

El Arte de la Ingeniería Inversa

Ricardo J. Rodríguez



#eCh!2004 - .:[CrackSLatinoS]:
www.ech2004.net

23 de Octubre de 2010

HackMeeting 2010
Zaragoza, Spain

Outline

- 1 **Introducción a la Ingeniería Inversa**
 - Qué es la Ingeniería Inversa
 - Motivación
 - La Ingeniería Inversa de código
- 2 **Conocimientos previos**
- 3 **Cinturón de herramientas**
 - Desensambladores y editores hexadecimales
 - Debuggers
 - Identificadores, editores PE y de recursos
 - Dumpeadores de memoria y emuladores
 - Monitores de APIs y reparadores de IAT
 - Documentación
- 4 **Técnicas de análisis**
 - Código muerto
 - Código 'vivo'
- 5 **Algunos métodos *anticracking***
 - Algunas APIs *antidebugging*
 - Métodos más avanzados
- 6 **Técnicas de *cracking***
 - CD Check
 - *Patching* y *loaders*
 - *Time-trials* y Registro de Windows
 - Captura del *serial* y *Keygenning*
 - Archivos de licencia
 - Desempacado (*unpacking*)
- 7 **Ejemplo práctico**
 - Estudio del crackME
 - Algoritmo de generación
- 8 **Conclusiones y agradecimientos**

- 1 **Introducción a la Ingeniería Inversa**
 - Qué es la Ingeniería Inversa
 - Motivación
 - La Ingeniería Inversa de código
- 2 **Conocimientos previos**
- 3 **Cinturón de herramientas**
 - Desensambladores y editores hexadecimales
 - Debuggers
 - Identificadores, editores PE y de recursos
 - Dumpeadores de memoria y emuladores
 - Monitores de APIs y reparadores de IAT
 - Documentación
- 4 **Técnicas de análisis**
 - Código muerto
 - Código 'vivo'
- 5 **Algunos métodos *anticracking***
 - Algunas APIs *antidebugging*
 - Métodos más avanzados
- 6 **Técnicas de *cracking***
 - CD Check
 - *Patching* y *loaders*
 - *Time-trials* y Registro de Windows
 - Captura del *serial* y *Keygenning*
 - Archivos de licencia
 - Desempacado (*unpacking*)
- 7 **Ejemplo práctico**
 - Estudio del crackME
 - Algoritmo de generación
- 8 **Conclusiones y agradecimientos**

Introducción a la Ingeniería Inversa (I)

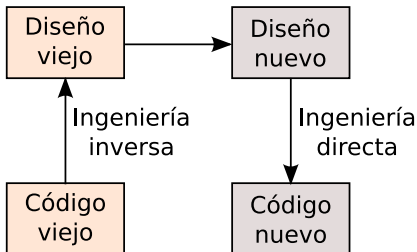
Ingeniería inversa (*reverse engineering*)

- 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 (III)

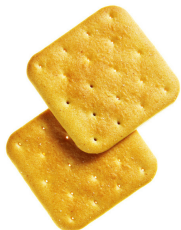
Reverse engineering code

- También conocida como *cracking*
- Eliminar protecciones de código (*copyrights*)
- NO siempre es malo: detección de *bugs*, potenciales *exploits*, ... en tus programas

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*, ... en tus programas
- **Crackers**: algo más que unas galletas...
 - **NO CONFUNDIR** con los *criminal hackers*



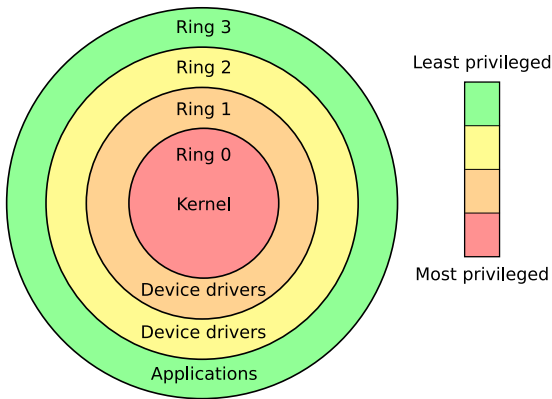
- 1 **Introducción a la Ingeniería Inversa**
 - Qué es la Ingeniería Inversa
 - Motivación
 - La Ingeniería Inversa de código
- 2 **Conocimientos previos**
- 3 **Cinturón de herramientas**
 - Desensambladores y editores hexadecimales
 - Debuggers
 - Identificadores, editores PE y de recursos
 - Dumpeadores de memoria y emuladores
 - Monitores de APIs y reparadores de IAT
 - Documentación
- 4 **Técnicas de análisis**
 - Código muerto
 - Código 'vivo'
- 5 **Algunos métodos *anticracking***
 - Algunas APIs *antidebugging*
 - Métodos más avanzados
- 6 **Técnicas de *cracking***
 - CD Check
 - *Patching* y *loaders*
 - *Time-trials* y Registro de Windows
 - Captura del *serial* y *Keygenning*
 - Archivos de licencia
 - Desempacado (*unpacking*)
- 7 **Ejemplo práctico**
 - Estudio del crackME
 - Algoritmo de generación
- 8 **Conclusiones y agradecimientos**

Conocimientos previos (I)

- **Ensamblador**
 - Básico e imprescindible
 - <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>
 - <http://ech2004.net/download.php?archivo=198&tipo=t>
- **Estructura interna de un PE (*Portable Executable*)**
 - Cada lenguaje de programación produce una cabecera diferente...
 - ... pero siempre las mismas primeras instrucciones
 - <http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.msp>
- **Debuggear programas**
 - *Breakpoints*
 - *Step into, step over, ...*

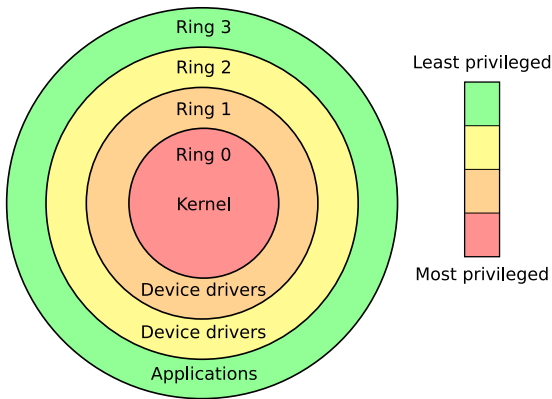
Conocimientos previos (II)

- 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)



“If a program runs, then it’s crackeable”

Conocimientos previos (III)

Tipos de aplicaciones

- *Freeware*: *software* de libre distribución
- *Shareware*: *software* con licencia de distribución
 - *Demoware*: algunas *features* del *software* inhabilitadas
 - *Trialware*: *software* con limitación temporal de uso
- *Adware*: publicidad embebida
- *Spyware*: malo malo...

- 1 **Introducción a la Ingeniería Inversa**
 - Qué es la Ingeniería Inversa
 - Motivación
 - La Ingeniería Inversa de código
- 2 **Conocimientos previos**
- 3 **Cinturón de herramientas**
 - Desensambladores y editores hexadecimales
 - Debuggers
 - Identificadores, editores PE y de recursos
 - Dumpeadores de memoria y emuladores
 - Monitores de APIs y reparadores de IAT
 - Documentación
- 4 **Técnicas de análisis**
 - Código muerto
 - Código 'vivo'
- 5 **Algunos métodos *anticracking***
 - Algunas APIs *antidebugging*
 - Métodos más avanzados
- 6 **Técnicas de *cracking***
 - CD Check
 - *Patching* y *loaders*
 - *Time-trials* y Registro de Windows
 - Captura del *serial* y *Keygenning*
 - Archivos de licencia
 - Desempacado (*unpacking*)
- 7 **Ejemplo práctico**
 - Estudio del crackME
 - Algoritmo de generación
- 8 **Conclusiones y agradecimientos**

Cinturón de herramientas (I)

Desensambladores

- W32Dasm
(<http://ech2004.net/download.php?archivo=52&tipo=h>)
- **HDasm** (<http://hdasm.narod.ru/>)
- **IDA Pro** (<http://www.hex-rays.com/idapro/>)
- Específicos
 - p32Dasm (VBasic)
 - Reflector (.NET)
 - DJ Java Decompiler (Java)
 - DeDe (Delphi)
 - DeRefox (VisualFox)
 - ...

Editor hexadecimal

- **HexWorkshop** (<http://www.bpsoft.com/>)

Cinturón de herramientas (II)

Debuggers

- Soft ICE
 - Trabaja(ba) en R0
 - Hay parches para trabajar sobre WinXP
- **OlllyDBG** (<http://www.ollydbg.de/>)
 - Trabaja en R3
 - Muchas variantes con muchos *plugins*
- EDB (<http://freshmeat.net/projects/edebugger>)
- TitanEngine
(<http://www.reversinglabs.com/products/TitanEngine.php>)
- ...

Cinturón de herramientas (III)

(www.ech2004.net, sección 'Herramientas Imprescindibles')

Identificadores y editores PE

- PEiD
- PEEditor
- Sabueso
- RDG Packer Detector
- ...

Visores de recursos

- eXe Scope
- XNResource Editor
- Resorce Hacker
- Restorator

Cinturón de herramientas (IV)

(www.ech2004.net, sección 'Herramientas Imprescindibles')

Dumpeadores de memoria

- LordPE Deluxe
- SirPE (para 32 y 64 bits)
- Pupe
- ProcDump
- ...

Emuladores

- HASP
- Sentinel
- Rock4

Cinturón de herramientas (V)

(www.ech2004.net, sección 'Herramientas Imprescindibles')

Monitores de APIs

- KaKeeware Application Monitor
- Event2Address (E2A)
- API Scan
- WinAPIOverride32
- ...

Reparadores de IAT

- Import REC (el abuelo Simpson :))
- ReVirgin
- Universal Import Fixer

Cinturón de herramientas (VI)

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/>)

Cinturón de herramientas (VI)

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.de/>)

1 Introducción a la Ingeniería Inversa

- Qué es la Ingeniería Inversa
- Motivación
- La Ingeniería Inversa de código

2 Conocimientos previos

3 Cinturón de herramientas

- Desensambladores y editores hexadecimales
- Debuggers
- Identificadores, editores PE y de recursos
- Dumpeadores de memoria y emuladores
- Monitores de APIs y reparadores de IAT
- Documentación

4 Técnicas de análisis

- Código muerto
- Código 'vivo'

5 Algunos métodos *anticracking*

- Algunas APIs *antidebugging*
- Métodos más avanzados

6 Técnicas de *cracking*

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

7 Ejemplo práctico

- Estudio del crackME
- Algoritmo de generación

8 Conclusiones y agradecimientos

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 69488A5300      push 00538A48
:004984B4 E8CCFBFFFF      call 00498085
:004984B9 85C0            test eax, eax
:004984BB 0F9499000000    je 0049855A
:004984C1 BE887C5200      mov esi, 00527C88
:004984C6 BF488A5300      mov edi, 00538A48
:004984CB 33C0            xor eax, eax
:004984CD 83C9FF          or ecx, FFFFFFFF
:004984D0 F2              repnz
```

Pasos

- 1 Identificar PE (¿está protegido?)
- 2 Desensamblar
- 3 Buscar mensajes de chico malo
- 4 Analizar camino hasta el mensaje
- 5 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 ¡!

```

00424915: PUSH registro.00422796 |UNIQUE| "english.dll"
00425000: MOV DWORD PTR SS:[EBP-90],registro.0042 |UNIQUE| "trial version expired"
00425008: MOV DWORD PTR SS:[EBP-90],registro.0042 |UNIQUE| "This trial version has exp"
00425016: MOV DWORD PTR SS:[EBP-90],registro.0042 |UNIQUE| "Este producto ha caducado!"
0042511E: MOV DWORD PTR SS:[EBP-90],registro.0042 |UNIQUE| "Este producto ha caducado."
00425187: PUSH registro.00422794 |UNIQUE| "alt"
00425190: MOV DWORD PTR SS:[EBP-90],registro.0042 |UNIQUE| "Error!"
00425420: PUSH registro.00422E49 |UNIQUE| "utilities.77 backdoor"
00425454: PUSH registro.00422E14 |UNIQUE| "http://www.mega-9999"
00425508: PUSH registro.00422796 |UNIQUE| "english.dll"
00425525: MOV DWORD PTR SS:[EBP-90],registro.0042 |UNIQUE| "Registration"
0042552C: PUSH registro.00422E79 |UNIQUE| "there is a problem when t"
00425531: PUSH registro.00422E67 |UNIQUE| "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...)

1 Introducción a la Ingeniería Inversa

- Qué es la Ingeniería Inversa
- Motivación
- La Ingeniería Inversa de código

2 Conocimientos previos

3 Cinturón de herramientas

- Desensambladores y editores hexadecimales
- Debuggers
- Identificadores, editores PE y de recursos
- Dumpeadores de memoria y emuladores
- Monitores de APIs y reparadores de IAT
- Documentación

4 Técnicas de análisis

- Código muerto
- Código 'vivo'

5 Algunos métodos anticracking

- Algunas APIs antidebugging
- Métodos más avanzados

6 Técnicas de cracking

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

7 Ejemplo práctico

- Estudio del crackME
- Algoritmo de generación

8 Conclusiones y agradecimientos

Algunas APIs antidebugging

- IsDebuggerPresent
- NtGlobalFlag
- ProcessHeapFlag
- FindWindowA
- SetUnhandledExceptionFilter/UnhandledExceptionFilter
- CreateToolhelp32Snapshot
- ZwQueryInformationProces
- CloseHandle
- CsrGetProcessId

Lectura recomendada: <http://pferrie.tripod.com/>

Métodos más avanzados (I)

Packers

- Reduce tamaño y/o encripta (*scrambling*) el código de los ejecutables
- Modifica secciones del PE (y más cosas...)
- Numerosos *packers* en el mercado: UPX, ASPack, ASProtect, ...
 - http://en.wikipedia.org/wiki/Executable_compression
- Algunos fáciles de crackear, otros no tanto :)
- Pasos a realizar
 - 1 Hallar el OEP (*Original Entry Point*)
 - 2 Dumpear el proceso de memoria (estará desempacado!)
 - 3 Arreglar la IAT (*Import Address Table*)

VM Packers (más complicados, pero no imposibles)

- Misma idea que *packers* PERO una VM ejecuta el código
- Ejemplos: Themida, EXECryptor, CodeVirtualizer

Métodos más avanzados (II)

Métodos complementarios de protección

- Redireccionamiento de la IAT
- Secciones virtuales (*antidump*)
- Emulación de API
- Cambios en la estructura del ejecutable (*PE header*)
- Ofuscamiento de código (código basura)
- Bytes perdidos (*stolen bytes*)
- Detección de *debuggers*, otras *tools* o *breakpoints* (BP y HBP)
- Nanomites (usado por Armadillo)

Detección de modificaciones

- CRC
- MD5
- GetFileTimeA, CreateFileA
- Monitoreo de zona de memoria

1 Introducción a la Ingeniería Inversa

- Qué es la Ingeniería Inversa
- Motivación
- La Ingeniería Inversa de código

2 Conocimientos previos

3 Cinturón de herramientas

- Desensambladores y editores hexadecimales
- Debuggers
- Identificadores, editores PE y de recursos
- Dumpeadores de memoria y emuladores
- Monitores de APIs y reparadores de IAT
- Documentación

4 Técnicas de análisis

- Código muerto
- Código 'vivo'

5 Algunos métodos *anticracking*

- Algunas APIs *antidebugging*
- Métodos más avanzados

6 Técnicas de *cracking*

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

7 Ejemplo práctico

- Estudio del crackME
- Algoritmo de generación

8 Conclusiones y agradecimientos

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 8B98 06  LEA  EAX,6
0041F160 > 74 00  JZ  0041F162
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  LEA  EDI,DMORD PTR SS:[ESP+114]
0041F160 5F 00010000  PUSH 100
0041F160 8D4C24 10  LEA  ECX,DMORD PTR SS:[ESP+10]
0041F160 5D      PUSH EAX
0041F160 8D424 18  LEA  EDI,DMORD PTR SS:[ESP+18]
0041F160 51      PUSH ECX
0041F160 8D424 14  LEA  EDI,DMORD PTR SS:[ESP+14]
0041F160 52      PUSH EDI
0041F160 5D      PUSH EAX
0041F160 8D4C24 20  LEA  ECX,DMORD PTR SS:[ESP+20]
0041F160 51      PUSH 100
0041F160 68 00010000  PUSH 0
0041F160 5A      PUSH ESI
0041F160 FF15 94C16100  JNZ  0041F163
0041F160 5C80    XOR  ESI,ESI
0041F160 > 75 00  JNZ  0041F162
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  LEA  ECX,DMORD PTR SS:[ESP+14]
0041F160 81C2 F0320000  ADD  EDI,5FD
0041F160 5D      PUSH EDI
0041F160 5D      PUSH EDI
0041F160 5D      PUSH EDI
0041F160 8B 74021E00  MOV  EDI,74021E00
0041F160 83C4 00  ADD  ESP,0
0041F160 F700    MUL  EDI
0041F160 5B80    SBB  EDI,EBX
0041F160 5E      POP  ESI
0041F160 40      INC  EAX
0041F160 5B      POP  EBX
0041F160 81C4 0C020000  ADD  ESP,20C
0041F160 C2 0400  RETN

```

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)
 - Simple
 - *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
 - La Ingeniería Inversa de código
- 2 **Conocimientos previos**
- 3 **Cinturón de herramientas**
 - Desensambladores y editores hexadecimales
 - Debuggers
 - Identificadores, editores PE y de recursos
 - Dumpeadores de memoria y emuladores
 - Monitores de APIs y reparadores de IAT
 - Documentación
- 4 **Técnicas de análisis**
 - Código muerto
 - Código 'vivo'
- 5 **Algunos métodos *anticracking***
 - Algunas APIs *antidebugging*
 - Métodos más avanzados
- 6 **Técnicas de *cracking***
 - CD Check
 - *Patching* y *loaders*
 - *Time-trials* y Registro de Windows
 - Captura del *serial* y *Keygenning*
 - Archivos de licencia
 - Desempacado (*unpacking*)
- 7 **Ejemplo práctico**
 - Estudio del crackME
 - Algoritmo de generación
- 8 **Conclusiones y agradecimientos**

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
00401002 E8 89020000 CALL <JMP.&kernel32.GetModuleHandleA>
00401007 68 94314000 MOV DWORD PTR DS:[403194],EAX
0040100C 6A 00 PUSH 0
0040100E 68 29104000 PUSH Ice9.00401029
00401013 6A 00 PUSH 0
00401015 6A 65 PUSH 65
00401017 FF35 94314000 PUSH DWORD PTR DS:[403194]
0040101D E8 3E020000 CALL <JMP.&user32.MessageBoxParamA>
00401022 6A 00 PUSH 0
00401024 E8 61020000 CALL <JMP.&kernel32.ExitProcess>
00401029 55 PUSH EBP
0040102A 8BEC MOV EBP,ESP
0040102C 8B45 0C MOV EAX,[ARG_2]
0040102F 83F9 10 CMP EAX,10
00401032 75 0F JNZ SHORT Ice9.00401043
00401034 6A 00 PUSH 0
00401036 FF75 08 PUSH [ARG_1]
00401039 E8 28020000 CALL <JMP.&user32.EndDialog>
0040103E E9 07010000 JMP Ice9.0040114A
00401043 > 3D 01020000 CMP 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
00401050 68 01000000 PUSH 0A1
00401055 FF75 08 PUSH [ARG_1]
00401058 E8 21020000 CALL <JMP.&user32.PostMessageA>
0040105D E9 E8000000 JMP Ice9.0040114A
00401062 > 3D 10010000 CMP EAX,110
00401067 75 38 JNZ SHORT Ice9.004010A1
00401069 E8 28020000 CALL <JMP.&kernel32.IsDebuggerPresent>
0040106E 83F9 01 CMP EAX,1
00401071 75 0A JNZ SHORT Ice9.0040107D
00401073 6A 00 PUSH 0
00401075 FF75 08 PUSH [ARG_1]
00401078 E8 E9010000 CALL <JMP.&user32.EndDialog>
0040107D > 33C9 XOR EAX,EAX

```

```

pModule = NULL
GetModuleHandleA

IParam = NULL
DlgProc = Ice9.00401029
hOwner = NULL
pTemplate = 65
hInst = NULL
DialogBoxParamA
ExitCode = 0
ExitProcess

Result = 0
hWnd
EndDialog

IParam
wParam = 2
Message = WM_NCLBUTTONDOWN
hWnd
PostMessageA

IsDebuggerPresent

Result = 0
hWnd
EndDialog

```

Pasos

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

```

00401000 6A 00 PUSH 0
00401002 E8 89020000 CALL <JMP.&kernel32.GetModuleHandleA>
00401007 R3 94314000 MOV DWORD PTR DS:[403194],EAX
0040100C 6A 00 PUSH 0
0040100E 68 29104000 PUSH Ice9.00401029
00401013 6A 00 PUSH 0
00401015 6A 65 PUSH 65
00401017 FF35 94314000 PUSH DWORD PTR DS:[403194]
0040101D E8 3E020000 CALL <JMP.&user32.DialogBoxParamA>
00401022 6A 00 PUSH 0
00401024 E8 61020000 CALL <JMP.&kernel32.ExitProcess>
00401029 55 PUSH EBP
0040102A 8BEC MOV EBX,ESP
0040102C 8B45 0C MOV EAX,[ARG_2]
0040102F 83F9 10 CMP EAX,10
00401032 75 0F JNZ SHORT Ice9.00401043
00401034 6A 00 PUSH 0
00401036 FF75 08 PUSH [ARG_1]
00401039 E8 28020000 CALL <JMP.&user32.EndDialog>
0040103E E9 07010000 JMP Ice9.0040114A
00401043 > 3D 01020000 CMP 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
00401050 68 71000000 PUSH 0A1
00401055 FF75 08 PUSH [ARG_1]
00401058 E8 21020000 CALL <JMP.&user32.PostMessageA>
0040105D E9 E8000000 JMP Ice9.0040114A
00401062 > 3D 10010000 CMP EAX,110
00401067 75 38 JNZ SHORT Ice9.004010A1
00401069 E8 28020000 CALL <JMP.&kernel32.IsDebuggerPresent>
0040106E 83F9 01 CMP EAX,1
00401071 75 0A JNZ SHORT Ice9.0040107D
00401073 6A 00 PUSH 0
00401075 FF75 08 PUSH [ARG_1]
00401078 E8 E9010000 CALL <JMP.&user32.EndDialog>
0040107D > 33C0 XOR EAX,EAX

```

Pasos

- 1 Llamada a IsDebuggerPresent → ocultar el Olly

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

```

00401000 6A 00 PUSH 0
00401002 E8 89020000 CALL <JMP.&kernel32.GetModuleHandleA>
00401007 A3 94314000 MOV DWORD PTR DS:[403194],EAX
0040100C 6A 00 PUSH 0
0040100E 68 29104000 PUSH Ice9.00401029
00401013 6A 00 PUSH 0
00401015 6A 65 PUSH 65
00401017 FF35 94314000 PUSH DWORD PTR DS:[403194]
0040101D E8 3E020000 CALL <JMP.&user32.DialogBoxParamA>
00401022 6A 00 PUSH 0
00401024 E8 61020000 CALL <JMP.&kernel32.ExitProcess>
00401029 55 PUSH EBP
0040102A 8BEC MOV EBP,ESP
0040102C 8B45 0C MOV EAX,[ARG_23]
0040102F 83F9 10 CMP EAX,10
00401032 75 0F JNZ SHORT Ice9.00401043
00401034 6A 00 PUSH 0
00401036 FF75 08 PUSH [ARG_13]
00401039 E8 28020000 CALL <JMP.&user32.EndDialog>
0040103E E9 07010000 JMP Ice9.0040114A
00401043 > 3D 01020000 CMP EAX,201
00401048 75 18 JNZ SHORT Ice9.00401062
0040104A 8B45 14 MOV EAX,[ARG_41]
0040104D 50 PUSH EAX
0040104E 6A 02 PUSH 2
00401050 68 71000000 PUSH 0A1
00401055 FF75 08 PUSH [ARG_13]
00401058 E8 21020000 CALL <JMP.&user32.PostMessageA>
0040105D E9 E8000000 JMP Ice9.0040114A
00401062 > 3D 10010000 CMP EAX,110
00401067 75 38 JNZ SHORT Ice9.004010A1
00401069 E8 28020000 CALL <JMP.&kernel32.IsDebuggerPresent>
0040106E 83F9 01 CMP EAX,1
00401071 75 0A JNZ SHORT Ice9.0040107D
00401073 6A 00 PUSH 0
00401075 FF75 08 PUSH [ARG_13]
00401078 E8 E9010000 CALL <JMP.&user32.EndDialog>
0040107D > 33C9 XOR 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 04304000 | PUSH Ice9.00403004 | Buffer = Ice9.00403004 |
| 0040100A | 68 E9030000 | PUSH 3E9 | ControlID = 3E9 (1001.) |
| 0040100F | FF75 00 | PUSH [ARG_1] | hWnd |
| 00401012 | E9 0C010000 | CALL <JMP.&user32.GetDlgItemTextA> | GetDlgItemTextA |
| 00401017 | 0BC0 | OR EAX, EAX | |
| 00401019 | > 74 0A | JE SHORT Ice9.004010F5 | |
| 0040101B | > 83F8 04 | CMPL EAX, 4 | |
| 0040101E | > 74 05 | JE SHORT Ice9.004010F5 | |
| 00401020 | > 83F8 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 | > 68 00020000 | PUSH 200 | hOwner = NULL |
| 00401038 | > E8 70010000 | CALL <JMP.&user32.MessageBoxA> | MessageBoxA |
| 0040103D | > EB 35 | JMP SHORT Ice9.0040113F | |
| 00401041 | > 68 00020000 | PUSH 200 | Count = 200 (512.) |
| 00401046 | > 68 98314000 | PUSH Ice9.00403198 | Buffer = Ice9.00403198 |
| 0040104B | > 68 EA030000 | PUSH 3EA | ControlID = 3EA (1002.) |
| 00401050 | > FF75 00 | PUSH [ARG_1] | hWnd |
| 00401055 | E9 4B010000 | CALL <JMP.&user32.GetDlgItemTextA> | GetDlgItemTextA |
| 0040105A | > 0BC0 | OR EAX, EAX | |
| 0040105C | > 75 15 | JBE SHORT Ice9.0040113A | |
| 0040105E | > 6A 00 | PUSH 0 | Style = MB_OK MB_APPLMODAL |
| 00401061 | > 68 04304000 | PUSH Ice9.00403004 | Title = "Error, Bad Boy" |
| 00401066 | > 68 68304000 | PUSH Ice9.00403060 | Text = "Where is the serial, Mr Cracker ? " |
| 0040106B | > 6A 00 | PUSH 0 | hOwner = NULL |
| 0040106D | > E8 40010000 | CALL <JMP.&user32.MessageBoxA> | MessageBoxA |
| 00401072 | > EB 35 | JMP SHORT Ice9.0040113F | |
| 00401077 | > E8 14000000 | CALL Ice9.00401153 | |
| 0040107C | > EB 09 | JMP SHORT Ice9.0040114A | |
| 0040107E | > B8 00000000 | MOV EAX, 0 | |
| 00401081 | > C9 | LEAVE | |
| 00401083 | > C2 1000 | RETN 10 | |
| 00401085 | > B8 01000000 | MOV EAX, 1 | |
| 00401088 | > C9 | LEAVE | |
| 0040108A | > 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 B4304000 | PUSH Ice9.004030B4 | Buffer = Ice9.004030B4 |
| 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 | 83F8 04 | CMR EAX, 4 | |
| 0040101E | 72 05 | JBE SHORT Ice9.004010F5 | |
| 0040101F | 83F8 0A | CMR EAX, 0A | |
| 00401019 | 76 15 | JBE SHORT Ice9.0040110A | |
| 004010F5 | 6A 00 | PUSH 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 |
| 00401098 | EB 95 | JMP SHORT Ice9.0040113F | MessageBoxA |
| 0040109A | 68 00220000 | 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 | CALL <JMP.&user32.GetDlgItemTextA> | hMnd |
| 00401121 | 0BC0 | OR EAX, EAX | GetDlgItemTextA |
| 00401123 | 75 15 | JNZ SHORT Ice9.0040113A | |
| 00401125 | 6A 00 | PUSH 0 | |
| 00401127 | 68 04304000 | PUSH Ice9.004030B4 | Style = MB_OK MB_APPLMODAL |
| 0040112C | 68 66304000 | PUSH Ice9.00403060 | Title = "Error, Bad Boy" |
| 00401131 | 6A 00 | PUSH 0 | Text = "Where is the serial, Mr Cracker ?" |
| 00401133 | E8 40010000 | CALL <JMP.&user32.MessageBoxA> | hOwner = NULL |
| 00401138 | EB 95 | JMP SHORT Ice9.0040113F | MessageBoxA |
| 0040113A | E8 14000000 | CALL Ice9.00401153 | |
| 0040113F | EB 09 | JMP SHORT Ice9.0040114A | |
| 00401141 | B8 00000000 | MOV EAX, 0 | |
| 00401146 | C9 | LEAVE | |
| 00401147 | C2 1000 | RETN 10 | |
| 0040114A | B8 01000000 | MOV EAX, 1 | |
| 0040114F | C9 | LEAVE | |
| 00401150 | 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] | hMnd |
| 00401012 | E9 95010000 | JMP SHORT Ice9.00401012 | 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 | |
| 00401019 | 76 15 | JBE SHORT Ice9.0040110A | |
| 004010F5 | 6A 00 | OR EBX,EBX | |
| 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 | JMP SHORT Ice9.0040113F | hOwner = NULL |
| 00401088 | EB 3E | CALL <JMP.&user32.MessageBoxA> | 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 SHORT Ice9.0040113A | hMnd |
| 00401121 | 0BC0 | 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 66304000 | 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 95 | JMP SHORT Ice9.0040113F | |
| 0040113A | CALL 14000000 | CALL Ice9.00401153 | |
| 0040113F | EB 09 | JMP SHORT Ice9.0040114A | |
| 00401141 | B8 00000000 | MOV EAX,0 | |
| 00401146 | C9 | LEAVE | |
| 00401147 | C2 1000 | RETN 10 | |
| 0040114A | B8 01000000 | MOV 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 05010000 | CALL <JMP.>user32.GetDlgItemTextA | GetDlgItemTextA |
| 00401017 | 0BC0 | OR EAX,EAX | |
| 00401019 | < 74 0A | JE SHORT Ice9.004010F5 | |
| 0040101B | < 83F8 04 | CMPL EAX,4 | |
| 0040101E | < 72 05 | JBE SHORT Ice9.004010F5 | |
| 0040101F | < 83F8 0A | CMPL EAX,6 | |
| 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 |
| 00401088 | EB 95 | OR EAX,EAX | MessageBoxA |
| 0040109A | < 68 00220000 | PUSH 200 | |
| 0040109F | < 68 9E314000 | PUSH Ice9.00403198 | Count = 200 (512.) |
| 004010A4 | < 68 EA030000 | PUSH 3EA | Buffer = Ice9.00403198 |
| 004010A9 | < FF75 08 | PUSH [ARG.1] | ControlID = 3EA (1002.) |
| 004010B1 | E9 4B010000 | CALL <JMP.>user32.GetDlgItemTextA | hMnd |
| 004010B7 | 0BC0 | OR EAX,EAX | GetDlgItemTextA |
| 004010B9 | < 75 15 | JNZ SHORT Ice9.0040113A | |
| 004010BA | > 6A 00 | PUSH 0 | |
| 004010BB | 68 04304000 | PUSH Ice9.00403004 | Style = MB_OK MB_APPLMODAL |
| 004010BC | 68 66304000 | PUSH Ice9.00403060 | Title = "Error, Bad Boy" |
| 004010C1 | 6A 00 | PUSH 0 | Text = "Where is the serial, Mr Cracker ?" |
| 004010C3 | E8 40010000 | CALL <JMP.>user32.MessageBoxA | hOwner = NULL |
| 004010C8 | EB 95 | OR EAX,EAX | MessageBoxA |
| 0040109A | E9 14000000 | JMP SHORT Ice9.0040113A | |
| 0040109F | > EB 09 | JMP SHORT Ice9.0040114A | |
| 004010A4 | > B8 00000000 | MOV EAX,0 | |
| 004010A9 | > C9 | LEAVE | |
| 004010B1 | > C2 1000 | RETN 10 | |
| 004010B7 | > B8 01000000 | MOV EAX,1 | |
| 004010B9 | > C9 | LEAVE | |
| 004010C1 | > 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 Icx9.00403004 | 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" |
| 00401203 | 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 Icx9.00403004 | ASCII "Error, Bad Boy" |
| 0040123A | PUSH Icx9.0040303A | ASCII "Hey!, you are DK?, its for Newbies..." |
| 0040124A | PUSH Icx9.00403004 | 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 | . 50 00000000 | CALL Ice9.004012B0 | | |
| 004011F5 | . 58 C8304000 | PUSH Ice9.004030C8 | | String2 = "" |
| 004011FA | . 58 98314000 | PUSH Ice9.00403198 | | String1 = "" |
| 004011FF | . E8 98000000 | CALL <JMP.&kernel32.lstrcpA> | | lstrcpA |
| 00401204 | . 0BC0 | OR EAX,EAX | | |
| 00401206 | √ 75 2B | JNZ SHORT Ice9.00401233 | | |
| 00401208 | . 58 84304000 | PUSH Ice9.004030B4 | | ASCII "DeAtH HaS cOMe" |
| 0040120D | . 58 83304000 | PUSH Ice9.00403083 | | 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 Ice9.00403083 | | 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 Ice9.00401212 |
| 00401233 | .J | 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 c0ME"
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 | 68B4304000 | PUSH  Ice9.004030B4
0040115B | E842010000 | CALL <JMP.&kerne132.1strlenA>
00401160 | 6A6A314000 | MOV  EDI,Ice9.0040316A
00401165 | 9902       | MOV  DWORD PTR DS:[EDI],EAX
00401167 | BF B4304000 | MOV  EDI,Ice9.004030B4
0040116C | 6619314000 | 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 | 6A 6A314000 | MOV  EDI,Ice9.0040316A
0040117E | 8B12       | MOV  EDI,DWORD PTR DS:[EDI]
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 | 2C 5A     | CMP  AL,5A
0040118B | 7E 05     | JLE  SHORT Ice9.00401192
0040118D | 83C8       | ADD  ECX,ERX
0040118F | 47        | INC  EDI
00401190 | EB EE     | JMP  SHORT Ice9.00401180
00401192 | 2C 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 | 330B       | XOR  EBX,EBX
004011B0 | 88C1       | MOV  EBX,ECX
004011B2 | B9 0A000000 | MOV  ECX,0A
004011B7 | 33D2       | XOR  EDX,EDX
004011B9 | F7F1      | DTZ  ECX
004011BB | 80C2 30    | ADD  DL,30
004011BE | 891433     | MOV  BYTE PTR DS:[EBX+ESI],DL
004011C1 | 83C3 01    | ADD  EBX,1
004011C4 | 83F8 00    | CMP  EBX,0
004011C7 | 74 02     | JE   SHORT Ice9.004011CB
004011C9 | EB EC     | JMP  SHORT Ice9.004011B7
004011CB | BF C2044000 | 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 | 33FB 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]
004011E6 | 68 B7304000 | PUSH  Ice9.004030E7
004011EB | 68 C2044000 | PUSH  Ice9.004030C3
004011F0 | 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 = "

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

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

```

00401153 | 55          | PUSH EBP
00401154 | 68 38C     | MOV EBP,ESP
00401156 | 68 B4304000 | PUSH Ice9.004030B4
0040115B | E8 4201000 | EB 4201000
00401160 | 6A 6A31400 | BA 6A31400
00401165 | 9902      | MOV EDI,Ice9.0040316A
00401167 | BF B430400 | MOV EDI,Ice9.004030B4
0040116C | 66 1951400 | 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 | 6A 6A31400 | MOV EDI,Ice9.0040316A
0040117E | 8B12     | MOV EDI,DWORD PTR DS:[EDI]
00401180 | 83C3 01  | ADD EBX,1
00401183 | 38D3     | 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 | 83C8     | ADD ECX,8
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 9A02000 | ADD ECX,29A
004011A2 | 69C9 3930000 | IMUL ECX,ECX,3039
004011A6 | 69C9 17    | SUB ECX,17
004011AB | 68C9 09    | IMUL ECX,ECX,9
004011AE | 33D0     | XOR EBX,EBX
004011B0 | 88C1     | MOV ECX,ECX
004011B2 | B9 0A000000 | MOV ECX,0A
004011B7 | 33D2     | XOR EDX,EDX
004011B9 | F7F1     | DTI 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 | 68 C204000 | 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 | 33FB 00    | CMP EBX,0
004011DE | 75 F3     | JNE SHORT Ice9.004011D0
004011DD | C607 00    | MOV BYTE PTR DS:[EDI],0
004011E0 | 803D B430400 | LEA EDI,DWORD PTR DS:[4030B4]
004011E6 | 68 E730400 | PUSH Ice9.004030E7
004011EB | 68 C204000 | PUSH Ice9.004030C3
004011F0 | E8 BB00000 | CALL Ice9.004012E0
004011F5 | 68 C304000 | PUSH Ice9.004030C3
004011FF | 68 9831400 | PUSH Ice9.00403198
004011FF | E8 9800000 | CALL <JMP_&kerne132.1stronpA>

```

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

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

```

00401153 | 55          | PUSH  EBP
00401154 | . 8BC8     | 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  EBX,EBX
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 | . 6A 6A314000 | MOV  EDI,Ice9.0040316A
0040117E | . 8B12     | MOV  EDX,0x00D0 PTR DS:[EDX]
00401180 | > 83C3 01  | CMP  EDI,EAX
00401183 | . 38D3     | CMP  EBX,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,EBX
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 | INVL ECX,ECX,3039
004011A8 | . 83E9 17   | SUB  ECX,1
004011AB | . 68C9 09   | INVL ECX,ECX,9
004011AD | . 33D0     | XOR  EBX,EBX
004011B0 | . 8BC1     | MOV  ECX,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   | JMP  SHORT Ice9.004011B7
004011CB | > . BF C9304000 | MOV  EDI,Ice9.004030C9
004011D0 | > . 8A433 FF | MOV  AL,BYTE PTR DS:[EBX+ESI-1]
004011D4 | . 8B07     | MOV  EBX,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,0x00D0 PTR DS:[4030B4]
004011E6 | . 68 E7304000 | PUSH  Ice9.004030E7
004011EB | . 68 C9304000 | PUSH  Ice9.004030C9
004011F0 | . E8 BB000000 | CALL  Ice9.004012E0
004011F5 | . 68 C9304000 | PUSH  Ice9.004030C9
004011FF | . 68 98314000 | PUSH  Ice9.00403198
004011FF | . E8 98000000 | CALL  <JMP.&kerne132.1strlenA>

```

- 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 | . E8 42010000 | CALL <JMP.&kerne132.1strlenA>
00401160 | . 8A 6A314000 | MOV EDI,Ice9.0040316A
00401165 | . 9902     | MOV DWORD PTR DS:[EDI],EAX
00401167 | . BF B4304000 | MOV EDI,Ice9.004030B4
0040116C | . EC 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 EDX,Ice9.0040316A
0040117E | . 8B12     | MOV EBX,ESI
00401180 | > 83C3 01  | CMP EBX,EBX
00401183 | . 3BD3     | JBE SHORT Ice9.0040119C
00401185 | . 74 15     | MOV AL,BYTE PTR DS:[EDI]
00401187 | . 8A07     | CMP AL,0A
00401189 | . 3C 5A     | JLE SHORT Ice9.00401192
0040118B | . 7E 05     | ROR ECX,EAX
0040118D | > 83C8     | INC EDI
0040118F | . 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     | ROR AL,0A
0040119A | > EB F1     | JMP SHORT Ice9.00401180
0040119C | > 81C1 9A0200 | ADD ECX,00000000
004011A2 | . 69C9 893000 | IMUL ECX,ECX,00000000
004011A6 | . 69C9 17  | IMUL ECX,ECX,0
004011AB | . 330B     | XOR EBX,EBX
004011AE | . 330B     | XOR ECX,ECX
004011B0 | . B9 0A000000 | MOV ECX,0A
004011B2 | > 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 EBX,ESI
004011C4 | . 83F8 00  | CMP EAX,0
004011C7 | . 74 02     | JBE SHORT Ice9.004011CB
004011C9 | . EB EC     | JMP SHORT Ice9.004011B7
004011CB | > . BF C9304000 | MOV EDI,Ice9.004030B3
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 B4304000 | PUSH Ice9.004030B7
004011E6 | . 68 C9304000 | PUSH Ice9.004030C9
004011FB | . E8 BB000000 | CALL Ice9.004012E9
004011FD | . 68 C9304000 | 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 = "

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 .<v 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
004011AD . 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 > 68C9 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 . 69C9 39300001 IMUL ECX,ECX,3039
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*+ = 0x30Guardar *DL* en el buffer *ESI* + *EBX**EBX* ++until *EAX* = 0

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 . 33D2 XOR EDI,EDI
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 . 48 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

$EDX = 0$

$EDX = EAX \bmod ECX$

$EAX /= ECX$

$DL += 0x30$

 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

```

$EDI = \text{buffer } 0x04030C8$

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 . 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.&kernel32.1strcmpA>

```

$EDI = \text{buffer } 0x04030C8$

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* 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,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>
004011FF

```

EDI = 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
 - La Ingeniería Inversa de código
- 2 **Conocimientos previos**
- 3 **Cinturón de herramientas**
 - Desensambladores y editores hexadecimales
 - Debuggers
 - Identificadores, editores PE y de recursos
 - Dumpeadores de memoria y emuladores
 - Monitores de APIs y reparadores de IAT
 - Documentación
- 4 **Técnicas de análisis**
 - Código muerto
 - Código 'vivo'
- 5 **Algunos métodos *anticracking***
 - Algunas APIs *antidebugging*
 - Métodos más avanzados
- 6 **Técnicas de *cracking***
 - CD Check
 - *Patching* y *loaders*
 - *Time-trials* y Registro de Windows
 - Captura del *serial* y *Keygenning*
 - Archivos de licencia
 - Desempacado (*unpacking*)
- 7 **Ejemplo práctico**
 - Estudio del crackME
 - Algoritmo de generación
- 8 **Conclusiones y agradecimientos**

Conclusiones y agradecimientos

Conclusiones

- 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' :)

Conclusiones y agradecimientos

Conclusiones

- 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' :)

Agradecimientos

- CrackSLatinoS
- Gente del HackMeeting

Conclusiones y agradecimientos

Conclusiones

- 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' :)

Agradecimientos

- CrackSLatinoS
- Gente del HackMeeting
- A vosotros por aguantarme! :D

El Arte de la Ingeniería Inversa

Ricardo J. Rodríguez



#eCh!2004 - .:[CrackSLatinoS]:
www.ech2004.net

23 de Octubre de 2010

HackMeeting 2010
Zaragoza, Spain