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

Introducción a la Ingeniería Inversa (I)

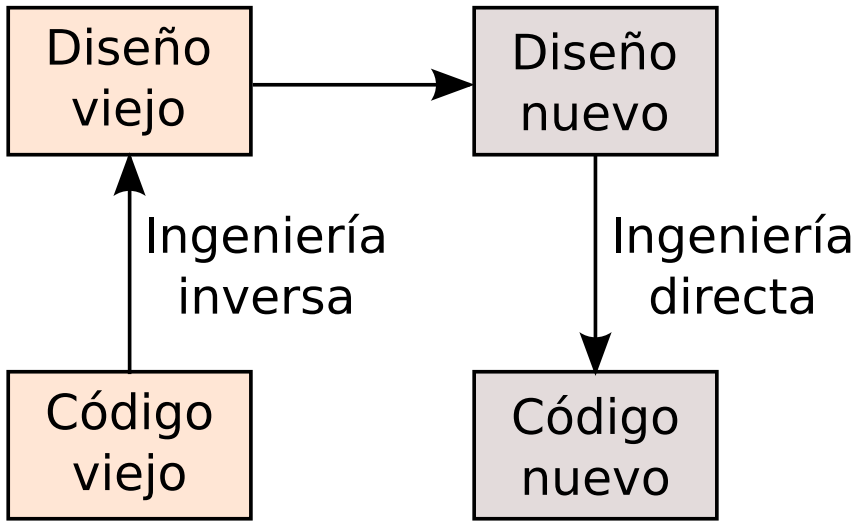
Ingeniería inversa (*reverse engineering*)

- 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)

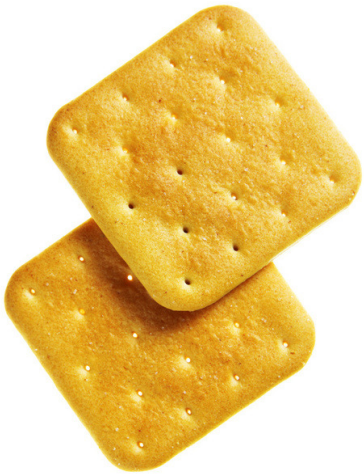
Reverse engineering code

- 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*

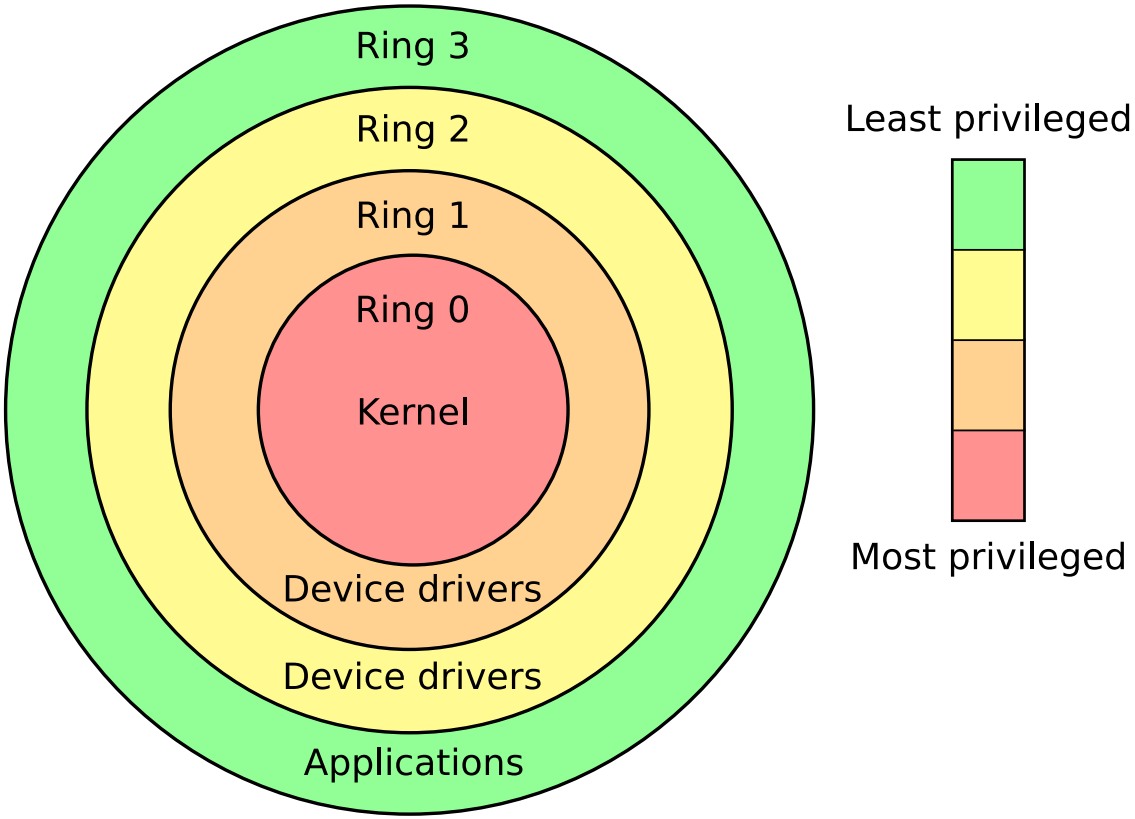


Conocimientos previos (I)

- 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.mspx`
- Debuggear programas
  - *Breakpoints*
  - *Step into*, *step over*, ...

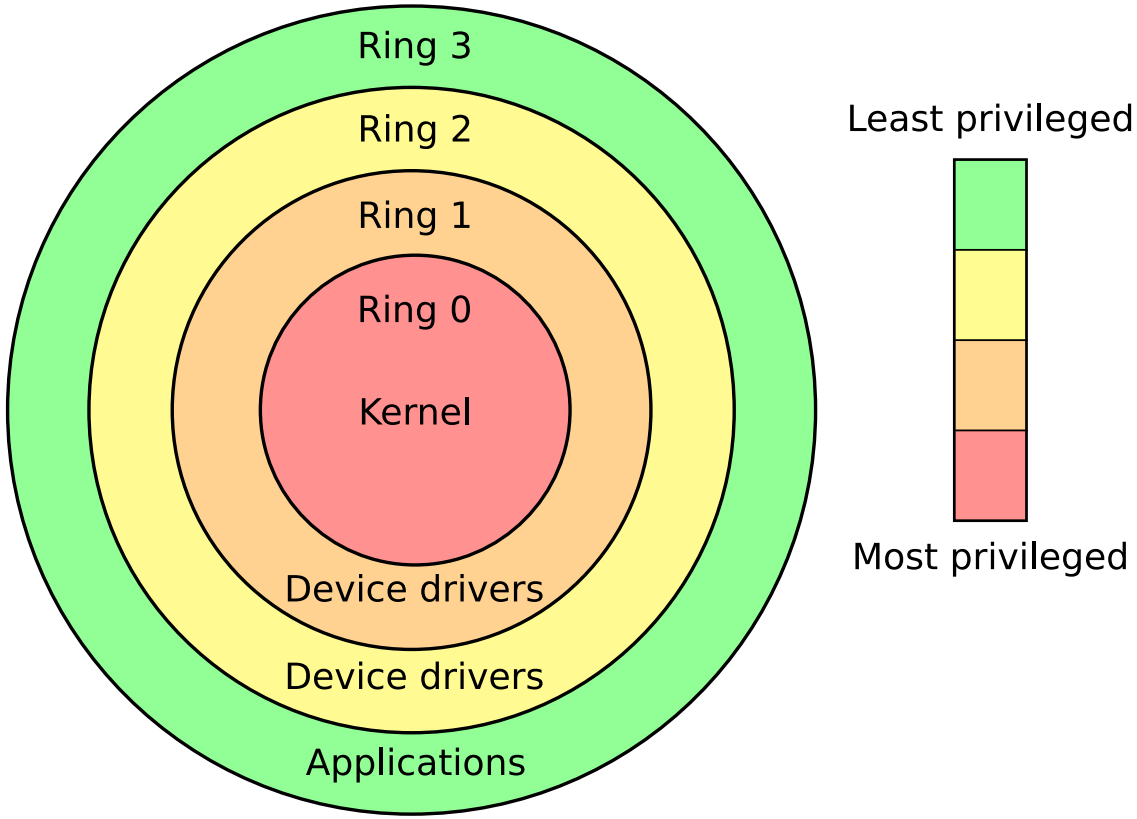
Conocimientos previos (II)

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



Conocimientos previos (II)

- Anillos de privilegio $R0 \ldots R3$
- 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...

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
- OllyDBG (`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
- Event2Adress (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/`)

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)

[a] Si el salto es largo (destino a más de 32 bits desde el lugar de origen), varía...

```
:004984AF 68488A5300          push 00538A48
:004984B4 E8CCFBFFFF          call 00498085
:004984B9 85C0                test eax, eax
:004984BB 0F8499000000        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 ¡!

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. . .)

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

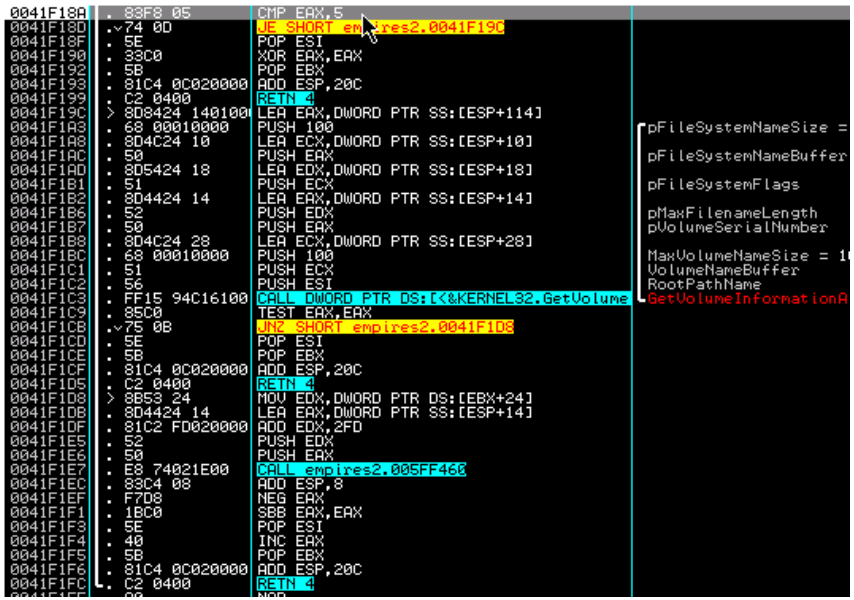
- Redireccionamiento de la IAT
- Secciones virtuales (*antidump*)
- Emulación de API
- Cambios en la estructura del ejecutable (*PE header*)
- Ofuscamiento de código (código basura)
- Bytes perdidos (*stolen bytes*)
- Detección de *debuggers*, otras *tools* o *breakpoints* (BP y HBP)
- Nanomites (usado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

Detección de modificaciones

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

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



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" $\rightarrow$ 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
- Chequeos 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
    $\rightarrow$ 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*

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)

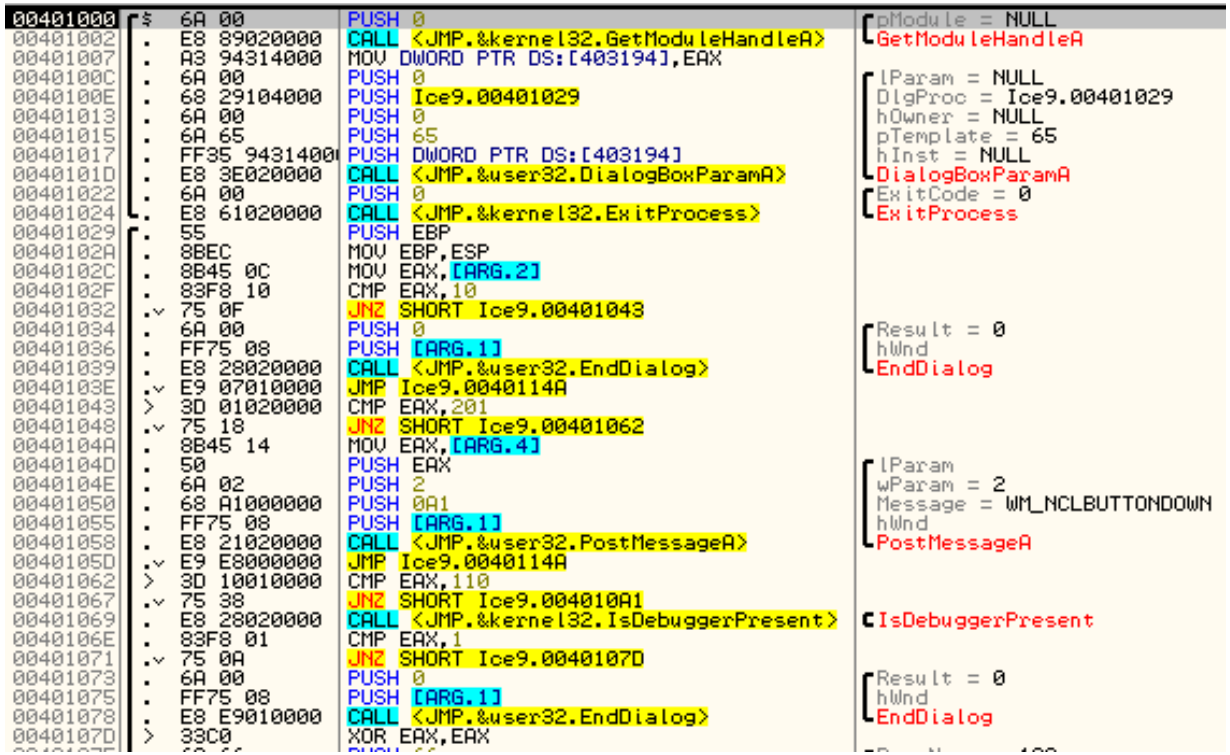
Ejemplo práctico (I): estudio del crackME

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

Pasos

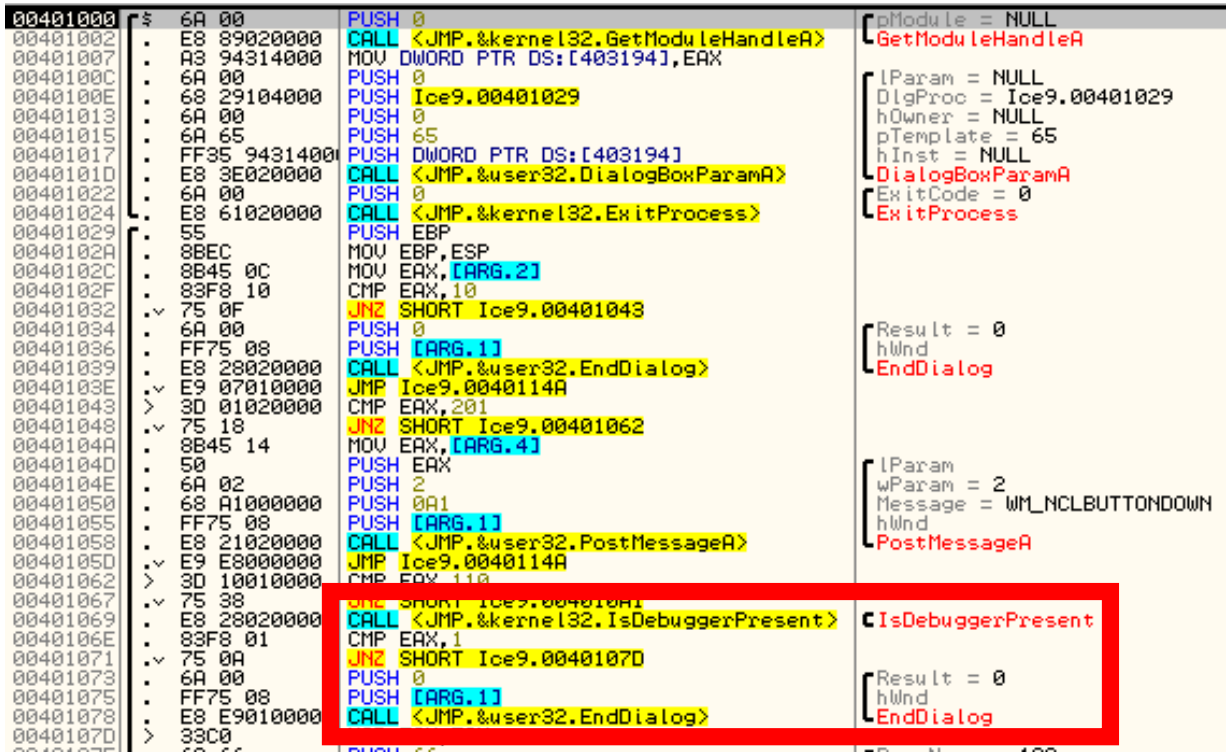
1. Analizar para ver si está limpio
2. Insertar nombre y código para ver chico malo
3. Cargar en *debugger* (e.g., OllyDBG)

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



Pasos

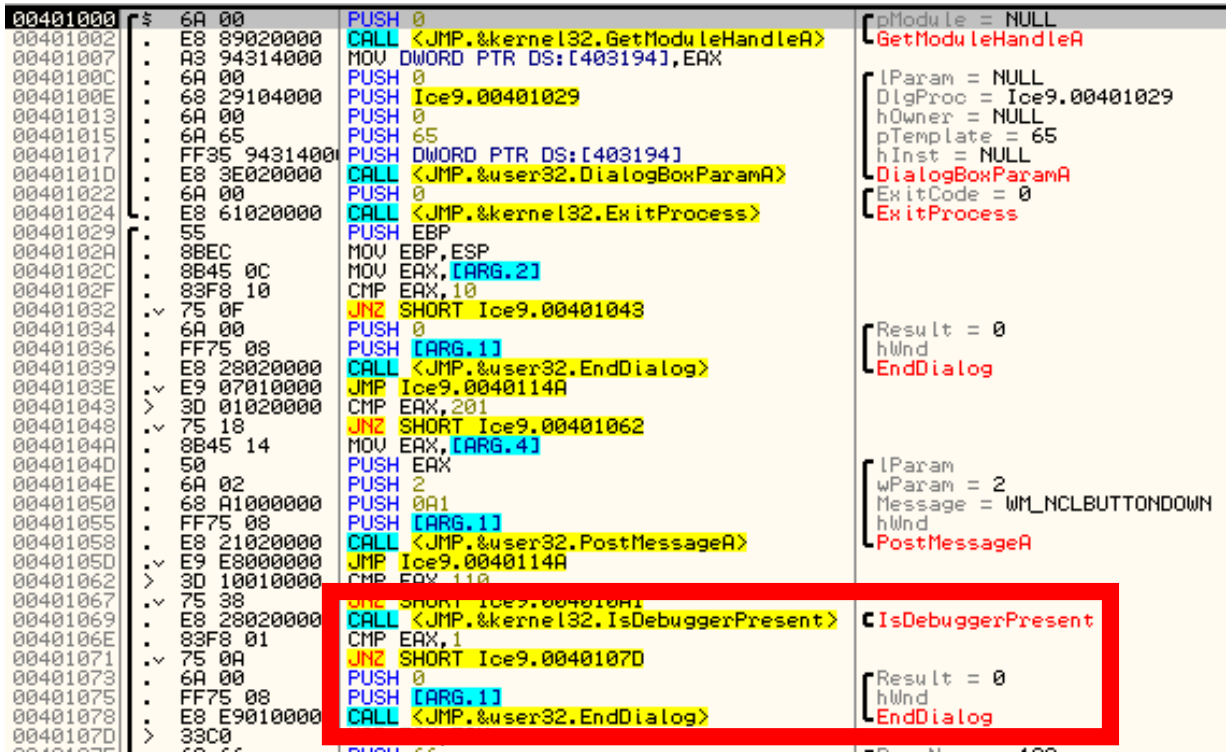
Ejemplo práctico (II): estudio del crackME (1)



Pasos

1. Llamada a `IsDebuggerPresent` → ocultar el Olly

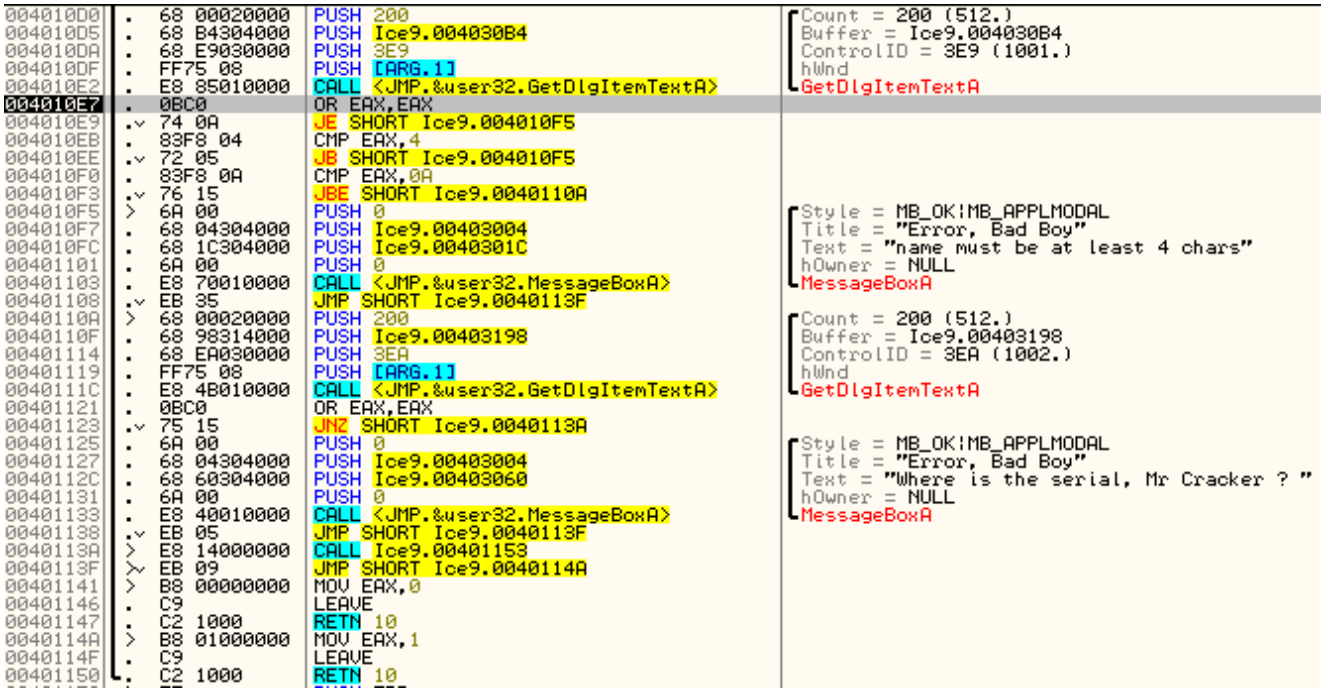
Ejemplo práctico (II): estudio del crackME (1)



Pasos

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

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



Pasos

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

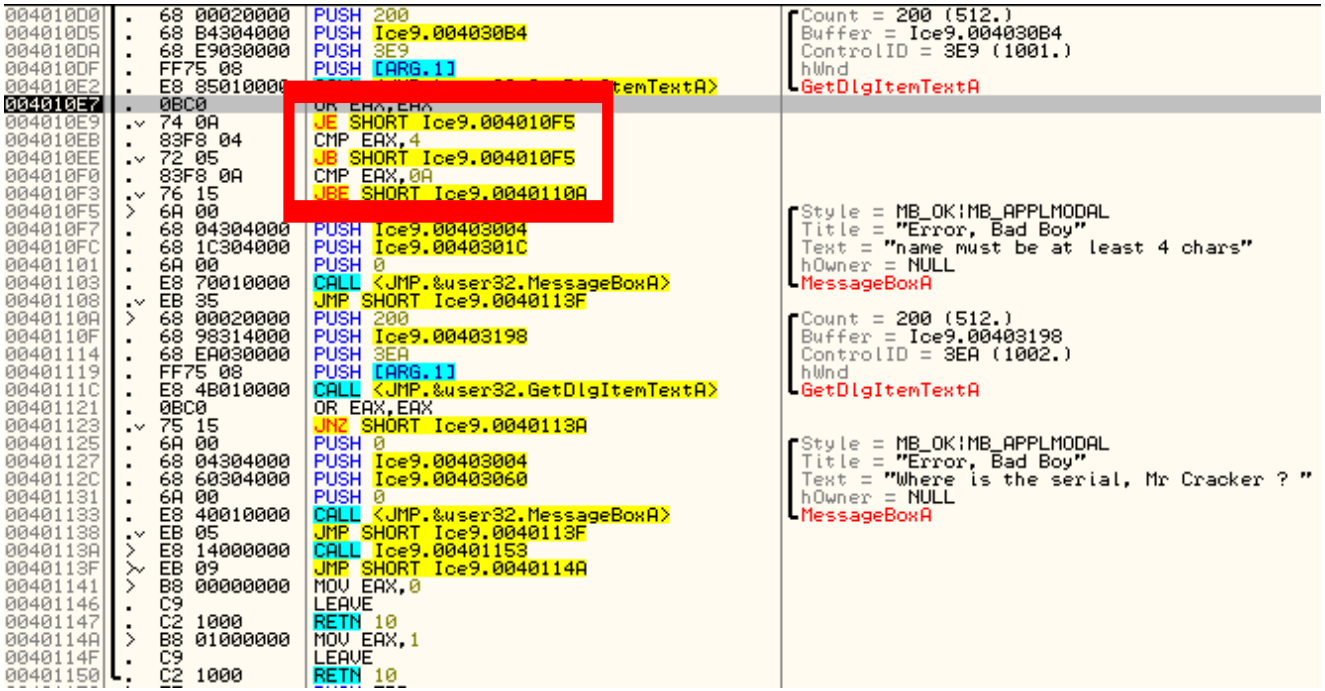
Ejemplo práctico (II): estudio del crackME (2)

```
004010D0  .  68 00020000   PUSH 200                          Count = 200 (512.)
004010D5  .  68 B4304000   PUSH Ice9.004030B4               Buffer = Ice9.004030B4
004010DA  .  68 E9030000   PUSH 3E9                          ControlID = 3E9 (1001.)
004010DF  .  FF75 08       PUSH [ARG.1]                      hWnd
004010E2  .  E8 85010000   CALL <JMP.&user32.GetDlgItemTextA> GetDlgItemTextA
004010E7  .  0BC0          OR EAX,EAX
004010E9  .v 74 0A         JE SHORT Ice9.004010F5
004010EB  .  83F8 04       CMP EAX,4
004010EE  .v 72 05         JB SHORT Ice9.004010F5
004010F0  .  83F8 0A       CMP EAX,0A
004010F3  .v 76 15         JBE SHORT Ice9.0040110A
004010F5  >  6A 00         PUSH 0                            Style = MB_OK|MB_APPLMODAL
004010F7  .  68 04304000   PUSH Ice9.00403004                Title = "Error, Bad Boy"
004010FC  .  68 1C304000   PUSH Ice9.0040301C                Text = "name must be at least 4 chars"
00401101  .  6A 00         PUSH 0                            hOwner = NULL
00401103  .  E8 70010000   CALL <JMP.&user32.MessageBoxA>    MessageBoxA
00401108  .v EB 35         JMP SHORT Ice9.0040113F
0040110A  >  68 00020000   PUSH 200                          Count = 200 (512.)
0040110F  .  68 98314000   PUSH Ice9.00403198                Buffer = Ice9.00403198
00401114  .  68 EA030000   PUSH 3EA                         ControlID = 3EA (1002.)
00401119  .  FF75 08       PUSH [ARG.1]                      hWnd
0040111C  .  E8 4B010000   CALL <JMP.&user32.GetDlgItemTextA> GetDlgItemTextA
00401121  .  0BC0          OR EAX,EAX
00401123  .v 75 15         JNZ SHORT Ice9.0040113A
00401125  .  6A 00         PUSH 0                            Style = MB_OK|MB_APPLMODAL
00401127  .  68 04304000   PUSH Ice9.00403004                Title = "Error, Bad Boy"
0040112C  .  68 60304000   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  .v EB 05         JMP SHORT Ice9.0040113F
0040113A  >  E8 14000000   CALL Ice9.00401153
0040113F  >v 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  L. C2 1000       RETN 10
```

Condiciones del nombre

- EAX: longitud del nombre

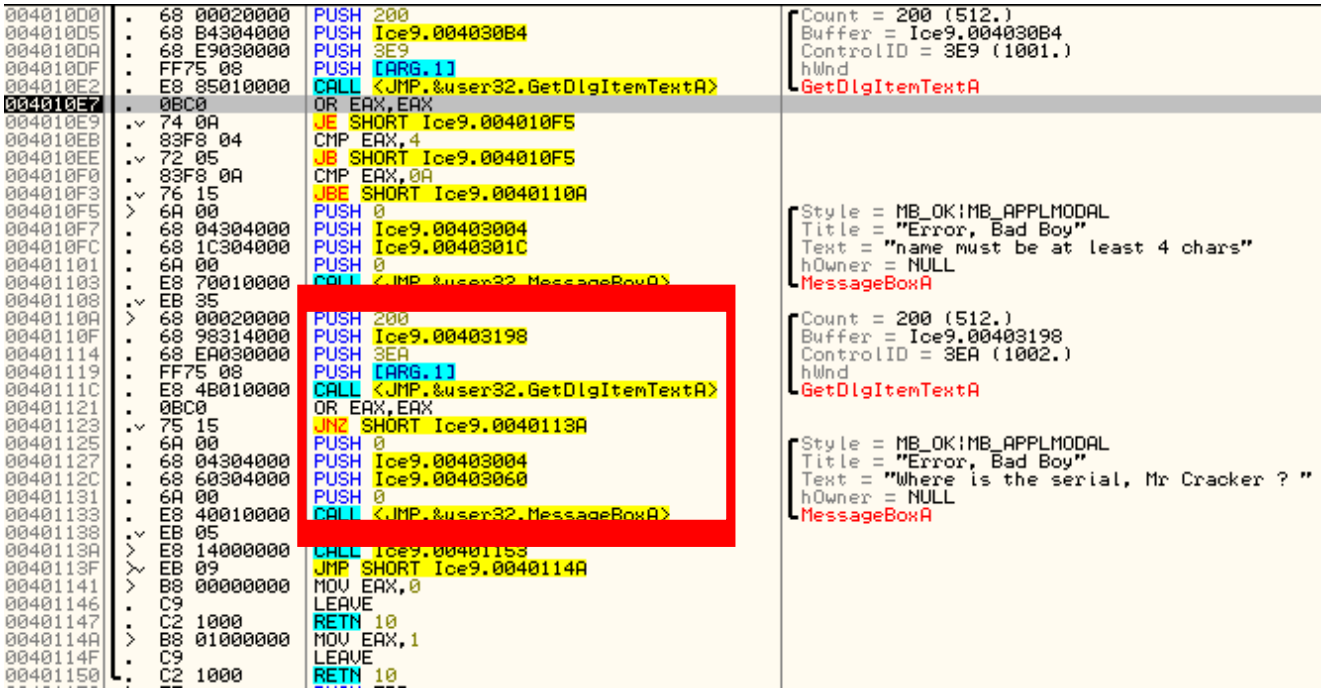
Ejemplo práctico (II): estudio del crackME (2)



Condiciones del nombre

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

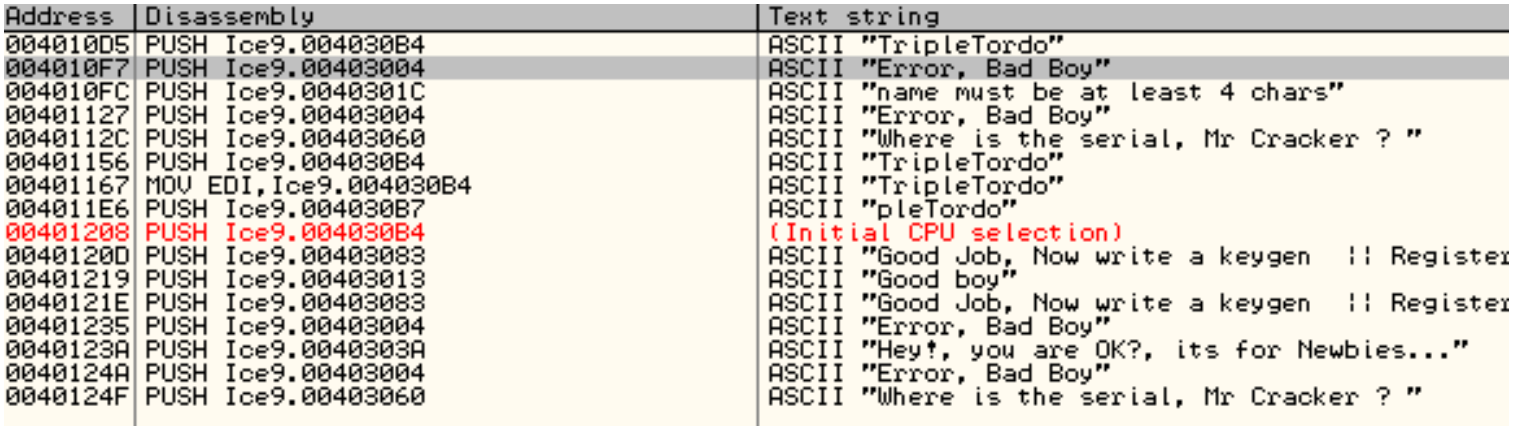
Ejemplo práctico (II): estudio del crackME (2)



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 Ice9.004030B4	ASCII "TripleTordo"
004010F7	PUSH Ice9.00403004	ASCII "Error, Bad Boy"
004010FC	PUSH Ice9.0040301C	ASCII "name must be at least 4 chars"
00401127	PUSH Ice9.00403004	ASCII "Error, Bad Boy"
0040112C	PUSH Ice9.00403060	ASCII "Where is the serial, Mr Cracker ? "
00401156	PUSH Ice9.004030B4	ASCII "TripleTordo"
00401167	MOV EDI,Ice9.004030B4	ASCII "TripleTordo"
004011E6	PUSH Ice9.004030B7	ASCII "pleTordo"
00401208	PUSH Ice9.004030B4	(Initial CPU selection)
0040120D	PUSH Ice9.00403083	ASCII "Good Job, Now write a keygen !! Register
00401219	PUSH Ice9.00403013	ASCII "Good boy"
0040121E	PUSH Ice9.00403083	ASCII "Good Job, Now write a keygen !! Register
00401235	PUSH Ice9.00403004	ASCII "Error, Bad Boy"
0040123A	PUSH Ice9.0040303A	ASCII "Hey!, you are OK?, its for Newbies..."
0040124A	PUSH Ice9.00403004	ASCII "Error, Bad Boy"
0040124F	PUSH Ice9.00403060	ASCII "Where is the serial, Mr Cracker ? "

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

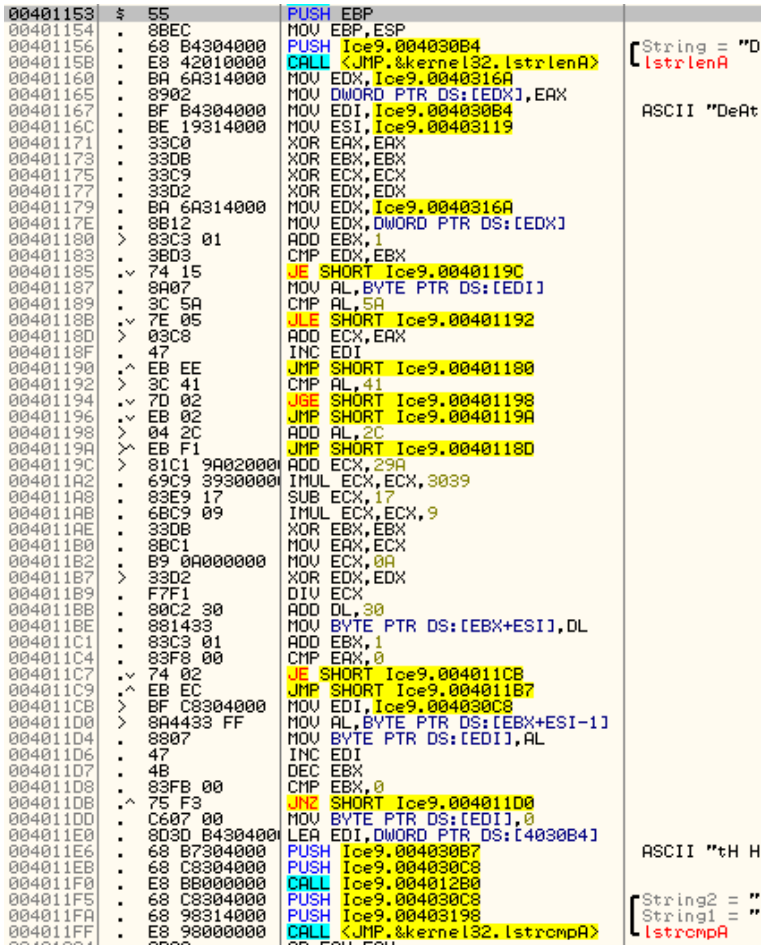
Ejemplo práctico (II): estudio del crackME (3)

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

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

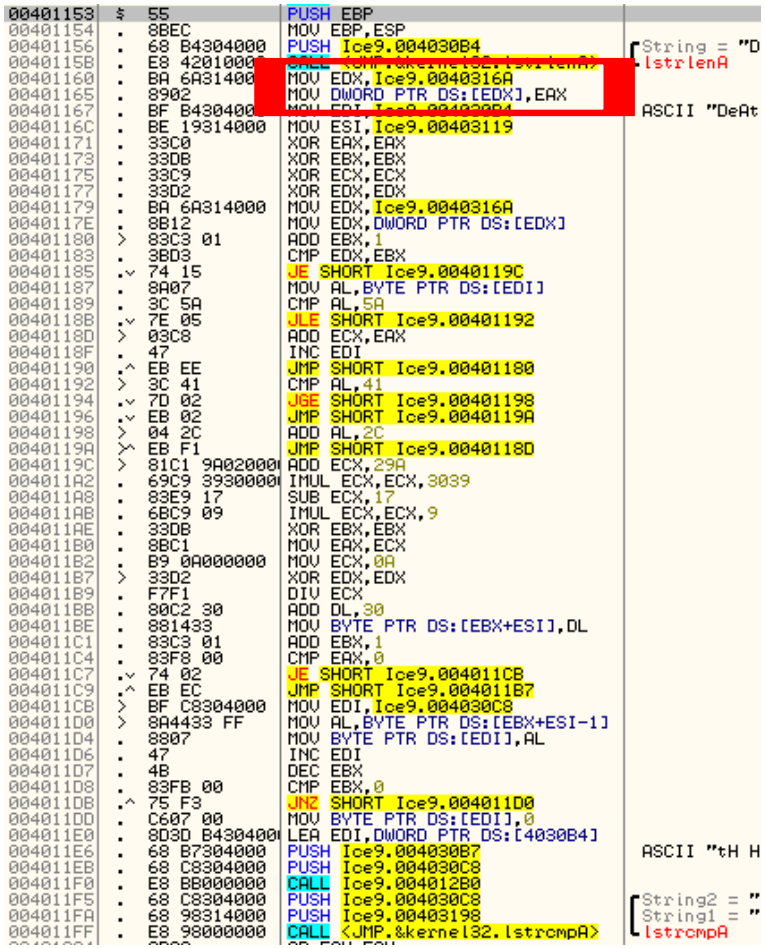
- Búsqueda de cadenas referenciadas
- Cadenas de 'chico bueno' y 'chico malo'
- `lstrcmpA`: comparación de cadenas
  - Valor de EAX determina igualdad

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



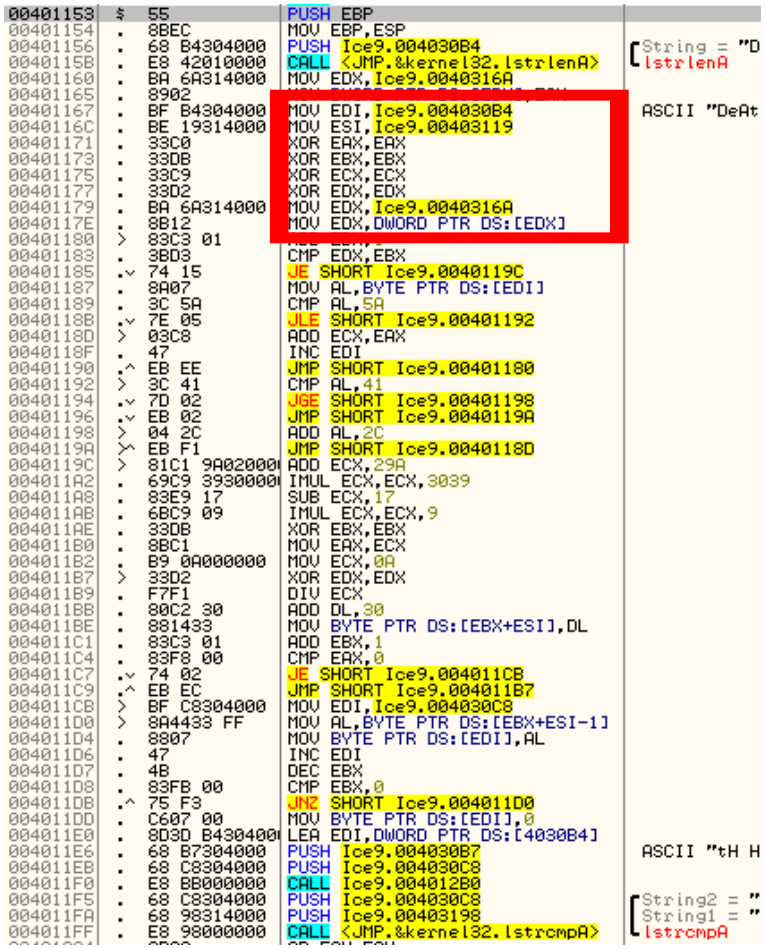
- 0x401153: Inicio algoritmo
- BP para estudio 'en caliente'

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



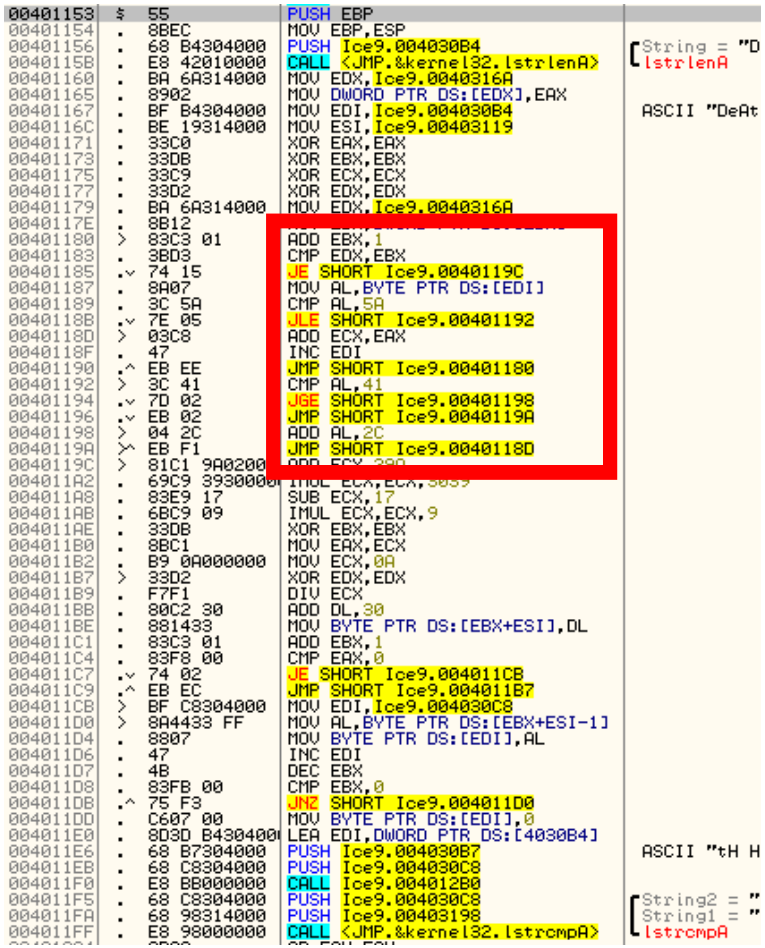
- 0x401153: Inicio algoritmo
- BP para estudio 'en caliente'
- 0x40316A: buffer longitud

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



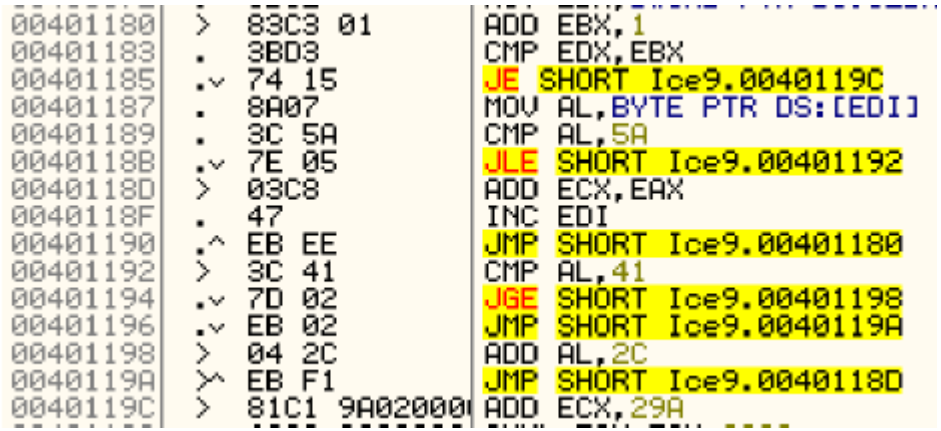
- 0x401153: Inicio algoritmo
- BP para estudio 'en caliente'
- 0x40316A: buffer longitud
- EDI: buffer cadena
- ESI: otro buffer :)
- EAX=EBX=ECX=EDX=0
- EDX=longitud

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



- 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  .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  >  04 2C          ADD AL,2C
0040119A  >^ EB F1          JMP SHORT Ice9.0040118D
0040119C  >  81C1 9A02000(  ADD ECX,29A
```

EDI = buffer del nombre
EDX = longitud del nombre
$EBX = 1$
while $EBX < EDX$ **do**
  AL = carácter apuntado por EDI
  if $AL \leq 0x5A$ **and** $AL \geq 0x41$ **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  >  04 2C          ADD AL,2C
0040119A  >^ EB F1          JMP SHORT Ice9.0040118D
0040119C  >  81C1 9A02000(  ADD ECX,29A
```

EDI = buffer del nombre
EDX = longitud del nombre
$EBX = 1$
while $EBX < EDX$ **do**
  AL = carácter apuntado por EDI
  if $AL \leq 0x5A$ **and** $AL \geq 0x41$ **then**
    $AL+ = 0x2C$
  end
  $ECX+ = EAX$
  $EDI++$
end

```
0040119C  > 81C1 9A02000(  ADD ECX,29A
004011A2  .  69C9 3930000(  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  .  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  >  04 2C          ADD AL,2C
0040119A  >^ EB F1          JMP SHORT Ice9.0040118D
0040119C  >  81C1 9A02000(  ADD ECX,29A
```

EDI = buffer del nombre
EDX = longitud del nombre
$EBX = 1$
while $EBX < EDX$ **do**
  AL = carácter apuntado por EDI
  if $AL \leq 0x5A$ **and** $AL \geq 0x41$ **then**
    $AL+ = 0x2C$
  end
  $ECX+ = EAX$
  $EDI++$
end

```
0040119C  > 81C1 9A02000(  ADD ECX,29A
004011A2  .  69C9 3930000(  IMUL ECX,ECX,3039
004011A8  .  83E9 17        SUB ECX,17
004011AB  .  6BC9 09        IMUL ECX,ECX,9
004011AE  .  [illegible]
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
```

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

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

```
0040119C  > 81C1 9A020000  ADD ECX,29A
004011A2  .  69C9 39300000  IMUL ECX,ECX,3039
004011A8  .  83E9 17        SUB ECX,17
004011AB
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
```

$EBX = 0$
$EAX = ECX$
$ECX = 0x0A$
repeat
  $EDX = 0$
  $EDX = EAX$ **mod** ECX
  $EAX /= ECX$
  $DL += 0x30$
  Guardar DL en el buffer $ESI + EBX$
  $EBX++$
until $EAX = 0$

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

```
0040119C  > 81C1 9A020000  ADD ECX,29A
004011A2  .  69C9 39300000  IMUL ECX,ECX,3039
004011A8  .  83E9 17        SUB ECX,17
004011AB  .  6BC9 0A        IMUL ECX,ECX,0A
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
```

$EBX = 0$
$EAX = ECX$
$ECX = 0x0A$
repeat
  $EDX = 0$
  $EDX = EAX$ **mod** ECX
  $EAX /= ECX$
  $DL += 0x30$
  Guardar DL en el buffer $ESI + EBX$
  $EBX++$
until $EAX = 0$

```
004011CB  > BF 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,DWORD PTR DS:[4030B4]
004011E6  .  68 B7304000    PUSH Ice9.004030B7
004011EB  .  68 C8304000    PUSH Ice9.004030C8
004011F0  .  E8 BB000000    CALL Ice9.004012B0
004011F5  .  68 C8304000    PUSH Ice9.004030C8
004011FA  .  68 98314000    PUSH Ice9.00403198
004011FF  .  E8 98000000    CALL <JMP.&kernel32.lstrcmpA>
```

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

```
0040119C  > 81C1 9A020000  ADD ECX,29A
004011A2  .  69C9 39300000  IMUL ECX,ECX,3039
004011A8  .  83E9 17        SUB ECX,17
004011AB
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
```

$EBX = 0$
$EAX = ECX$
$ECX = 0x0A$
repeat
  $EDX = 0$
  $EDX = EAX$ **mod** ECX
  $EAX / = ECX$
  $DL+ = 0x30$
  Guardar DL en el buffer $ESI + EBX$
  $EBX + +$
until $EAX = 0$

```
004011CB  > BF 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
004011E0  .  8D3D B4304000  LEA EDI,DWORD PTR DS:[4030B4]
004011E6  .  68 B7304000    PUSH Ice9.004030B7
004011EB  .  68 C8304000    PUSH Ice9.004030C8
004011F0  .  E8 BB000000    CALL Ice9.004012B0
004011F5  .  68 C8304000    PUSH Ice9.004030C8
004011FA  .  68 98314000    PUSH Ice9.00403198
004011FF  .  E8 98000000    CALL <JMP.&kernel32.lstrcmpA>
```

EDI = buffer 0x04030C8
Revertimos la cadena apuntada por ESI y guardamos en EDI

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

```
0040119C  > 81C1 9A020000  ADD ECX,29A
004011A2  .  69C9 39300000  IMUL ECX,ECX,3039
004011A8  .  83E9 17        SUB ECX,17
004011AB
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
```

$EBX = 0$
$EAX = ECX$
$ECX = 0x0A$
repeat
  $EDX = 0$
  $EDX = EAX$ **mod** ECX
  $EAX / = ECX$
  $DL+ = 0x30$
  Guardar DL en el buffer $ESI + EBX$
  $EBX++$
until $EAX = 0$

```
004011CB  > BF 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.lstrcmpA>
```

$EDI =$ buffer 0x04030C8
Revertimos la cadena apuntada por ESI y guardamos en EDI
Añadimos resto del nombre (desde el 4^o carácter) al final de la nueva cadena

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

```
0040119C  > 81C1 9A020000  ADD ECX,29A
004011A2  .  69C9 39300000  IMUL ECX,ECX,3039
004011A8  .  83E9 17        SUB ECX,17
004011AB
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
```

$EBX = 0$
$EAX = ECX$
$ECX = 0x0A$
repeat
  $EDX = 0$
  $EDX = EAX$ **mod** ECX
  $EAX / = ECX$
  $DL+ = 0x30$
  Guardar DL en el buffer $ESI + EBX$
  $EBX++$
until $EAX = 0$

```
004011CB  > BF 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,DWORD PTR DS:[4030B4]
004011E6  .  68 B7304000    PUSH Ice9.004030B7
004011EB  .  68 C8304000    PUSH Ice9.004030C8
004011F0
004011F5  .  68 C8304000    PUSH Ice9.004030C8
004011FA  .  68 98314000    PUSH Ice9.00403198
004011FF  .  E8 98000000    CALL <JMP.&kernel32.lstrcmpA>
```

$EDI =$ 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

Conclusiones y agradecimientos

Conclusiones

- Cualquier protección es *crackeable*
- Mundo de constante evolución $\rightarrow$ 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 $\rightarrow$ 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 $\rightarrow$ 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