Introducción a la Ingeniería Inversa
Diseño de Aplicaciones Seguras

Ricardo J. Rodríguez
rjrodriguez@unizar.es

Universidad
Zaragoza

19 de Enero de 2012

Grupo de Ingeniería de Sistemas de Eventos Discretos
Universidad de Zaragoza

Outline

Universidad Zaragoza 1542

Universidad Zaragoza 1542

Introducción a la Ingeniería Inversa (I)

Ingeniería inversa (*reverse engineering*)

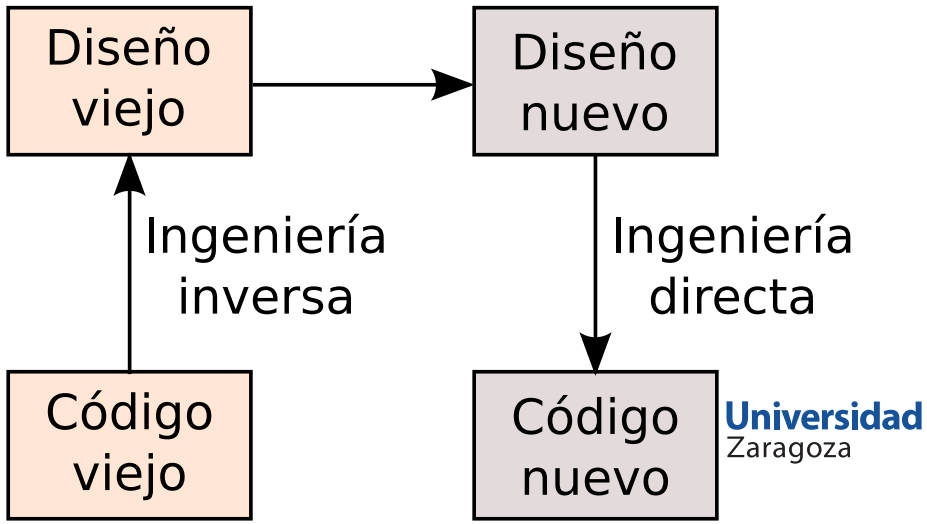
- Descubrir cómo funciona (algo) a partir de un análisis exhaustivo
- Mejora de productos/sistemas viejos
- Diferentes dominios de aplicación
  - *Hardware* (*legacy hardware*)
  - *Software* (e.g. Samba)

Universidad Zaragoza 1542

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



Universidad Zaragoza

Introducción a la Ingeniería Inversa (II)

Motivación

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

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

Encontrar vulnerabilidades en el software

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

Universidad Zaragoza 1542

Introducción a la Ingeniería Inversa (III)

Aproximaciones

- Caja blanca
  - Analizar y entender el código fuente (o binario desensamblado)
  - Encontrar errores de programación y/o implementación
  - Analizador estático: búsqueda de patrones (¿falso positivo?)
  - Ejemplos: *WhiteBox SecureAssistant*, *IDAPro*, *SourceScope*...
- Caja negra
  - Analizar programa según diferentes entradas → software en ejecución
  - No es tan efectivo, pero requiere menos experiencia
  - Método habitual para detectar exploits
- Caja gris
  - Combinación de las dos anteriores
  - Ejecución de programa mediante debug...
  - ...alimentándolo con diferentes entradas
  - Ejemplos: Rational's Purify (análisis uso/consumo memoria), Valgrind

Introducción a la Ingeniería Inversa (III)

Reverse engineering code

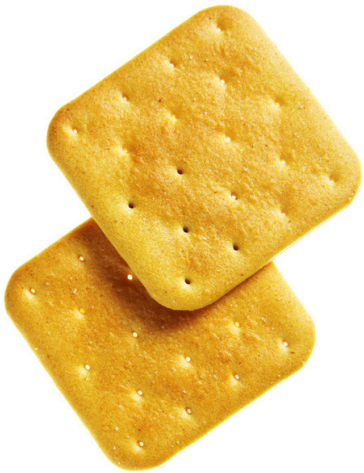
- También conocida como *cracking*
- Eliminar protecciones de código (*copyrights*)
- NO siempre es malo: detección de *bugs*, potenciales *exploits*, ...
- Lucha contra el malware

Universidad Zaragoza 1542

Introducción a la Ingeniería Inversa (III)

Reverse engineering code

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



Universidad Zaragoza

Universidad Zaragoza 1542

Conocimientos previos (I)

- Ensamblador (`http://www.intel.com/products/processor/manuals/`)

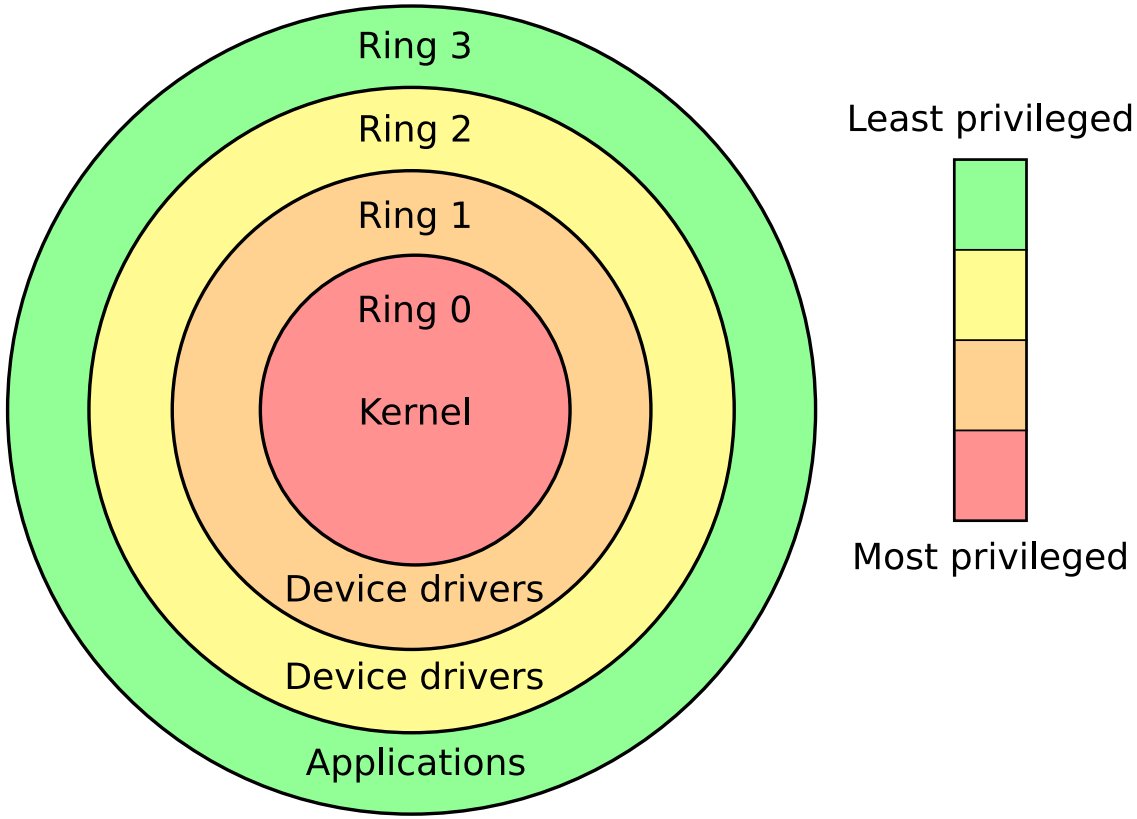
Conocimientos previos (I)

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

Universidad Zaragoza 1542

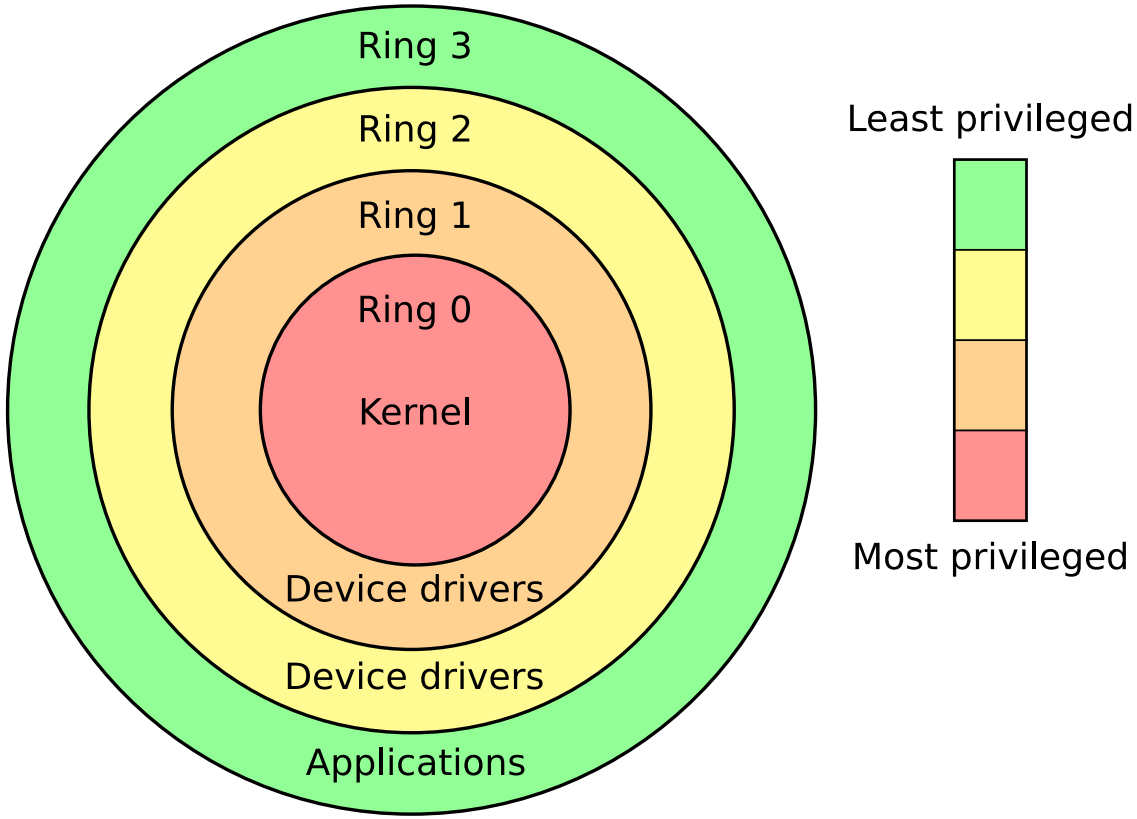
Conocimientos previos (II)

- 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)



"When a program runs, then it can be cracked"

Universidad Zaragoza 1542

Conocimientos de manejo de debuggers

Nociones de debugger

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

Universidad Zaragoza
1542

Conocimientos de arquitectura x86 básicos (I)

Registros

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

Universidad Zaragoza 1542

Conocimientos de arquitectura x86 básicos (II)

Registros de propósito general

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

Universidad Zaragoza 1542

Conocimientos de arquitectura x86 básicos (III) (1)

Registros de bandera (*flags*)

- Cada bit significa una cosa diferente

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

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

Conocimientos de arquitectura x86 básicos (III) (2)

Registros de bandera (*flags*)

- Cada bit significa una cosa diferente

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

- *AF*: auxiliar (operaciones BCD)
- *DF*: dirección (ops. de cadena, dirección ascendete o descendente en recorrido)
- *IF*: interrupciones (0 $\rightarrow$ "enmascaramiento" de ints)
- *TF*: *trap flag* (ejecución paso a paso, ya veremos...)

Universidad Zaragoza 1542

Conocimientos de arquitectura x86 básicos (IV) (1)

Tipos de instrucciones

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

Universidad Zaragoza 1542

Conocimientos de arquitectura x86 básicos (IV) (2)

Tipos de instrucciones

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

Universidad Zaragoza 1542

Conocimientos de arquitectura x86 básicos (V) (3)

Flujo del programa

- **jmp**
- **call** $\rightarrow$ **ret**
- Pueden ser cercanos o lejanos, según destino

La pila

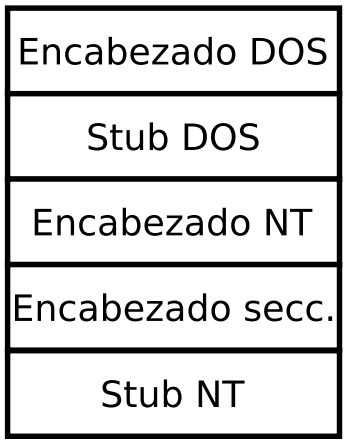
- LIFO (*Last In First Out*)
- Paso de parámetros a procedimientos
- **push** / **pop**
- **call**
- **ret**

push ebp
mov ebp, esp
sub esp, SIZE_LOCAL_VARIABLES

Código de la subrutina
mov esp, ebp
pop ebp
ret SIZE_INPUT_PARAMS; `4 by default`

Universidad Zaragoza 1542

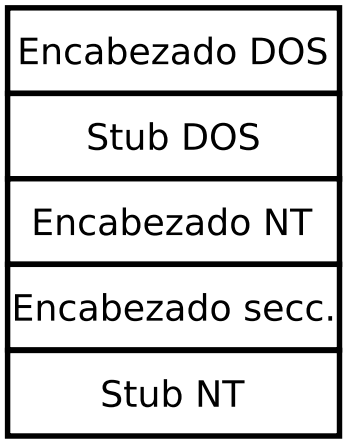
Conocimientos de las estructuras PE (I)



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

Zaragoza 1542

Conocimientos de las estructuras PE (II)



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

Universidad Zaragoza 1542

Conocimientos de las estructuras PE (II)

Referencias

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

Universidad Zaragoza 1542

Universidad Zaragoza 1542

Herramientas útiles (I)

Básicas

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

Universidad Zaragoza 1542

Herramientas útiles (II)

Otras...

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

Herramientas útiles (II)

Otras...

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

Universidad Zaragoza 1542

Herramientas útiles (III)

Documentación: manuales y tutoriales

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

Herramientas útiles (III)

Documentación: manuales y tutoriales

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

Universidad Zaragoza 1542

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 y desensamblar
2. Buscar mensajes de chico malo
3. Analizar camino hasta el mensaje
4. NOPear salto/desviar camino

Análisis de código muerto: ejemplos (II)

A la caza del *serial*

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

Pasos

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

Universidad Zaragoza 1542

Análisis de código 'vivo': descripción

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

(luego veremos un ejemplo...)

Introducción a la Ingeniería Inversa
Diseño de Aplicaciones Seguras

Ricardo J. Rodríguez
rjrodriguez@unizar.es

Universidad
Zaragoza

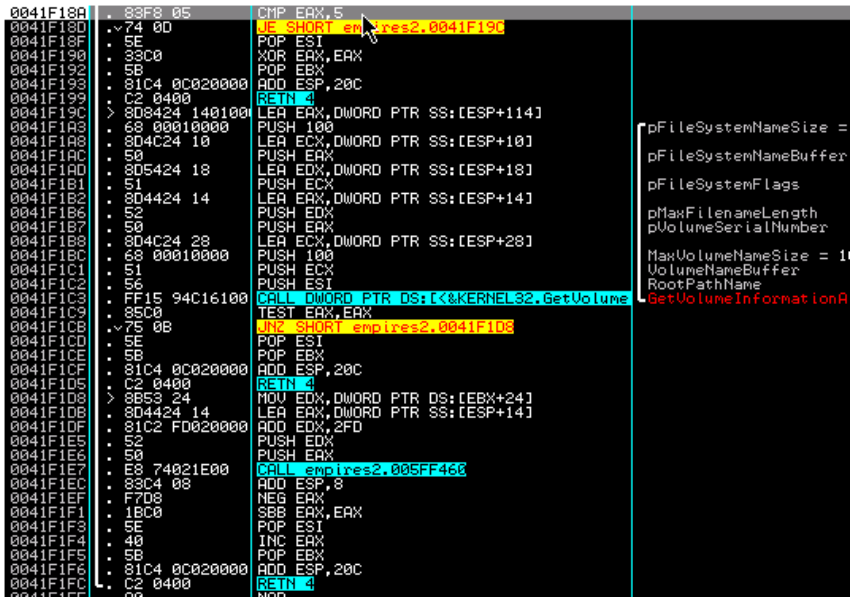
19 de Enero de 2012

Grupo de Ingeniería de Sistemas de Eventos Discretos
Universidad de Zaragoza

Universidad Zaragoza
1542

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`

Universidad Zaragoza

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

Universidad Zaragoza 1542

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...

Universidad Zaragoza
1542

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*

Universidad Zaragoza 1542

Algunos métodos *antireversing*

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

Universidad Zaragoza 1542

Universidad Zaragoza
1542

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)

Universidad Zaragoza

Ejemplo práctico (I): estudio del crackME

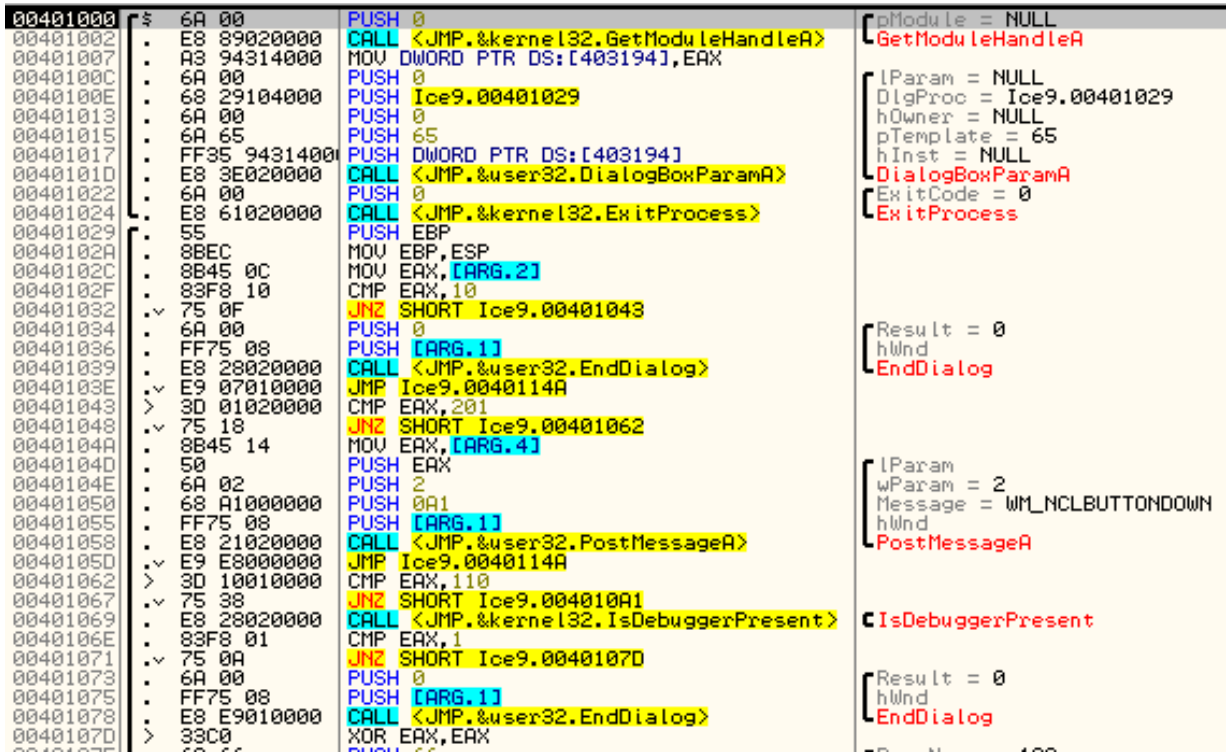
- *CrackMe* sencillito de `http://www.crackmes.de`
- Objetivo: hacer un generador de claves

Pasos

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

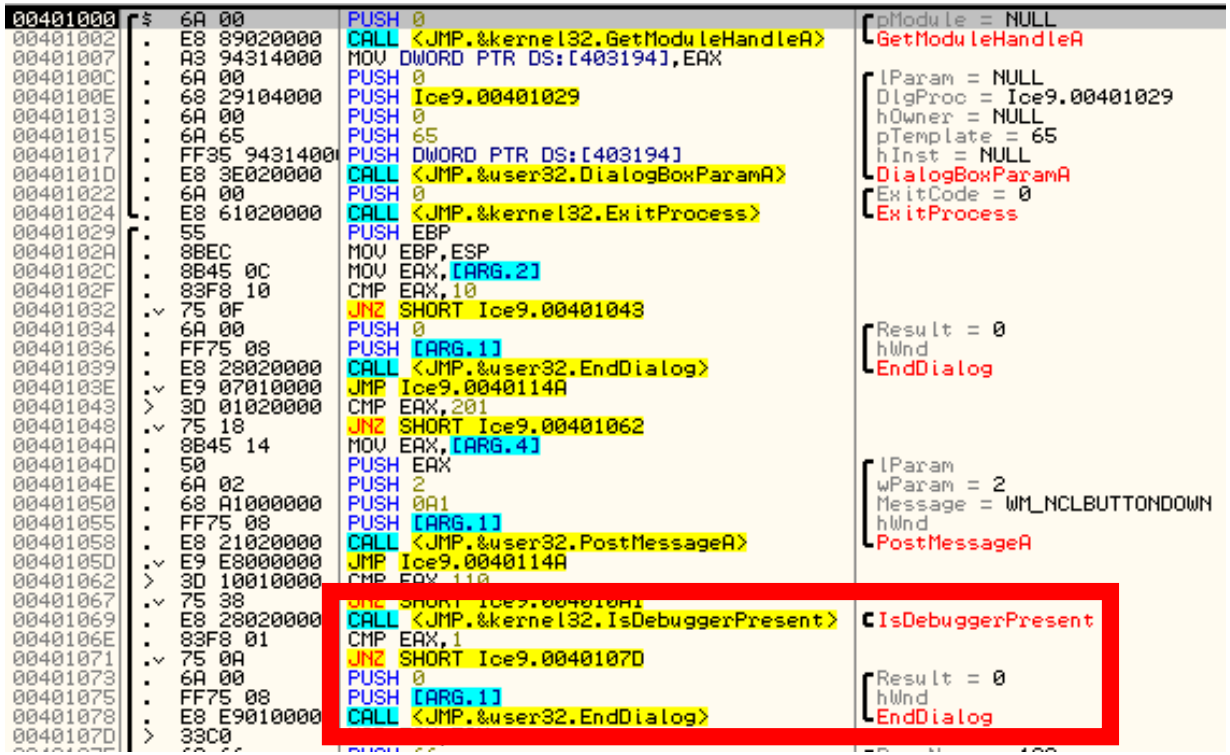
Universidad Zaragoza

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



Pasos

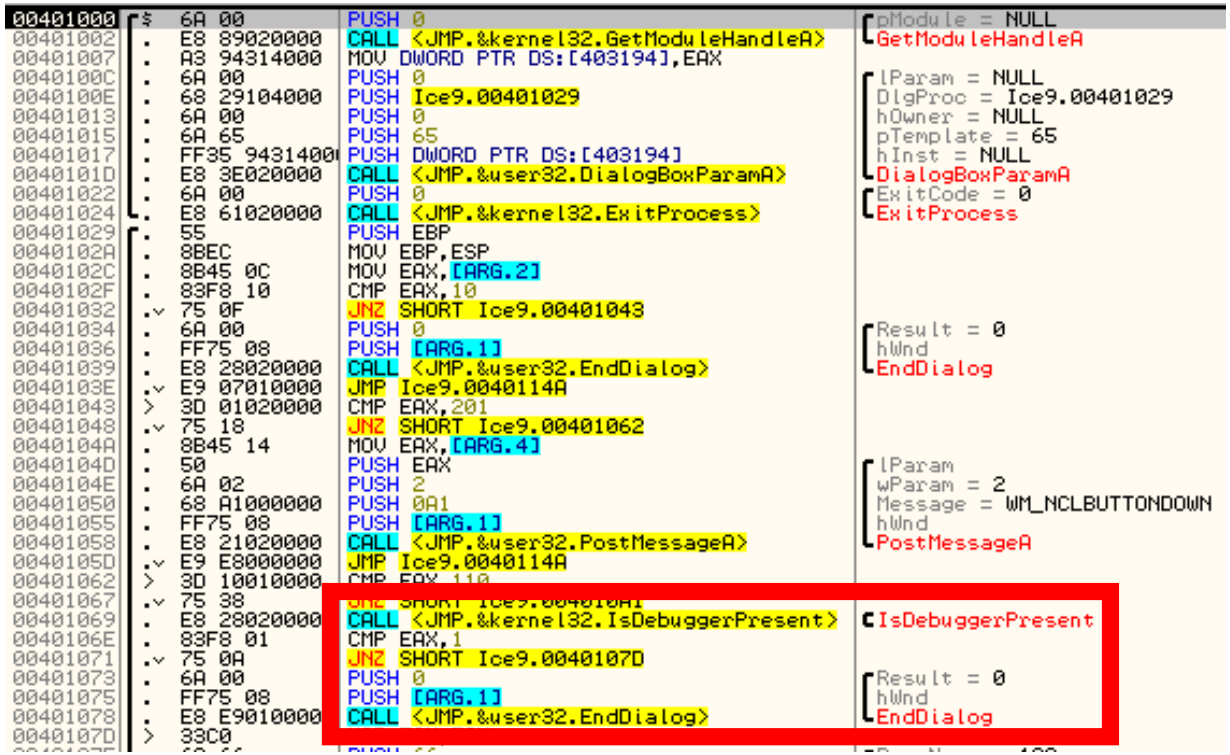
Ejemplo práctico (II): estudio del crackME (1)



Pasos

1. Llamada a `IsDebuggerPresent` → ocultar el Olly

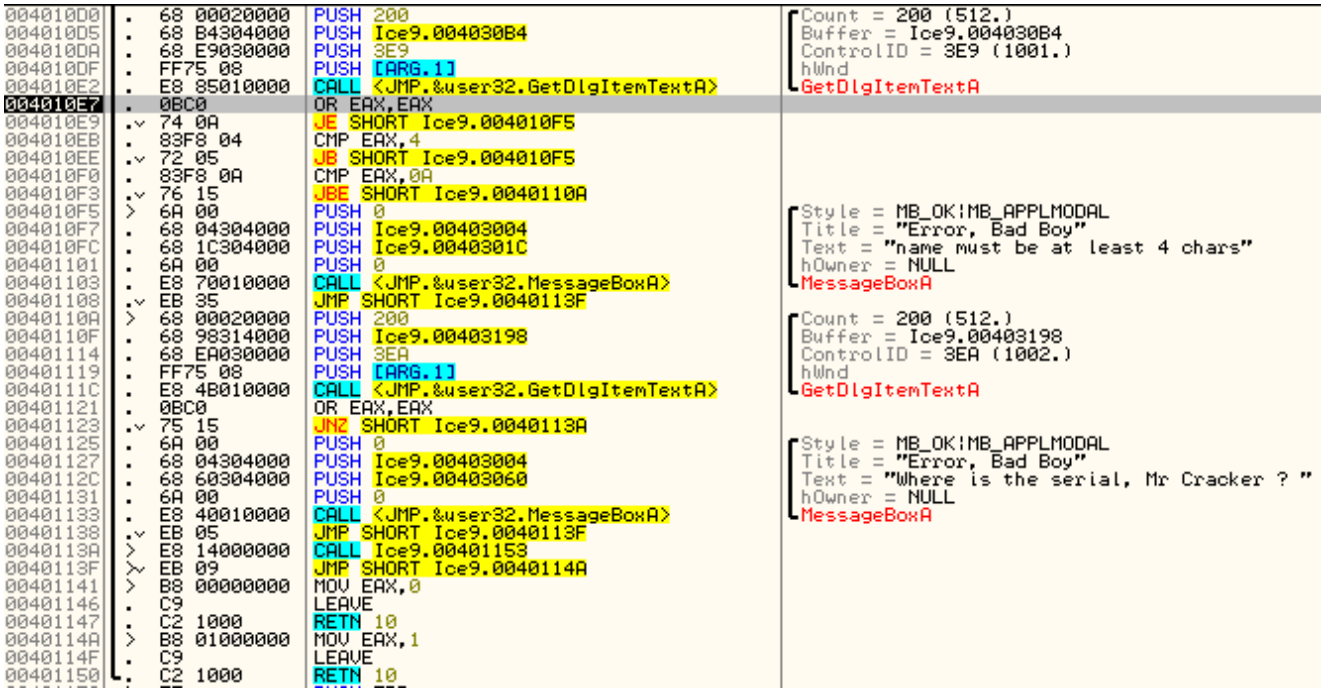
Ejemplo práctico (II): estudio del crackME (1)



Pasos

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

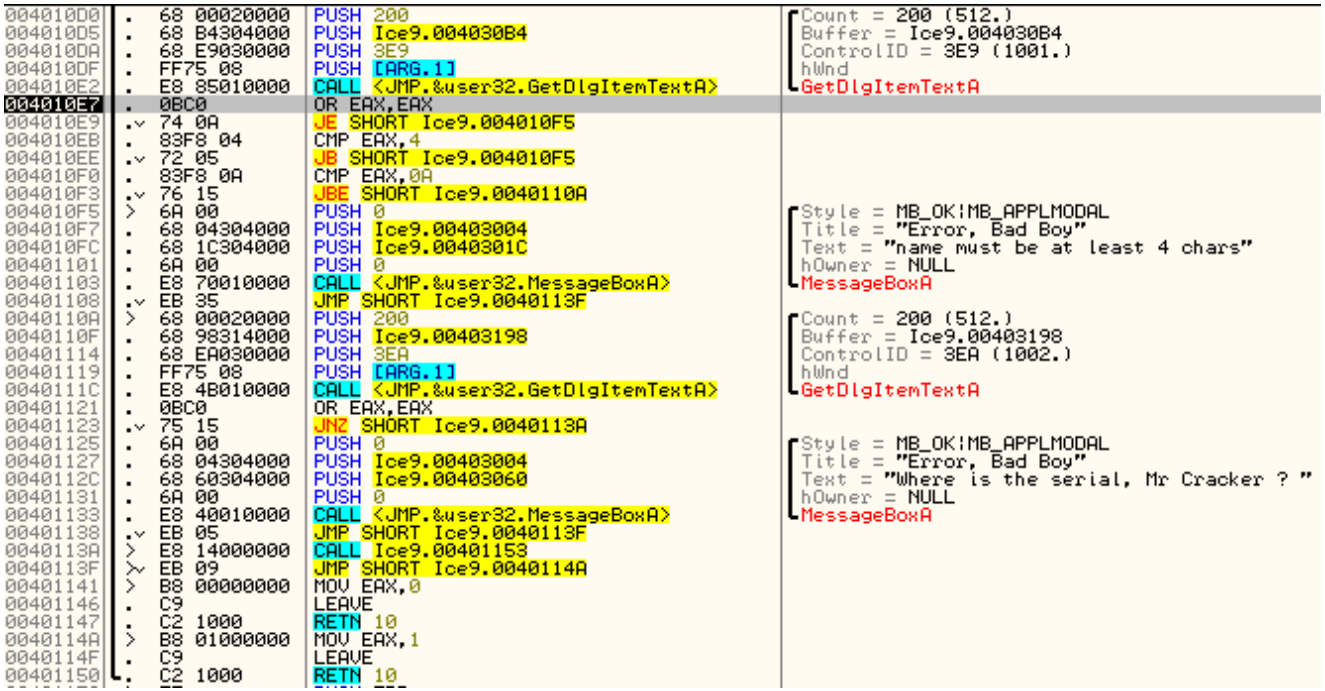
Ejemplo práctico (II): estudio del crackME (1)



Pasos

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

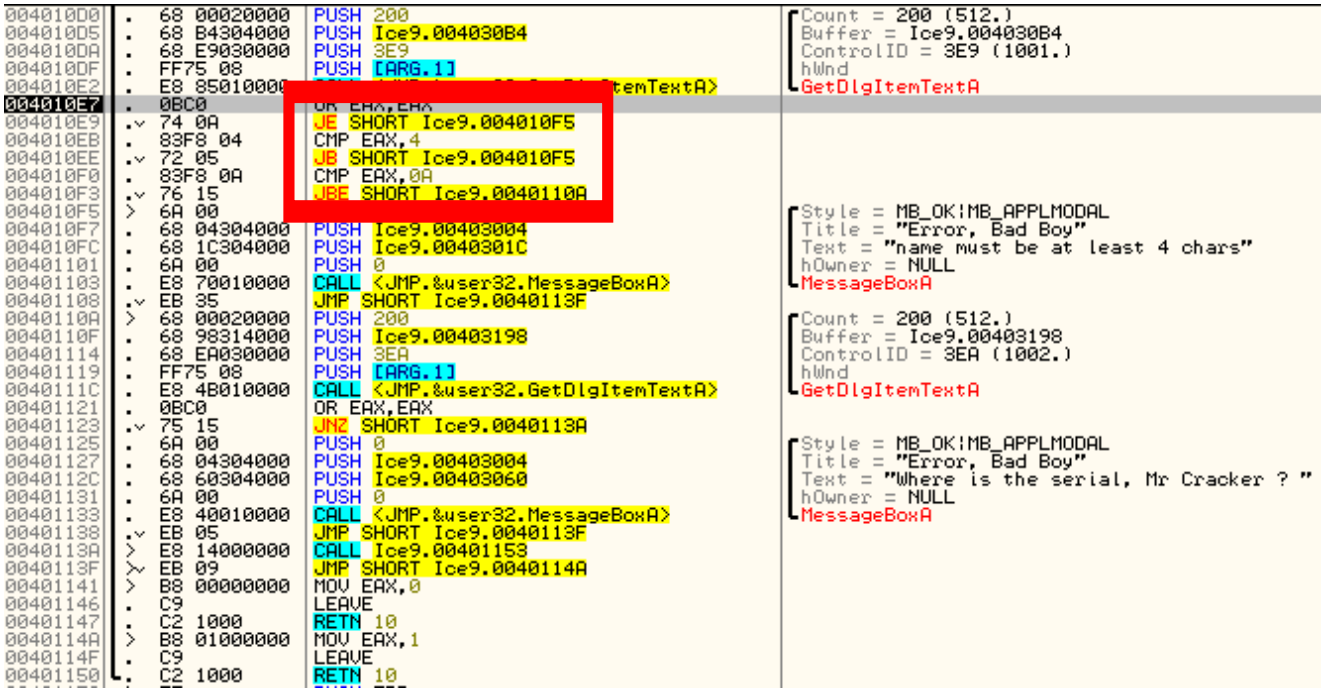
Ejemplo práctico (II): estudio del crackME (2)



Condiciones del nombre

- EAX: longitud del nombre

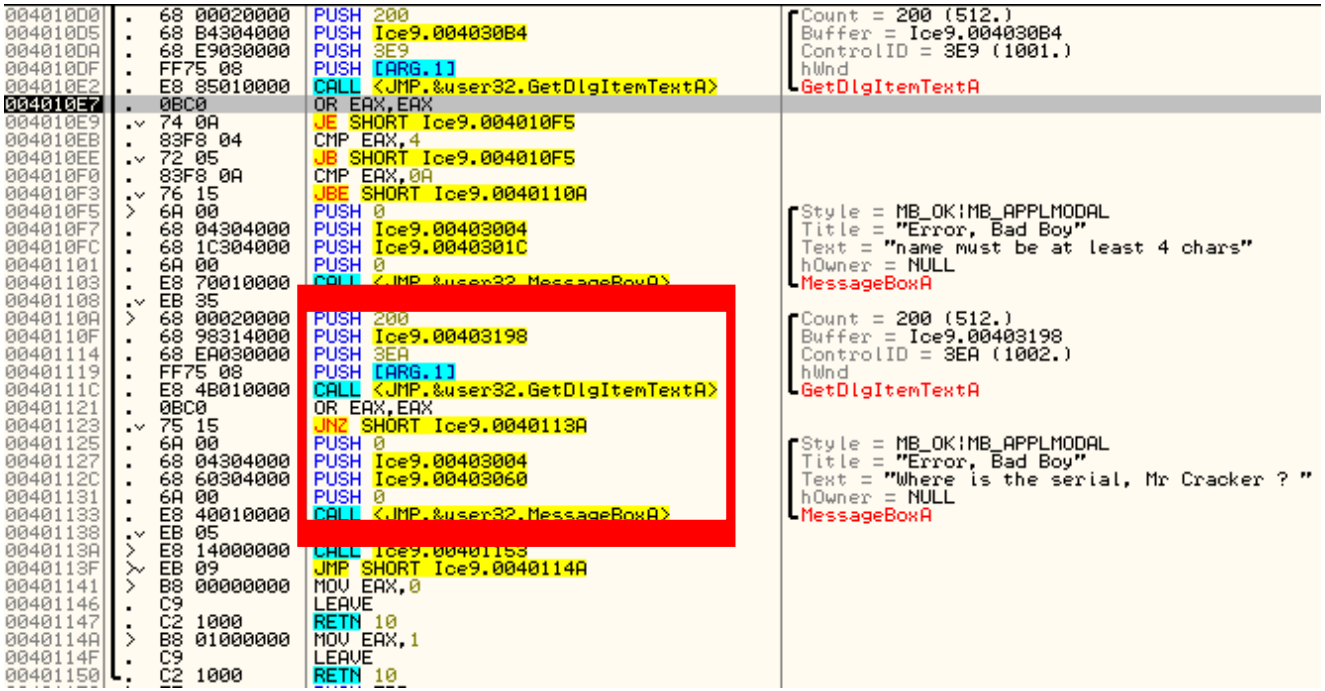
Ejemplo práctico (II): estudio del crackME (2)



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'

Universidad Zaragoza 1542

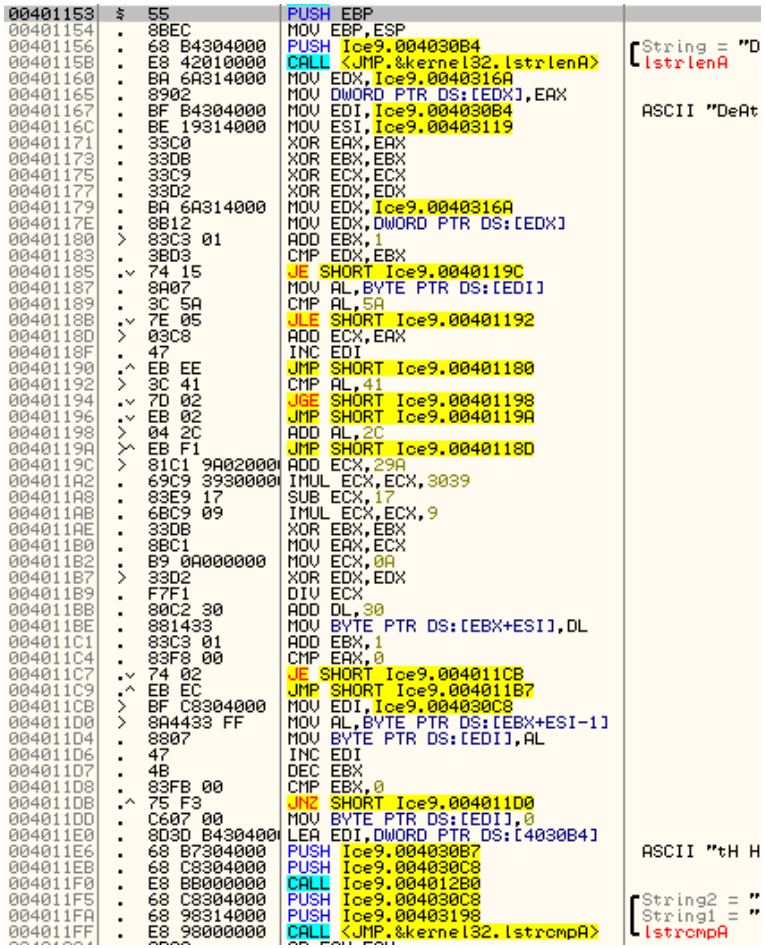
Ejemplo práctico (II): estudio del crackME (3)

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

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

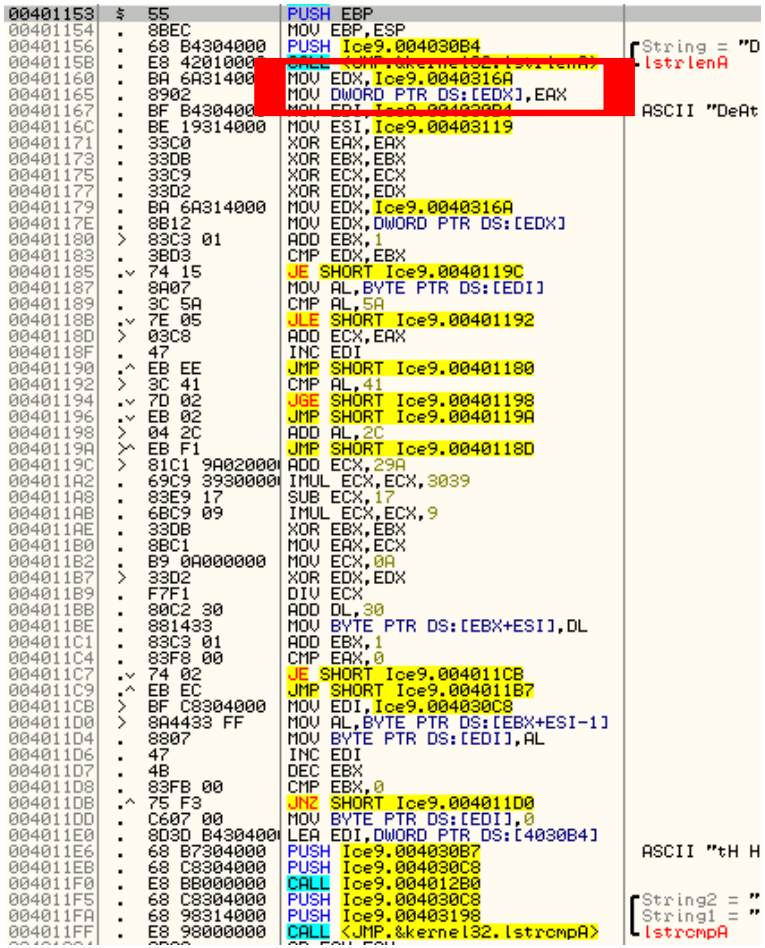
- Búsqueda de cadenas referenciadas
- Cadenas de 'chico bueno' y 'chico malo'
- `lstrcmpA`: comparación de cadenas
  - Valor de EAX determina igualdad

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



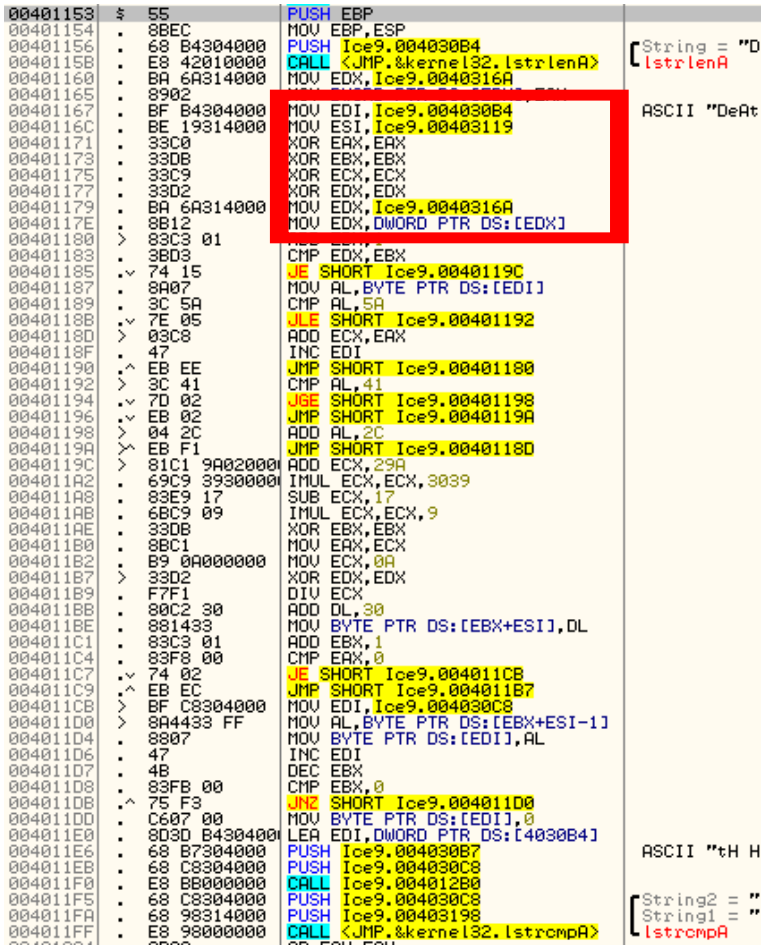
- 0x401153: Inicio algoritmo
- BP para estudio 'en caliente'

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)

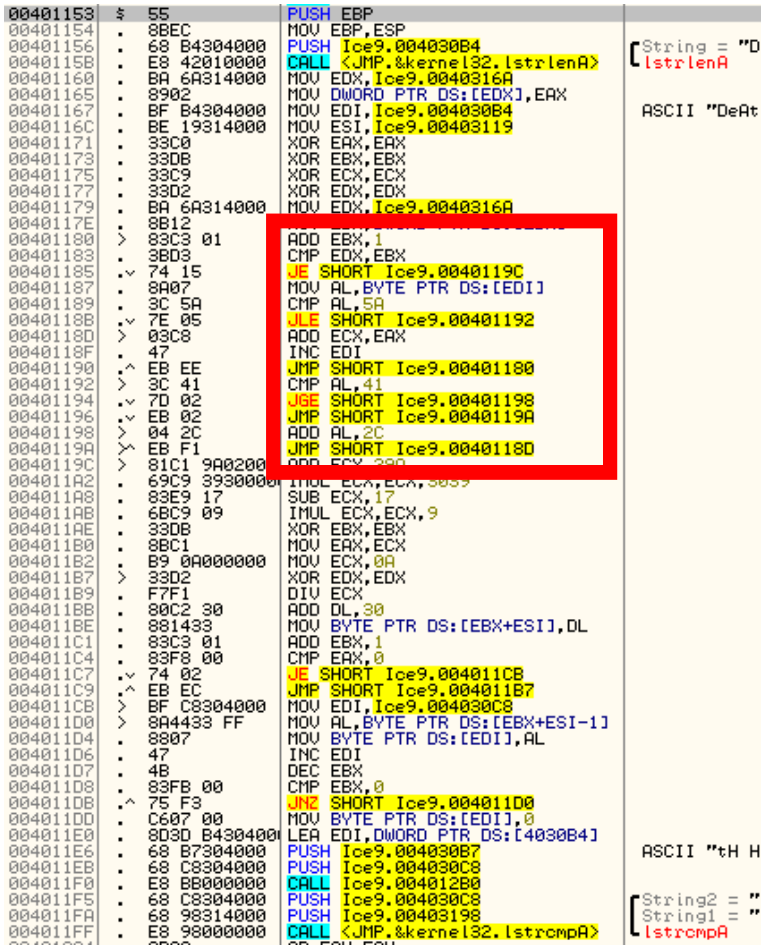


```
00401153  $  55              PUSH EBP
00401154  .  8BEC            MOV EBP,ESP
00401156  .  68 B4304000    PUSH Ice9.004030B4                [String = "D
0040115B  .  E8 42010000    CALL <JMP.&kernel32.lstrlenA>     [lstrlenA
00401160  .  BA 6A314000    MOV EDX,Ice9.0040316A
00401165  .  8902
00401167  .  BF B4304000    MOV EDI,Ice9.004030B4             ASCII "DeAt
0040116C  .  BE 19314000    MOV ESI,Ice9.00403119
00401171  .  33C0           XOR EAX,EAX
00401173  .  33DB           XOR EBX,EBX
00401175  .  33C9           XOR ECX,ECX
00401177  .  33D2           XOR EDX,EDX
00401179  .  BA 6A314000    MOV EDX,Ice9.0040316A
0040117E  .  8B12           MOV EDX,DWORD PTR DS:[EDX]
00401180  >  83C3 01
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
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
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 B430400(  LEA EDI,DWORD PTR DS:[4030B4]
004011E6  .  68 B7304000    PUSH Ice9.004030B7                ASCII "tH H
004011EB  .  68 C8304000    PUSH Ice9.004030C8
004011F0  .  E8 BB000000    CALL Ice9.004012B0
004011F5  .  68 C8304000    PUSH Ice9.004030C8                [String2 = "
004011FA  .  68 98314000    PUSH Ice9.00403198                [String1 = "
004011FF  .  E8 98000000    CALL <JMP.&kernel32.lstrcmpA>     [lstrcmpA
```

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

Universidad Zaragoza 1542

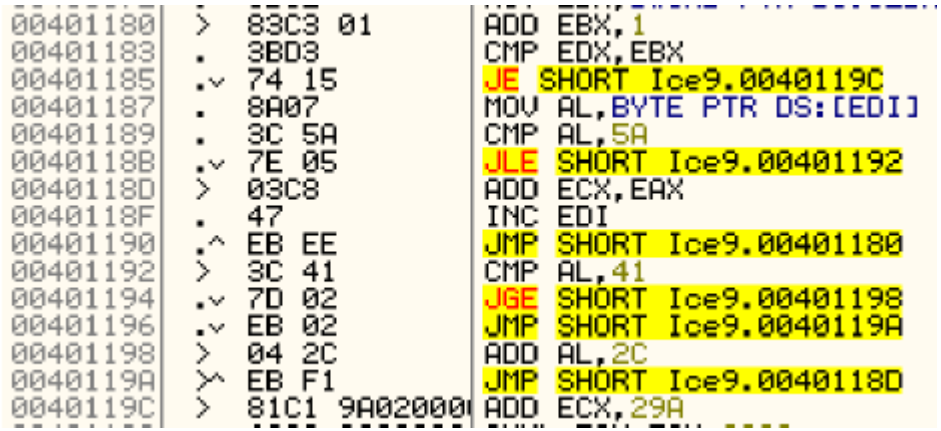
Ejemplo práctico (III): algoritmo de generación (1)



- 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

Universidad Zaragoza 1542

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



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
```

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

Universidad Zaragoza

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

```
0040119C  > 81C1 9A02000( ADD ECX,29A
004011A2  .  69C9 3930000( 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 $EAX = 0$

Universidad Zaragoza 1542

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>
```

Universidad Zaragoza 1542

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

Universidad Zaragoza

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

```
0040119C  > 81C1 9A02000( ADD ECX,29A
004011A2  .  69C9 3930000( 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 4º carácter) al final de la nueva cadena

Universidad Zaragoza 1542

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

Universidad Zaragoza 1542

Universidad Zaragoza 1542

Conclusiones y otras aplicaciones

- 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' :)

Otras aplicaciones

- Fuzzing
- Exploiting
- DBI (Dynamic Binary Instrumentation)

Universidad Zaragoza 1542

Universidad Zaragoza 1542

Investigación en Ingeniería Inversa

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

Universidad Zaragoza
1542

Frameworks DBI (I)

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

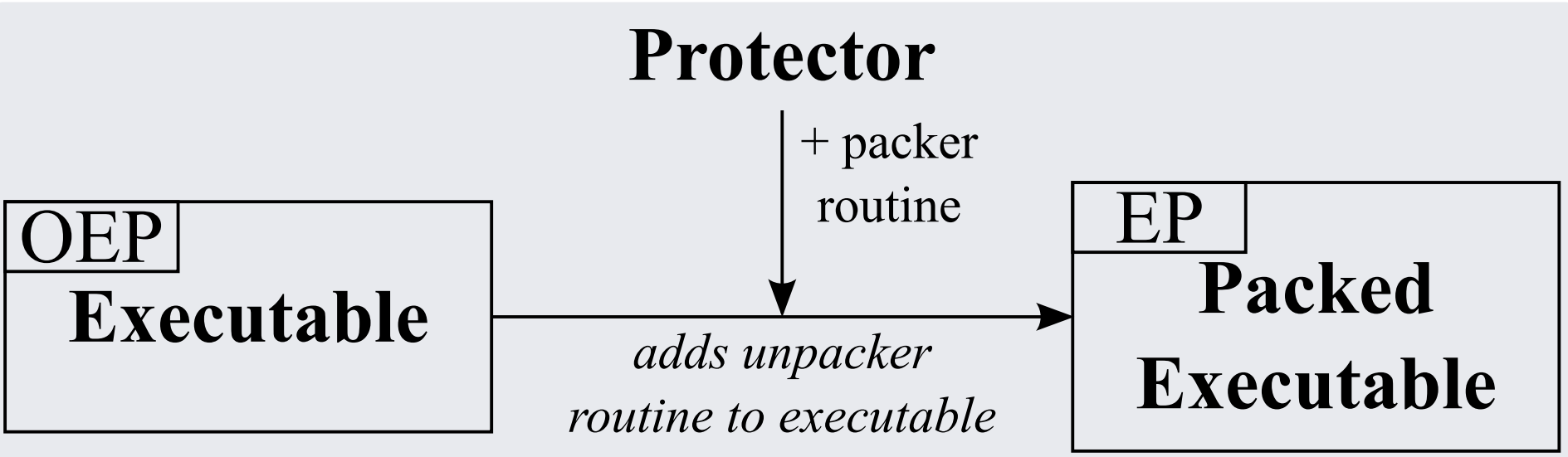
Universidad Zaragoza 1542

Frameworks DBI (II)

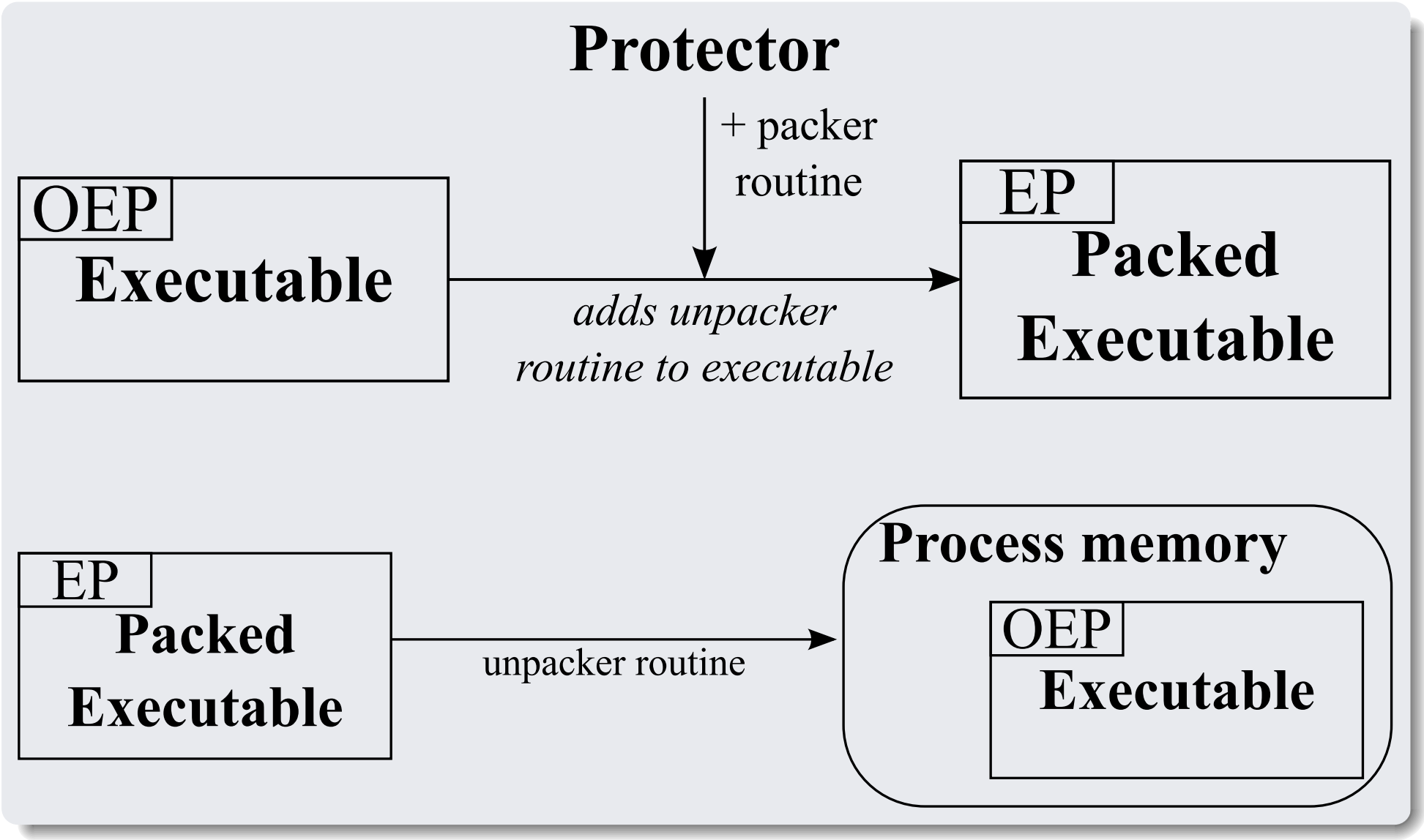
- PFC: Estudio comparativo de frameworks de Instrumentación Dinámica de Ejecutables
- Alguna cosa curiosa:
  - No. instrucciones ejecutadas varía entre frameworks
- Charla aceptada en RootedCON 2012 (y próxima publicación :))
  - Mejora del proceso de desempacado usando DBI
  - Paso imprescindible en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

Universidad Zaragoza 1542

Frameworks DBI (III)



Frameworks DBI (III)



Introducción a la Ingeniería Inversa
Diseño de Aplicaciones Seguras

Ricardo J. Rodríguez
rjrodriguez@unizar.es

Universidad
Zaragoza

19 de Enero de 2012

Grupo de Ingeniería de Sistemas de Eventos Discretos
Universidad de Zaragoza