuggerPresent>	IsDebuggerPresent
0040106E	83F9 01	CMPEAX, 1	
00401071	> 75 0A	JNZ SHORT Ice9.0040107D	
00401073	6A 00	PUSH 0	Result = 0
00401075	FF75 08	PUSH [ARG.1]	hWnd
00401078	E8 E9010000	CALL <JMP.&user32.EndDialog>	EndDialog
0040107D	> 33C0	OR EAX, EAX	

Pasos

- 1 Llamada a IsDebuggerPresent → ocultar el Olly

Ejemplo práctico (II): estudio del crackME (1)

00401000	6A 00	PUSH 0	pModule = NULL
00401002	E8 89020000	CALL <JMP.&kernel32.GetModuleHandleA>	GetModuleHandleA
00401007	A3 94314000	MOV DWORD PTR DS:[403194], EAX	
0040100C	6A 00	PUSH 0	IParam = NULL
0040100E	68 29104000	PUSH Ice9.00401029	DlgProc = Ice9.00401029
00401013	6A 00	PUSH 0	hOwner = NULL
00401015	6A 65	PUSH 65	pTemplate = 65
00401017	FF35 94314000	PUSH DWORD PTR DS:[403194]	hInst = NULL
0040101D	E8 3E020000	CALL <JMP.&user32.ShowDialogParamA>	DialogBoxParamA
00401022	6A 00	PUSH 0	ExitCode = 0
00401024	E8 61020000	CALL <JMP.&kernel32.ExitProcess>	ExitProcess
00401029	55	PUSH EBP	
0040102A	8BEC	MOV EBP, ESP	
0040102C	8B45 0C	MOV EAX, [ARG.2]	
0040102F	83F9 10	CMF EAX, 10	
00401032	75 0F	JNG SHORT Ice9.00401043	
00401034	6A 00	PUSH 0	Result = 0
00401036	FF75 08	PUSH [ARG.1]	hWnd
00401039	E8 28020000	CALL <JMP.&user32.EndDialog>	EndDialog
0040103E	E9 07010000	JMP Ice9.0040114A	
00401043	> 3D 01020000	CMF EAX, 201	
00401048	75 18	JNG SHORT Ice9.00401062	
0040104A	8B45 14	MOV EAX, [ARG.4]	
0040104D	50	PUSH EAX	
0040104E	6A 02	PUSH 2	IParam = 2
00401050	68 71000000	PUSH 0A1	wParam = WM_NCLBUTTONDOWN
00401055	FF75 08	PUSH [ARG.1]	hWnd
00401058	E8 21020000	CALL <JMP.&user32.PostMessageA>	PostMessageA
0040105D	E9 E8000000	JMP Ice9.0040114A	
00401062	> 3D 10010000	CMF EAX, 110	
00401067	75 38	JNG SHORT Ice9.004010A1	
00401069	E8 28020000	CALL <JMP.&kernel32.IsDebuggerPresent>	IsDebuggerPresent
0040106E	83F9 01	CMF EAX, 1	
00401071	75 0A	JNG SHORT Ice9.0040107D	
00401073	6A 00	PUSH 0	Result = 0
00401075	FF75 08	PUSH [ARG.1]	hWnd
00401078	E8 E9010000	CALL <JMP.&user32.EndDialog>	EndDialog
0040107D	33C9	MOV EAX, EAX	

Pasos

- 1 Llamada a IsDebuggerPresent → ocultar el Olly
- 2 BP en GetDlgItemTextA e introducimos datos...

Ejemplo práctico (II): estudio del crackME (1)

00401000	68 00020000	PUSH 200	Count = 200 (512.)
00401005	68 B4304000	PUSH Ice9.00403004	Buffer = Ice9.00403004
0040100A	68 E9030000	PUSH 3E9	ControlID = 3E9 (1001.)
0040100F	FF75 00	PUSH [ARG.1]	hWnd
00401012	E9 9C010000	CALL <JMP.&user32.GetDlgItemTextA>	GetDlgItemTextA
00401017	0BC0	OR EAX, EAX	
00401019	> 74 0A	JE SHORT Ice9.004010F5	
0040101B	> 83F8 04	CMF EAX, 4	
0040101E	> 72 05	JBE SHORT Ice9.004010F5	
0040101F	> 83F8 0A	CMF EAX, 0A	
00401023	> 76 15	JBE SHORT Ice9.0040110A	
00401025	> 6A 00	PUSH 0	
00401027	68 04304000	PUSH Ice9.00403004	Style = MB_OK MB_APPLMODAL
0040102C	68 1C304000	PUSH Ice9.0040301C	Title = "Error, Bad Boy"
00401031	6A 00	PUSH 0	Text = "name must be at least 4 chars"
00401033	E8 70010000	CALL <JMP.&user32.MessageBoxA>	hOwner = NULL
00401038	EB 35	JMP SHORT Ice9.0040113F	MessageBoxA
0040103A	68 00020000	PUSH 200	
0040103D	68 98314000	PUSH Ice9.00403198	Count = 200 (512.)
00401041	68 EA030000	PUSH 3EA	Buffer = Ice9.00403198
00401046	FF75 00	PUSH [ARG.1]	ControlID = 3EA (1002.)
0040104B	E9 4B010000	CALL <JMP.&user32.GetDlgItemTextA>	hWnd
00401050	0BC0	OR EAX, EAX	GetDlgItemTextA
00401052	> 75 15	JBE SHORT Ice9.0040113A	
00401054	6A 00	PUSH 0	
00401056	68 04304000	PUSH Ice9.00403004	Style = MB_OK MB_APPLMODAL
0040105B	68 68304000	PUSH Ice9.00403060	Title = "Error, Bad Boy"
00401060	6A 00	PUSH 0	Text = "Where is the serial, Mr Cracker ? "
00401062	E8 40010000	CALL <JMP.&user32.MessageBoxA>	hOwner = NULL
00401067	> 95	JMP SHORT Ice9.0040113F	MessageBoxA
00401069	> B8 14000000	CALL Ice9.00401153	
0040106E	> EB 09	JMP SHORT Ice9.0040114A	
00401071	> B8 00000000	MOV EAX, 0	
00401074	> C9	LEAVE	
00401076	> C9 1000	RETN 10	
00401079	> B8 01000000	MOV EAX, 1	
0040107C	> C9	LEAVE	
0040107E	> C2 1000	RETN 10	

Pasos

- 1 Llamada a IsDebuggerPresent → ocultar el Olly
- 2 BP en GetDlgItemTextA e introducimos datos...
- 3 Alcanzamos código de la aplicación (CTRL + F9)

Ejemplo práctico (II): estudio del crackME (2)

00401000	68 00220000	PUSH 200	Count = 200 (512.)
00401005	68 04304000	PUSH Ice9.00403004	Buffer = Ice9.00403004
0040100A	68 E9030000	PUSH 3E9	ControlID = 3E9 (1001.)
0040100F	FF75 08	PUSH [ARG.1]	hMnd
00401012	E9 95010000	CALL <JMP.&user32.GetDlgItemTextA>	GetDlgItemTextA
00401017	0BC0	OR EAX,EAX	
00401019	74 0A	JE SHORT Ice9.004010F5	
0040101B	83F9 04	CMR EAX,4	
0040101E	72 05	JBE SHORT Ice9.004010F5	
0040101F	83F9 0A	CMR EAX,0A	
00401019	76 15	JBE SHORT Ice9.0040110A	
004010F5	6A 00	PUSH 0	
004010F7	68 04304000	PUSH Ice9.00403004	Style = MB_OK MB_APPLMODAL
004010FC	68 1C304000	PUSH Ice9.0040301C	Title = "Error, Bad Boy"
00401091	6A 00	PUSH 0	Text = "name must be at least 4 chars"
00401093	E8 70010000	CALL <JMP.&user32.MessageBoxA>	hOwner = NULL
00401098	EB 95	JMP SHORT Ice9.0040113F	MessageBoxA
0040109A	68 00220000	PUSH 200	
0040109F	68 9E314000	PUSH Ice9.00403198	Count = 200 (512.)
004010A4	68 EA030000	PUSH 3EA	Buffer = Ice9.00403198
004010A7	FF75 08	PUSH [ARG.1]	ControlID = 3EA (1002.)
004010AC	E9 4B010000	CALL <JMP.&user32.GetDlgItemTextA>	hMnd
004010B1	0BC0	OR EAX,EAX	GetDlgItemTextA
004010B3	75 15	JNZ SHORT Ice9.0040113A	
004010B5	6A 00	PUSH 0	
004010B7	68 04304000	PUSH Ice9.00403004	Style = MB_OK MB_APPLMODAL
004010BA	68 68304000	PUSH Ice9.00403060	Title = "Error, Bad Boy"
004010B3	6A 00	PUSH 0	Text = "Where is the serial, Mr Cracker ?"
004010B5	E8 40010000	CALL <JMP.&user32.MessageBoxA>	hOwner = NULL
004010B8	EB 95	JMP SHORT Ice9.0040113F	MessageBoxA
004010BA	E8 14000000	CALL Ice9.00401153	
004010BF	EB 09	JMP SHORT Ice9.0040114A	
004010C1	B8 00000000	MOV EAX,0	
004010C4	C9	LEAVE	
004010C7	C2 1000	RETN 10	
004010CA	B8 01000000	MOV EAX,1	
004010CD	C9	LEAVE	
004010D0	C2 1000	RETN 10	

Condiciones del nombre

- EAX: longitud del nombre

Ejemplo práctico (II): estudio del crackME (2)

00401000	68 0020000	PUSH 200	Count = 200 (512.)
00401005	68 B4304000	PUSH Ice9.004030B4	Buffer = Ice9.004030B4
0040100A	68 E9030000	PUSH 3E9	ControlID = 3E9 (1001.)
0040100F	FF75 08	PUSH [ARG_1]	hWnd
00401012	E9 95010000	JMP SHORT Ice9.00401012	GetDlgItemTextA
00401017	00C0	OR EAX,EAX	
00401019	74 0A	JE SHORT Ice9.004010F5	
0040101B	83F9 04	CMPL EAX,4	
0040101E	72 05	JBE SHORT Ice9.004010F5	
0040101F	83F9 0A	CMPL EAX,0A	
00401019	76 15	JBE SHORT Ice9.0040110A	
004010F5	6A 00	MOVB AL,0	
004010F7	68 04304000	PUSH Ice9.004030B4	Style = MB_OK MB_APPLMODAL
004010FC	68 1C304000	PUSH Ice9.0040301C	Title = "Error, Bad Boy"
00401091	6A 00	PUSH 0	Text = "name must be at least 4 chars"
00401093	E8 70010000	CALL <JMP.&user32.MessageBoxA>	hOwner = NULL
00401088	EB 3E	JMP SHORT Ice9.0040113F	MessageBoxA
00401094	68 00200000	PUSH 200	
0040109F	68 9E314000	PUSH Ice9.00403198	Count = 200 (512.)
00401114	68 EA030000	PUSH 3EA	Buffer = Ice9.00403198
00401119	FF75 08	PUSH [ARG_1]	ControlID = 3EA (1002.)
0040111C	E9 4B010000	JMP <JMP.&user32.GetDlgItemTextA>	hWnd
00401121	00C0	OR EAX,EAX	GetDlgItemTextA
00401123	75 15	JNZ SHORT Ice9.0040113A	
00401125	6A 00	PUSH 0	Style = MB_OK MB_APPLMODAL
00401127	68 04304000	PUSH Ice9.004030B4	Title = "Error, Bad Boy"
0040112C	68 68304000	PUSH Ice9.00403060	Text = "Where is the serial, Mr Cracker ? "
00401131	6A 00	PUSH 0	hOwner = NULL
00401133	E8 40010000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA
00401138	EB 3E	JMP SHORT Ice9.0040113F	
0040113A	CALL 14000000	CALL Ice9.00401153	
0040113F	EB 09	JMP SHORT Ice9.0040114A	
00401141	B8 00000000	MOVB EAX,0	
00401146	C9	LEAVE	
00401147	C2 1000	RETN 10	
0040114A	B8 01000000	MOVB EAX,1	
0040114F	C9	LEAVE	
00401150	C2 1000	RETN 10	

Condiciones del nombre

- EAX: longitud del nombre
- $EAX > 4 \ \&\& \ EAX \leq 0Ah$

Ejemplo práctico (II): estudio del crackME (2)

00401000	68 00220000	PUSH 200	Count = 200 (512.)
00401005	68 04304000	PUSH Ice9.00403004	Buffer = Ice9.00403004
0040100A	68 E9030000	PUSH 3E9	ControlID = 3E9 (1001.)
0040100F	FF75 08	PUSH [ARG.1]	hMnd
00401012	E9 95010000	CALL <JMP.>user32.GetDlgItemTextA	GetDlgItemTextA
00401017	0BC0	OR EAX,EAX	
00401019	< 74 0A	JBE SHORT Ice9.004010F5	
0040101B	< 83F9 04	CMPL EAX,4	
0040101E	< 72 05	JBE SHORT Ice9.004010F5	
0040101F	< 83F9 0A	CMPL EAX,0A	
00401023	< 76 15	JBE SHORT Ice9.0040110A	
00401025	> 6A 00	PUSH 0	
00401027	68 04304000	PUSH Ice9.00403004	Style = MB_OK MB_APPLMODAL
0040102C	68 1C304000	PUSH Ice9.0040301C	Title = "Error, Bad Boy"
00401031	6A 00	PUSH 0	Text = "name must be at least 4 chars"
00401033	E8 70010000	CALL <JMP.>user32.MessageBoxA	hOwner = NULL
00401038	EB 95	OR EAX,EAX	MessageBoxA
0040103A	< 68 00220000	PUSH 200	
0040103F	< 68 9E314000	PUSH Ice9.00403198	Count = 200 (512.)
00401044	< 68 EA030000	PUSH 3EA	Buffer = Ice9.00403198
00401049	< FF75 08	PUSH [ARG.1]	ControlID = 3EA (1002.)
0040104E	E9 4B010000	CALL <JMP.>user32.GetDlgItemTextA	hMnd
00401053	0BC0	OR EAX,EAX	GetDlgItemTextA
00401055	< 75 15	JNZ SHORT Ice9.0040113A	
00401057	> 6A 00	PUSH 0	
00401059	68 04304000	PUSH Ice9.00403004	Style = MB_OK MB_APPLMODAL
0040105E	68 68304000	PUSH Ice9.00403060	Title = "Error, Bad Boy"
00401063	6A 00	PUSH 0	Text = "Where is the serial, Mr Cracker ?"
00401065	E8 40010000	CALL <JMP.>user32.MessageBoxA	hOwner = NULL
0040106A	EB 95	OR EAX,EAX	MessageBoxA
0040106C	E9 14000000	JMP SHORT Ice9.00401153	
0040106F	> EB 09	JMP SHORT Ice9.0040114A	
00401071	> B8 00000000	MOV EAX,0	
00401074	> C9	LEAVE	
00401076	> C2 1000	RETN 10	
00401078	> B8 01000000	MOV EAX,1	
0040107B	> C9	LEAVE	
0040107D	> C2 1000	RETN 10	

Condiciones del nombre

- EAX: longitud del nombre
- $EAX > 4 \ \&\& \ EAX \leq 0Ah$
- Si longitud de serial = 0 \rightarrow MessageBoxA

Ejemplo práctico (II): estudio del crackME (3)

Address	Disassembly	Text string
004010D5	PUSH Icx9,004030B4	ASCII "TripleTordo"
004010F7	PUSH Icx9,004030B4	ASCII "Error, Bad Boy"
004010FC	PUSH Icx9,0040301C	ASCII "name must be at least 4 chars"
00401127	PUSH Icx9,004030B4	ASCII "Error, Bad Boy"
0040112C	PUSH Icx9,00403060	ASCII "Where is the serial, Mr Cracker ? "
00401156	PUSH Icx9,004030B4	ASCII "TripleTordo"
00401167	MOV EDI,Icx9,004030B4	ASCII "TripleTordo"
004011E6	PUSH Icx9,004030B7	ASCII "pleTordo"
00401206	PUSH Icx9,004030B4	(Initial CPU selection)
0040120D	PUSH Icx9,004030B3	ASCII "Good Job, Now write a keygen !! Register
00401219	PUSH Icx9,00403013	ASCII "Good boy"
0040121E	PUSH Icx9,004030B3	ASCII "Good Job, Now write a keygen !! Register
00401235	PUSH Icx9,004030B4	ASCII "Error, Bad Boy"
0040123A	PUSH Icx9,0040303A	ASCII "Hey!, you are DK?, its for Newbies..."
0040124A	PUSH Icx9,004030B4	ASCII "Error, Bad Boy"
0040124F	PUSH Icx9,00403060	ASCII "Where is the serial, Mr Cracker ? "

- Búsqueda de cadenas referenciadas
- Cadenas de 'chico bueno' y 'chico malo'

Ejemplo práctico (II): estudio del crackME (3)

004011F0	68	0B000000	CALL Ice9.004012B0	String2 = ""
004011F5	68	C8304000	PUSH Ice9.004030C8	String1 = ""
004011FA	68	98314000	PUSH Ice9.00403198	lstrcmpA
004011FF	E8	98000000	CALL <JMP.&kernel32.lstrcmpA>	
00401204	00	0BC0	OR EAX,EAX	
00401206	>	75 2B	JNZ SHORT Ice9.00401233	
00401208	68	84304000	PUSH Ice9.004030B4	ASCII "DeAtH HaS c0ME"
0040120D	68	83304000	PUSH Ice9.004030B3	ASCII "Good Job, Now write a
00401212	E8	99000000	CALL Ice9.004012B0	
00401217	6A	00	PUSH 0	Style = MB_OK MB_APPLMODAL
00401219	68	13304000	PUSH Ice9.00403013	Title = "Good boy"
0040121E	68	83304000	PUSH Ice9.004030B3	Text = "Good Job, Now write
00401223	6A	00	PUSH 0	hOwner = NULL
00401225	E8	4E000000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA
0040122A	6A	00	PUSH 0	ExitCode = 0
0040122C	E8	59000000	CALL <JMP.&kernel32.ExitProcess>	ExitProcess
00401231	EB	13	JMP SHORT Ice9.00401246	
00401233	>	6A 00	PUSH 0	Style = MB_OK MB_APPLMODAL
00401235	68	04304000	PUSH Ice9.00403004	Title = "Error, Bad Boy"
0040123A	68	3A304000	PUSH Ice9.0040303A	Text = "Hey!, you are OK?, i
0040123F	6A	00	PUSH 0	hOwner = NULL
00401241	E8	32000000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA
00401246	>	EB 13	JMP SHORT Ice9.0040125B	
00401248	6A	00	PUSH 0	Style = MB_OK MB_APPLMODAL
0040124A	68	04304000	PUSH Ice9.00403004	Title = "Error, Bad Boy"
0040124F	68	60304000	PUSH Ice9.00403060	Text = "where is the serial,
00401254	6A	00	PUSH 0	hOwner = NULL
00401256	E8	1D000000	CALL <JMP.&user32.MessageBoxA>	MessageBoxA

- Búsqueda de cadenas referenciadas
- Cadenas de 'chico bueno' y 'chico malo'

Ejemplo práctico (II): estudio del crackME (3)

004011F0	.E8	00000000	JMP SHORT Ice9.00401200
004011F5	.E8	C8304000	PUSH Ice9.00403000
004011FA	.E8	98314000	PUSH Ice9.00403198
004011FF	.E8	98000000	CALL <JMP.&kernel32.lstrcmpA>
00401204	.0BC0		OR EAX,EAX
00401206	.J	75 2B	JNZ SHORT Ice9.00401233
00401208	.E8	84304000	PUSH Ice9.004030B4
0040120D	.E8	83304000	PUSH Ice9.00403083
00401212	.E8	99000000	CALL Ice9.004012B0
00401217	.E8	6A 00	PUSH 0
00401219	.E8	13304000	PUSH Ice9.00403013
0040121E	.E8	83304000	PUSH Ice9.00403083
00401223	.E8	6A 00	PUSH 0
00401225	.E8	4E000000	CALL <JMP.&user32.MessageBoxA>
0040122A	.E8	6A 00	PUSH 0
0040122C	.E8	59000000	CALL <JMP.&kernel32.ExitProcess>
00401231	.EB	13	JMP SHORT Ice9.0040123E
00401233	.E8	6A 00	PUSH 0
00401235	.E8	04304000	PUSH Ice9.00403004
0040123A	.E8	3A304000	PUSH Ice9.0040303A
0040123F	.E8	6A 00	PUSH 0
00401241	.E8	32000000	CALL <JMP.&user32.MessageBoxA>
00401246	.EB	13	JMP SHORT Ice9.0040125B
00401248	.E8	6A 00	PUSH 0
0040124A	.E8	04304000	PUSH Ice9.00403004
0040124F	.E8	60304000	PUSH Ice9.00403060
00401254	.E8	6A 00	PUSH 0
00401256	.E8	1D000000	CALL <JMP.&user32.MessageBoxA>


```
String2 = ""
String1 = ""
lstrcmpA

ASCII "DeAtH HaS cOMe"
ASCII "Good Job, Now write a"

Style = MB_OK|MB_APPLMODAL
Title = "Good boy"
Text = "Good Job, Now write a"
hOwner = NULL
MessageBoxA
ExitCode = 0
ExitProcess

Style = MB_OK|MB_APPLMODAL
Title = "Error, Bad Boy"
Text = "Hey!, you are OK?", i
hOwner = NULL
MessageBoxA

Style = MB_OK|MB_APPLMODAL
Title = "Error, Bad Boy"
Text = "where is the serial,"
hOwner = NULL
MessageBoxA
```

- Búsqueda de cadenas referenciadas
- Cadenas de 'chico bueno' y 'chico malo'
- lstrcmpA: comparación de cadenas
 - Valor de EAX determina igualdad

Ejemplo práctico (III): algoritmo de generación (1)

```

00401153 | 55          | PUSH EBP
00401154 | 68EC       | MOV EBP,ESP
00401156 | 68 B4304000 | PUSH Ice9.004030B4
0040115B | E8 42010000 | CALL <JMP.&kerne132.1strlenA>
00401160 | 6A 6A314000 | MOV ICE9.0040316A
00401165 | 9902       | MOV DWORD PTR DS:[EDX],EAX
00401167 | BF B4304000 | MOV EDI,Ice9.004030B4
0040116C | 66 19314000 | MOV ESI,Ice9.00403119
00401171 | 33C0       | XOR EAX,EAX
00401173 | 33D0       | XOR EBX,EBX
00401175 | 33C9       | XOR ECX,ECX
00401177 | 33D2       | XOR EDX,EDX
00401179 | BA 6A314000 | MOV EDI,Ice9.0040316A
0040117E | 8B12       | MOV EDX,DWORD PTR DS:[EDX]
00401180 | 33C3 01    | ADD EBX,1
00401183 | 38D3       | CMP EDX,EBX
00401185 | 74 15      | JE SHORT Ice9.0040119C
00401187 | 9A07       | MOV AL,BYTE PTR DS:[EDI]
00401189 | 3C 5A      | CMP AL,5A
0040118B | 7E 05      | JLE SHORT Ice9.00401192
0040118D | 83C8       | ADD ECX,ERM
0040118F | 47         | INC EDI
00401190 | EB EE      | JMP SHORT Ice9.00401180
00401192 | 3C 41      | CMP AL,41
00401194 | 7D 02      | JGE SHORT Ice9.00401198
00401196 | EB 02      | JMP SHORT Ice9.0040119A
00401198 | 84 2C      | ADD AL,2C
0040119A | EB F1      | JMP SHORT Ice9.00401180
0040119C | 81C1 9A020000 | ADD ECX,29A
004011A2 | 69C9 39300000 | IMUL ECX,ECX,3039
004011A6 | 69C9 17     | SUB ECX,1
004011A8 | 68C9 09     | IMUL ECX,ECX,9
004011AE | 33D0       | XOR EBX,EBX
004011B0 | 8BC1       | MOV ECX,ECX
004011B2 | B9 0A000000 | MOV ECX,0A
004011B7 | 33D2       | XOR EDX,EDX
004011B9 | F7F1       | DTU ECX
004011BB | 80C2 30     | ADD DL,30
004011BE | 891433     | MOV BYTE PTR DS:[EBX+ESI],DL
004011C1 | 83C3 01     | ADD ECX,1
004011C4 | 83F8 00     | CMP EAX,0
004011C7 | 74 02      | JE SHORT Ice9.004011CB
004011C9 | EB EC      | JMP SHORT Ice9.004011B7
004011CB | BF C2040000 | MOV EDI,Ice9.004030C3
004011D0 | 804433 FF  | MOV AL,BYTE PTR DS:[EBX+ESI-1]
004011D4 | 8B07       | MOV BYTE PTR DS:[EDI],AL
004011D6 | 47         | INC EDI
004011D7 | 48         | DEC EBX
004011D8 | C7FB 00    | CMP EBX,0
004011DE | 75 F3      | JNE SHORT Ice9.004011D0
004011DD | C607 00    | MOV BYTE PTR DS:[EDI],0
004011E0 | 803D B4304000 | LEA EDI,DWORD PTR DS:[4030B4]
004011E4 | 68 B7304000 | PUSH Ice9.004030E7
004011E6 | 68 C2040000 | PUSH Ice9.004030C3
004011EB | E8 BB000000 | CALL Ice9.004012E0
004011F5 | 68 C3040000 | PUSH Ice9.004030C3
004011FF | 68 98314000 | PUSH Ice9.00403198
004011FF | E8 98000000 | CALL <JMP.&kerne132.1strncpyA>

```

[String = "D

ASCII "DeA

ASCII "tH H

[String2 = "

[String1 = "

[!strncpyA

- 0x401153: Inicio algoritmo
- BP para estudio 'en caliente'

Ejemplo práctico (III): algoritmo de generación (1)

```

00401153 | 55 | PUSH EBP
00401154 | 68 | MOV EBP,ESP
00401156 | 68 | PUSH Ice9.004030B4
0040115B | E8 | EB 4201000 | String = "
00401160 | 6A | MOV EDI,Ice9.0040316A | strlenA
00401165 | 99 | MOV EDI,Ice9.0040316A
00401167 | BF | MOV EDI,Ice9.004030B4
0040116C | 95 | MOV ESI,Ice9.00403119 | ASCII "DeA
00401171 | 33 | XOR EBX,EBX
00401173 | 33 | XOR EBX,EBX
00401175 | 33 | XOR ECX,ECX
00401177 | 33 | XOR EDX,EDX
00401179 | BA | MOV EDI,Ice9.0040316A
0040117E | 8B | MOV EDI,WORD PTR DS:[EDX]
00401180 | 83 | ADD EBX,1
00401183 | 3B | CMP EDX,EBX
00401185 | 74 | JE SHORT Ice9.0040119C
00401187 | 8B | MOV AL,BYTE PTR DS:[EDI]
00401189 | 3C | CMP AL,0A
0040118B | 7E | JLE SHORT Ice9.00401192
0040118D | 83 | ADD ECX,EBX
0040118F | 47 | INC EDI
00401190 | EB | JMP SHORT Ice9.00401188
00401192 | 3C | CMP AL,1
00401194 | 7D | JGE SHORT Ice9.00401198
00401196 | EB | JMP SHORT Ice9.0040119A
00401198 | 84 | ADD AL,2C
0040119A | 7E | JLE SHORT Ice9.00401188
0040119C | 83 | ADD ECX,29A
004011A2 | 69 | INTP ECX,ECX,0039
004011A6 | 69 | INTP ECX,1
004011AB | 69 | INTP ECX,9
004011AE | 33 | XOR EBX,EBX
004011B0 | 8B | MOV ECX,ECX
004011B2 | B9 | MOV ECX,0A
004011B7 | 33 | XOR EDX,EDX
004011B9 | F7 | DTB ECX
004011BB | 80 | ADD DL,30
004011BE | 8B | MOV BYTE PTR DS:[EBX+ESI],DL
004011C1 | 8B | ADD EBX,1
004011C4 | 3B | CMP EBX,0
004011C7 | 74 | JE SHORT Ice9.004011CB
004011C9 | EB | JMP SHORT Ice9.004011B7
004011CB | BF | MOV EDI,Ice9.004030C3
004011D0 | BA | MOV AL,BYTE PTR DS:[EBX+ESI-1]
004011D4 | 8B | MOV BYTE PTR DS:[EDI],AL
004011D6 | 47 | INC EDI
004011D7 | 4B | DEC EBX
004011D8 | 3B | CMP EBX,0
004011DB | 75 | JNZ SHORT Ice9.004011D0
004011DD | C7 | MOV BYTE PTR DS:[EDI],0
004011E0 | 80 | LER EDI,WORD PTR DS:[4030B4] | ASCII "tH H
004011E4 | 68 | PUSH Ice9.004030B7
004011E6 | 68 | PUSH Ice9.004030C3
004011EB | E8 | BE000000 | String2 = ""
004011ED | 68 | PUSH Ice9.00403120
004011F5 | 68 | PUSH Ice9.004030C3
004011FF | 68 | PUSH Ice9.00403193 | String1 = ""
004011FF | E8 | 90000000 | CALL <JMP_&kerne132.1stronpA> | strlenpA

```

- 0x401153: Inicio algoritmo
- BP para estudio 'en caliente'
- 0x40316A: buffer longitud

Ejemplo práctico (III): algoritmo de generación (1)

```

00401153 | 55          | PUSH EBP
00401154 | . 88EC     | MOV EBP,ESP
00401156 | . 68 B4304000 | PUSH Ice9.004030B4
0040115B | . E8 42010000 | CALL <JMP.&kerne132.1strlenA>
00401160 | . 6A 6A314000 | MOV EDI,Ice9.0040316A
00401165 | . 8902     | MOV EAX,EAX
00401167 | . BF B4304000 | MOV EDI,Ice9.004030B4
0040116C | . 8E 19314000 | XOR ESI,Ice9.00403119
00401171 | . 33C0     | XOR EAX,EAX
00401173 | . 33D0     | XOR EBX,EBX
00401175 | . 33C9     | XOR ECX,ECX
00401177 | . 33D2     | XOR EDX,EDX
00401179 | . 6A 6A314000 | MOV EDI,Ice9.0040316A
0040117E | . 8B12     | MOV EDI,OFFSET PTR DS:[EDX]
00401180 | > 83C3 01  | CMP EAX,EBX
00401183 | . 38D3     | CMP AL,EBX
00401185 | . 74 15   | JE SHORT Ice9.0040119C
00401187 | . 8A07     | MOV AL,BYTE PTR DS:[EDI]
00401189 | . 3C 5A   | CMP AL,5A
0040118B | . 7E 05   | JLE SHORT Ice9.00401192
0040118D | > 83C8     | ADD ECX,EAX
0040118F | . 47     | INC EDI
00401190 | . EB EE   | JPF SHORT Ice9.00401180
00401192 | > 3C 41   | CMP AL,41
00401194 | . 7D 02   | JGE SHORT Ice9.00401198
00401196 | . EB 02   | JPF SHORT Ice9.0040119A
00401198 | > 84 2C   | ADD AL,2C
0040119A | . EB F1   | JPF SHORT Ice9.00401180
0040119C | > 81C1 9A020000 | ADD ECX,29A
004011A2 | . 69C9 39300000 | INVL ECX,ECX,3039
004011A6 | . 83E9 17   | SUB ECX,17
004011AB | . 68C9 09   | INVL ECX,ECX,9
004011AE | . 330B     | XOR EBX,EBX
004011B0 | . 88C1     | MOV EBX,ECX
004011B2 | . B9 0A000000 | MOV ECX,0A
004011B7 | > 33D2     | XOR EDX,EDX
004011B9 | . F7F1     | DTU ECX,1
004011BB | . 80C2 30   | ADD DL,30
004011BE | . 891433   | MOV BYTE PTR DS:[EBX+ESI],DL
004011C1 | . 83E9 01   | ADD EBX,1
004011C4 | . 83F8 00   | CMP EBX,0
004011C7 | . 74 02   | JE SHORT Ice9.004011CB
004011C9 | . EB EC   | JPF SHORT Ice9.004011B7
004011CB | > . BF C2040000 | MOV EDI,Ice9.004030C3
004011D0 | > . 8A433 FF | MOV AL,BYTE PTR DS:[EBX+ESI-1]
004011D4 | . 8307     | MOV BYTE PTR DS:[EDI],AL
004011D6 | . 47     | INC EDI
004011D7 | . 4B     | DEC EBX
004011D8 | . 83FB 00   | CMP EBX,0
004011DE | . 75 F3   | JNE SHORT Ice9.004011D0
004011DD | . C607 00   | MOV BYTE PTR DS:[EDI],0
004011E0 | . 803D B4304000 | LEA EDI,OFFSET PTR DS:[4030B4]
004011E4 | . 68 B4304000 | PUSH Ice9.004030B4
004011E6 | . 68 C2040000 | PUSH Ice9.004030C3
004011F0 | . E8 B0000000 | CALL Ice9.004012E0
004011F5 | . 68 C2040000 | PUSH Ice9.004030C3
004011FF | . 68 98314000 | PUSH Ice9.00403198
004011FF | . E8 98000000 | CALL <JMP.&kerne132.1strncpyA>

```

- 0x401153: Inicio algoritmo
- BP para estudio 'en caliente'
- 0x40316A: buffer longitud
- EDI: buffer cadena
- ESI: otro buffer :)
- EAX=EBX=ECX=EDX=0
- EDX=longitud

Ejemplo práctico (III): algoritmo de generación (1)

```

00401153  55      PUSH    EBP
00401154  8BC8    MOV    EBP,ESP
00401156  68 B4304000  PUSH   Ice9.004030B4
0040115B  EB 42010000  CALL  <JMP.&kerne132.1strlenA>
00401160  BA 6A314000  MOV    EDI,Ice9.0040316A
00401165  8902    MOV    DWORD PTR DS:[EDI],EAX
00401167  BF B4304000  MOV    EDI,Ice9.004030B4
0040116C  66 19314000  MOV    ESI,Ice9.00403119
00401171  33C0    XOR    EAX,EAX
00401173  330B    XOR    EBX,EBX
00401175  33C9    XOR    ECX,ECX
00401177  33D2    XOR    EDX,EDX
00401179  BA 6A314000  MOV    EDX,Ice9.0040316A
0040117E  8B12    MOV    EBX,ESI
00401180  83C3 01    CMP    EDX,EBX
00401183  74 15    JBE   SHORT Ice9.0040119C
00401187  8A07    MOV    AL,BYTE PTR DS:[EDI]
00401189  3C 5A    CMP    AL,5A
0040118B  7E 05    JLE   SHORT Ice9.00401192
0040118D  83C8    ADD    ECX,EAX
0040118F  47      INC    EDI
00401190  EB EE    JMP   SHORT Ice9.00401180
00401192  3C 41    CMP    AL,41
00401194  7D 02    JGE   SHORT Ice9.00401198
00401196  EB 02    JMP   SHORT Ice9.0040119A
00401198  84 2C    MOV    AL,2C
0040119A  7E F1    JLE   SHORT Ice9.00401180
0040119C  81C1 9A0200  ADD    ECX,9A0200
004011A2  68C9 393000  MOV    ECX,ECX,393000
004011A6  68C9 17    MOV    ECX,ECX,17
004011A8  68C9 09    IMUL  ECX,ECX,9
004011AE  330B    XOR    EBX,EBX
004011B0  8BC1    MOV    ECX,ECX
004011B2  B9 0A000000  MOV    ECX,0A
004011B7  33D2    XOR    EDX,EDX
004011B9  F7F1    DIV    ECX
004011BB  80C2 30    ADD    DL,30
004011BE  891433    MOV    BYTE PTR DS:[EBX+ESI],DL
004011C1  8BC3 01    MOV    ECX,1
004011C4  83F8 00    CMP    EAX,0
004011C7  74 02    JE    SHORT Ice9.004011CB
004011C9  EB EC    JMP   SHORT Ice9.004011B7
004011CB  68 C9304000  MOV    EDI,Ice9.004030C3
004011D0  8A433 FF    MOV    AL,BYTE PTR DS:[EBX+ESI-1]
004011D4  8B07    MOV    BYTE PTR DS:[EDI],AL
004011D6  47      INC    EDI
004011D7  4B      DEC    EBX
004011D8  C7FB 00    CMP    EBX,0
004011DE  75 F3    JNB   SHORT Ice9.004011D0
004011DD  C607 00    MOV    BYTE PTR DS:[EDI],0
004011E0  803D B4304000  LEA   EDI,DWORD PTR DS:[4030B4]
004011E4  68 87304000  PUSH   Ice9.004030E7
004011E6  68 C9304000  PUSH   Ice9.004030C3
004011EB  EB B0000000  CALL  Ice9.004012E0
004011F5  68 C9304000  PUSH   Ice9.004030C3
004011FF  68 98314000  PUSH   Ice9.00403198
004011FF  EB 98000000  CALL  <JMP.&kerne132.1strncpyA>

```

[String = "D

ASCII "DeA

ASCII "tH H

[String2 = ""
String1 = ""
IstrlenA

- 0x401153: Inicio algoritmo
- BP para estudio 'en caliente'
- 0x40316A: buffer longitud
- EDI: buffer cadena
- ESI: otro buffer :)
- EAX=EBX=ECX=EDX=0
- EDX=longitud
- EBX=contador

Ejemplo práctico (III): algoritmo de generación (2)

```

00401180 > 83C3 01 ADD EBX,1
00401183 . 3BD3 CMP EDX,EBX
00401185 .> 74 15 JE SHORT Ice9.0040119C
00401187 . 8A07 MOV AL,BYTE PTR DS:[EDI]
00401189 . 3C 5A CMP AL,5A
0040118B .> 7E 05 JLE SHORT Ice9.00401192
0040118D > 03C8 ADD ECX,EAX
0040118F . 47 INC EDI
00401190 > EB EE JMP SHORT Ice9.00401180
00401192 > 3C 41 CMP AL,41
00401194 .> 7D 02 JGE SHORT Ice9.00401198
00401196 > EB 02 JMP SHORT Ice9.0040119A
00401198 > 04 2C ADD AL,2C
0040119A > EB F1 JMP SHORT Ice9.00401180
0040119C > 81C1 9A020001 ADD ECX,29A

```

```

EDI = buffer del nombre
EDX = longitud del nombre
EBX = 1
while EBX < EDX do
    AL = carácter apuntado por EDI
    if AL ≤ 0x5A and AL ≥ 0x41 then
        | AL+ = 0x2C
    end
    ECX+ = EAX
    EDI ++
end

```

Ejemplo práctico (III): algoritmo de generación (2)

```

00401180 > 83C3 01      ADD EBX,1
00401183 . 3BD3        CMP EDX,EBX
00401185 .<v 74 15      JE SHORT Ice9.0040119C
00401187 . 8A07        MOV AL,BYTE PTR DS:[EDI]
00401189 . 3C 5A       CMP AL,5A
0040118B . 7E 05       JLE SHORT Ice9.00401192
0040118D > 03C8        ADD ECX,EAX
0040118F . 47          INC EDI
00401190 .^ EB EE       JMP SHORT Ice9.00401180
00401192 > 3C 41       CMP AL,41
00401194 .<v 7D 02      JGE SHORT Ice9.00401198
00401196 .<v EB 02      JMP SHORT Ice9.0040119A
00401198 .<v 04 2C      ADD AL,2C
0040119A .<v EB F1      JMP SHORT Ice9.00401180
0040119C > 81C1 9A020001 ADD ECX,29A

```

```

EDI = buffer del nombre
EDX = longitud del nombre
EBX = 1
while EBX < EDX do
    AL = carácter apuntado por EDI
    if AL ≤ 0x5A and AL ≥ 0x41 then
        AL+ = 0x2C
    end
    ECX+ = EAX
    EDI + +
end

```

```

0040119C > 81C1 9A020001 ADD ECX,29A
004011A2 . 69C9 39300001 IMUL ECX,ECX,3039
004011A8 . 83E9 17      SUB ECX,17
004011AB . 6BC9 09      IMUL ECX,ECX,9
004011AE . 33DB        XOR EBX,EBX
004011B0 . 8BC1        MOV EAX,ECX
004011B2 . B9 0A000000 MOV ECX,0A
004011B7 > 33D2        XOR EDX,EDX
004011B9 . F7F1        DIV ECX
004011BB . 80C2 30     ADD DL,30
004011BE . 881433     MOV BYTE PTR DS:[EBX+ESI],DL
004011C1 . 83C3 01     ADD EBX,1
004011C4 . 83F8 00     CMP EAX,0
004011C7 .<v 74 02     JE SHORT Ice9.004011CB
004011C9 .^ EB EC     JMP SHORT Ice9.004011B7

```

Ejemplo práctico (III): algoritmo de generación (2)

```

00401180 > 83C3 01      ADD EBX,1
00401183 . 3B03        CMP EDX,EBX
00401185 > 74 15       JE SHORT Ice9.0040119C
00401187 . 8A07        MOV AL,BYTE PTR DS:[EDI]
00401189 . 3C 5A       CMP AL,5A
0040118B . 7E 05       JLE SHORT Ice9.00401192
0040118D > 03C8       ADD ECX,EAX
0040118F > 47         INC EDI
00401190 > EB EE       JMP SHORT Ice9.00401180
00401192 > 3C 41       CMP AL,41
00401194 > 7D 02       JGE SHORT Ice9.00401198
00401196 > EB 02       JMP SHORT Ice9.0040119A
00401198 > 04 2C       ADD AL,2C
0040119A > EB F1       JMP SHORT Ice9.0040118D
0040119C > 81C1 9A020001 ADD ECX,29A

```

```

EDI = buffer del nombre
EDX = longitud del nombre
EBX = 1
while EBX < EDX do
    AL = carácter apuntado por EDI
    if AL ≤ 0x5A and AL ≥ 0x41 then
        AL += 0x2C
    end
    ECX += EAX
    EDI ++
end

```

```

0040119C > 81C1 9A020001 ADD ECX,29A
004011A2 > 69C9 39300001 IMUL ECX,ECX,3039
004011A8 > 83E9 17      SUB ECX,17
004011AB > 6BC9 09      IMUL ECX,ECX,9
004011AE > 8BC1        MOV EAX,ECX
004011B0 . B9 0A000000 MOV ECX,0A
004011B2 > 33D2        XOR EDX,EDX
004011B7 > F7F1        DIV ECX
004011B9 . 90C2 30     ADD DL,30
004011BB . 801433     MOV BYTE PTR DS:[EBX+ESI],DL
004011BE . 83C3 01     ADD EBX,1
004011C0 . 83F8 00     CMP EAX,0
004011C2 > 74 02       JE SHORT Ice9.004011CB
004011C4 . EB EC       JMP SHORT Ice9.004011B7

```

```

ECX+ = 0x29A
ECX* = 0x3039
ECX- = 0x17
ECX* = 0x9

```

Ejemplo práctico (III): algoritmo de generación (3)

```

0040119C > 481C1 9A020001 ADD ECX,29A
004011A2 . 69C9 39300001 IMUL ECX,ECX,3039
004011A8 . 83E9 17 SUB ECX,17
004011AB . 83E9 17 SUB ECX,17
004011AE . 33DB XOR EBX,EBX
004011B0 . 8BC1 MOV EAX,ECX
004011B2 . B9 0A000000 MOV ECX,0A
004011B7 > 33D2 XOR EDX,EDX
004011B9 . F7F1 DIV ECX
004011BB . 80C2 30 ADD DL,30
004011BE . 881433 MOV BYTE PTR DS:[EBX+ESI],DL
004011C1 . 83C3 01 ADD EBX,1
004011C4 . 83F8 00 CMP EAX,0
004011C7 . 74 02 JE SHORT Ice9.004011CB
004011C9 . EB EC JMP SHORT Ice9.004011B7

```

 $EBX = 0$ $EAX = ECX$ $ECX = 0x0A$

repeat

 $EDX = 0$ $EDX = EAX \bmod ECX$ $EAX / = ECX$ $DL+ = 0x30$ Guardar DL en el buffer $ESI + EBX$ $EBX ++$ until $EAX = 0$

Ejemplo práctico (III): algoritmo de generación (3)

```

0040119C > 81C1 9A020001 ADD ECX,29A
004011A2 . 69C9 39300001 IMUL ECX,ECX,3039
004011A8 . 83E9 17 SUB ECX,17
004011AB . 33D8 00000000 XOR ECX,ECX
004011AE . 33DB XOR EBX,EBX
004011B0 . 8BC1 MOV EAX,ECX
004011B2 . B9 0A000000 MOV ECX,0A
004011B7 > 33D2 XOR EDX,EDX
004011B9 . F7F1 DIV ECX
004011BB . 80C2 30 ADD DL,30
004011BE . 881433 MOV BYTE PTR DS:[EBX+ESI],DL
004011C1 . 83C3 01 ADD EBX,1
004011C4 . 83F8 00 CMP EAX,0
004011C7 . 74 02 JE SHORT Ice9.004011CB
004011C9 . EB EC JMP SHORT Ice9.004011B7

```

EBX = 0

EAX = *ECX*

ECX = 0x0A

repeat

EDX = 0

EDX = *EAX* mod *ECX*

EAX / = *ECX*

DL + = 0x30

 Guardar *DL* en el buffer *ESI* + *EBX*

EBX + +

until *EAX* = 0

```

004011CB > 5BF C8304000 MOV EDI,Ice9.004030C8
004011D0 > 8A4433 FF MOV AL,BYTE PTR DS:[EBX+ESI-1]
004011D4 . 8807 MOV BYTE PTR DS:[EDI],AL
004011D6 . 47 INC EDI
004011D7 . 4B DEC EBX
004011D8 . 83FB 00 CMP EBX,0
004011DB . 75 F3 JNZ SHORT Ice9.004011D0
004011DD . C607 00 MOV BYTE PTR DS:[EDI],0
004011E0 . 8D3D B4304000 LEA EDI,0WORD PTR DS:[4030B4]
004011E6 . 68 B7304000 PUSH Ice9.004030B7
004011EB . 68 C8304000 PUSH Ice9.004030C8
004011F0 . E8 B0000000 CALL Ice9.004012B0
004011F5 . 68 C8304000 PUSH Ice9.004030C8
004011FA . 68 98314000 PUSH Ice9.00403198
004011FF . E8 98000000 CALL <JMP.&kernel32.1stronpA>

```

Ejemplo práctico (III): algoritmo de generación (3)

```

0040119C > 581C1 9A020001 ADD ECX,29A
004011A2 . 69C9 39300001 IMUL ECX,ECX,3039
004011A8 . 83E9 17 SUB ECX,17
004011AB . 69C9 39300001 IMUL ECX,ECX,3039
004011AE . 330B XOR EBX,EBX
004011B0 . 8BC1 MOV EAX,ECX
004011B2 . B9 0A000000 MOV ECX,0A
004011B7 > 33D2 XOR EDX,EDX
004011B9 . F7F1 DIV ECX
004011BB . 80C2 30 ADD DL,30
004011BE . 881433 MOV BYTE PTR DS:[EBX+ESI],DL
004011C1 . 83C3 01 ADD EBX,1
004011C4 . 83F8 00 CMP EAX,0
004011C7 . 74 02 JE SHORT Ice9.004011CB
004011C9 . EB EC JMP SHORT Ice9.004011B7

```

EBX = 0

EAX = *ECX*

ECX = 0x0A

repeat

EDX = 0

EDX = *EAX* mod *ECX*

EAX / = *ECX*

DL + = 0x30

 Guardar *DL* en el buffer *ESI* + *EBX*

EBX + +

until *EAX* = 0

```

004011CB > 5BF C8304000 MOV EDI,Ice9.004030C8
004011D0 > 8A4433 FF MOV AL,BYTE PTR DS:[EBX+ESI-1]
004011D4 . 8807 MOV BYTE PTR DS:[EDI],AL
004011D6 . 47 INC EDI
004011D7 . 4B DEC EBX
004011D8 . 83FB 00 CMP EBX,0
004011DB . 75 F3 JNZ SHORT Ice9.004011D0

```

EDI = buffer 0x04030C8

Revertimos la cadena apuntada por *ESI* y guardamos en *EDI*

```

004011E0 . 803D B4304000 LEA EDI,DWORD PTR DS:[4030B4]
004011E6 . 68 B7304000 PUSH Ice9.004030B7
004011EB . 68 C8304000 PUSH Ice9.004030C8
004011F0 . E8 B8000000 CALL Ice9.004012B0
004011F5 . 68 C8304000 PUSH Ice9.004030C8
004011FA . 68 98314000 PUSH Ice9.00403198
004011FF . E8 98000000 CALL <JMP.&kernel32.lstrncpy>

```

Ejemplo práctico (III): algoritmo de generación (3)

```

0040119C > 81C1 9A020001 ADD ECX,29A
004011A2 . 69C9 39300001 IMUL ECX,ECX,3039
004011A8 . 83E9 17 SUB ECX,17
004011AB . 33D8 XOR ECX,ECX
004011AE . 33DB XOR EBX,EBX
004011B0 . 8BC1 MOV EAX,ECX
004011B2 . B9 0A000000 MOV ECX,0A
004011B7 > 33D2 XOR EDX,EDX
004011B9 . F7F1 DIV ECX
004011BB . 80C2 30 ADD DL,30
004011BE . 881433 MOV BYTE PTR DS:[EBX+ESI],DL
004011C1 . 83C3 01 ADD EBX,1
004011C4 . 83F8 00 CMP EAX,0
004011C7 . 74 02 JE SHORT Ice9.004011CB
004011C9 . EB EC JMP SHORT Ice9.004011B7

```

$EBX = 0$

$EAX = ECX$

$ECX = 0x0A$

repeat

$EDX = 0$

$EDX = EAX \bmod ECX$

$EAX / = ECX$

$DL + = 0x30$

Guardar DL en el buffer $ESI + EBX$

$EBX + +$

until $EAX = 0$

```

004011CB > 5BF C8304000 MOV EDI,Ice9.004030C8
004011D0 > 8A4433 FF MOV AL,BYTE PTR DS:[EBX+ESI-1]
004011D4 . 8807 MOV BYTE PTR DS:[EDI],AL
004011D6 . 47 INC EDI
004011D7 . 4B DEC EBX
004011D8 . 83FB 00 CMP EBX,0
004011DB . 75 F3 JNZ SHORT Ice9.004011D0
004011DD . C607 00 MOV BYTE PTR DS:[EDI],0
004011E0 .
004011E6 . 68 B7304000 PUSH Ice9.004030B7
004011EB . 68 C8304000 PUSH Ice9.004030C8
004011F0 . E8 BB000000 CALL Ice9.004012B0
004011F5 .
004011FA . 68 98314000 PUSH Ice9.00403198
004011FF . E8 98000000 CALL <JMP.&kernel32.1stronpA>

```

$EDI = \text{buffer } 0x04030C8$

Revertimos la cadena apuntada por ESI y guardamos en EDI
 Añadimos resto del nombre (desde el 4º carácter) al final de la
 nueva cadena

Ejemplo práctico (III): algoritmo de generación (3)

```

0040119C > 581C1 9A020001 ADD ECX,29A
004011A2 . 69C9 39300001 IMUL ECX,ECX,3039
004011A8 . 83E9 17 SUB ECX,17
004011AB . 330B XOR EBX,EBX
004011AE . 8BC1 MOV EAX,ECX
004011B0 . B9 0A000000 MOV ECX,0A
004011B7 > 33D2 XOR EDX,EDX
004011B9 . F7F1 DIV ECX
004011BB . 80C2 30 ADD DL,30
004011BE . 881433 MOV BYTE PTR DS:[EBX+ESI],DL
004011C1 . 83C3 01 ADD EBX,1
004011C4 . 83F8 00 CMP EAX,0
004011C7 . 74 02 JE SHORT Ice9.004011CB
004011C9 . EB EC JMP SHORT Ice9.004011B7

```

$EBX = 0$

$EAX = ECX$

$ECX = 0x0A$

repeat

$EDX = 0$

$EDX = EAX \bmod ECX$

$EAX / = ECX$

$DL + = 0x30$

 Guardar DL en el buffer $ESI + EBX$

$EBX + +$

until $EAX = 0$

```

004011CB > 5BF C8304000 MOV EDI,Ice9.004030C8
004011D0 > 8A4433 FF MOV AL,BYTE PTR DS:[EBX+ESI-1]
004011D4 . 8807 MOV BYTE PTR DS:[EDI],AL
004011D6 . 47 INC EDI
004011D7 . 4B DEC EBX
004011D8 . 83FB 00 CMP EBX,0
004011DB . 75 F3 JNZ SHORT Ice9.004011D0
004011DD . C607 00 MOV BYTE PTR DS:[EDI],0
004011E0 . 6D3D B4304000 LEA EDI,DMWORD PTR DS:[4030B4]
004011E6 . 68 B7304000 PUSH Ice9.004030B7
004011EB . 68 C8304000 PUSH Ice9.004030C8
004011F0 . 68 C8304000 PUSH Ice9.004030C8
004011F5 . 68 98314000 PUSH Ice9.00403198
004011FA . EB 98000000 CALL <JMP.&kernel32.1strcmpA>

```

$EDI = \text{buffer } 0x04030C8$

Revertimos la cadena apuntada por ESI y guardamos en EDI

Añadimos resto del nombre (desde el 5º carácter) al final de la nueva cadena

Y ya se compara la cadena construida con la introducida

- 1 Introducción a la Ingeniería Inversa
 - Qué es la Ingeniería Inversa
 - Motivación
 - Aproximaciones a la Ingeniería Inversa
- 2 Conocimientos previos
- 3 Refrescando la memoria...
 - Conocimientos de manejo de debuggers
 - Conocimientos de arquitectura x86 básica
 - Conocimientos de las estructuras PE
- 4 Herramientas útiles
- 5 Técnicas de análisis
 - Código muerto
 - Código 'vivo'
- 6 Técnicas de *reversing*
 - CD Check
 - *Patching* y *loaders*
 - *Time-trials* y Registro de Windows
 - Captura del *serial* y *Keygenning*
 - Archivos de licencia
 - Desempacado (*unpacking*)
- 7 Algunos métodos *antireversing*
- 8 Ejemplo práctico
 - Estudio del crackME
 - Algoritmo de generación
- 9 **Conclusiones**
- 10 Investigación en Ingeniería Inversa

Conclusiones y otras aplicaciones

- **Cualquier protección es *crackeable***
- Mundo de constante evolución → nuevas protecciones, nuevos métodos
- **Leer y practicar mucho**
- Usar y programar más *software* libre
 - Que no 'hacer' más *software* 'libre' :)

Otras aplicaciones

- **Fuzzing**
- **Exploiting**
- **DBI** (Dynamic Binary Instrumentation)

- 1 Introducción a la Ingeniería Inversa
 - Qué es la Ingeniería Inversa
 - Motivación
 - Aproximaciones a la Ingeniería Inversa
- 2 Conocimientos previos
- 3 Refrescando la memoria...
 - Conocimientos de manejo de debuggers
 - Conocimientos de arquitectura x86 básica
 - Conocimientos de las estructuras PE
- 4 Herramientas útiles
- 5 Técnicas de análisis
 - Código muerto
 - Código 'vivo'
- 6 Técnicas de *reversing*
 - CD Check
 - *Patching* y *loaders*
 - *Time-trials* y Registro de Windows
 - Captura del *serial* y *Keygenning*
 - Archivos de licencia
 - Desempacado (*unpacking*)
- 7 Algunos métodos *antireversing*
- 8 Ejemplo práctico
 - Estudio del crackME
 - Algoritmo de generación
- 9 Conclusiones
- 10 Investigación en Ingeniería Inversa

Investigación en Ingeniería Inversa

- Comparativa de **rendimiento con/sin protecciones**
- **Nuevas técnicas de protección**
 - Basadas en Redes de Petri
 - ?
- Búsqueda de **nuevas vulnerabilidades**
- **Conferencias académicas y revistas**
 - Working Conference on Reverse Engineering (WCRE), CORE B (2010)
 - IEEE Security & Privacy, Q3 (2010)

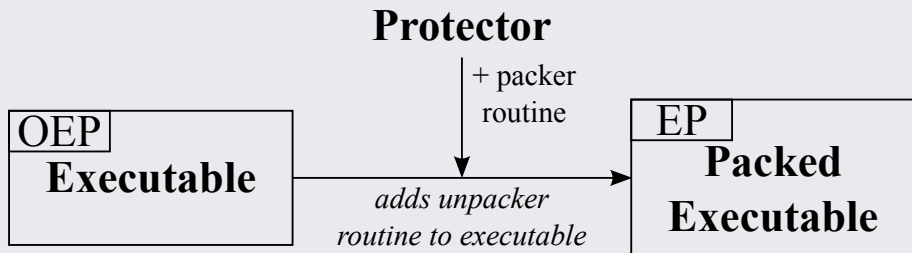
Frameworks DBI (I)

- **Dynamic Binary Instrumentation**
- **Análisis del comportamiento de un ejecutable en tiempo de ejecución**
- **Añade código de instrumentación:**
 - 1 ¿qué se va a ejecutar?
 - 2 ¿hago algo?
- **Diferentes frameworks DBI**
 - Pin (de Intel)
 - Valgrind
 - DynamoRIO

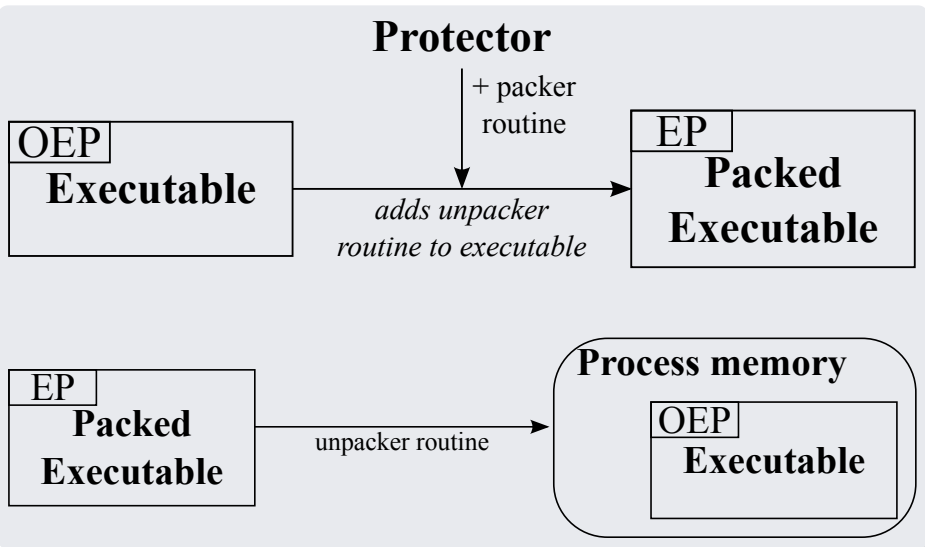
Frameworks DBI (II)

- PFC: Estudio comparativo de frameworks de Instrumentación Dinámica de Ejecutables
- Alguna cosa curiosa:
 - No. instrucciones ejecutadas varía entre frameworks
- Charla aceptada en **RootedCON 2012** (y próxima publicación :))
 - **Mejora del proceso de desempacado usando DBI**
 - Paso imprescindible en desempacado: encontrar **Original Entry Point (OEP)**

Frameworks DBI (III)



Frameworks DBI (III)



Introducción a la Ingeniería Inversa

Diseño de Aplicaciones Seguras

Ricardo J. Rodríguez

rjrodriguez@unizar.es



Universidad
Zaragoza

19 de Enero de 2012

Grupo de Ingeniería de Sistemas de Eventos Discretos
Universidad de Zaragoza