Mejora en el Proceso de Desempacado usando Técnicas DBI

Ricardo J. Rodríguez

rjrodriguez@unizar.es

tw: @RicardoJRdez – http://www.ricardojrodriguez.es

Universidad Zaragoza

3 de Marzo de 2012

RootedCON 2012

Madrid, Spain

$whoami

- Miembro de CLS desde sus inicios (2000)
- Investigador (PhD candidate) en Universidad de Zaragoza

Líneas de investigación

- Rendimiento de sistemas software complejos
- Ingeniería de Software segura
- Sistemas de Tolerancia a Fallos (diseño y modelado)

$whoami

- Miembro de CLS desde sus inicios (2000)
- Investigador (PhD candidate) en Universidad de Zaragoza

Líneas de investigación

- Rendimiento de sistemas software complejos
- Ingeniería de Software segura
- Sistemas de Tolerancia a Fallos (diseño y modelado)
- Análisis *malware*

Motivación (I)

¿Qué puedo hacer para proteger mi ejecutable de los malos? (1)

- No distribuirlo

Motivación (I)

¿Qué puedo hacer para proteger mi ejecutable de los malos? (1)

- No distribuirlo
  - Where is my fuckin' money, ha?

Motivación (I)

¿Qué puedo hacer para proteger mi ejecutable de los malos? (1)

- No distribuirlo
  - Where is my fuckin' money, ha?
- GPL'd it: spread the love

Motivación (I)

¿Qué puedo hacer para proteger mi ejecutable de los malos? (1)

- No distribuirlo
  - Where is my fuckin' money, ha?
- GPL'd it: spread the love
  - Nos quedaremos sin trabajo :'(

Motivación (I)

¿Qué puedo hacer para proteger mi ejecutable de los malos? (1)

- No distribuirlo
  - Where is my fuckin' money, ha?
- GPL'd it: spread the love
  - Nos quedaremos sin trabajo :'(
- Rezar a [rellenar según creencias]

Motivación (II)

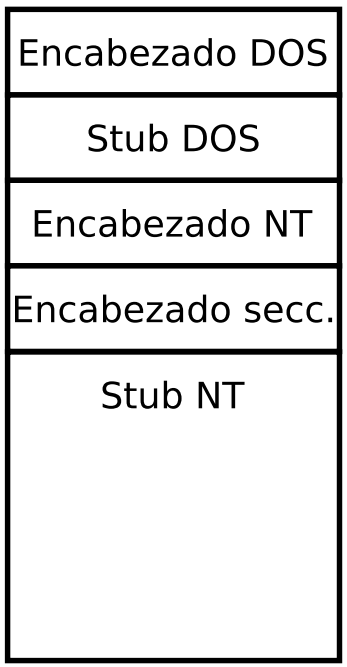
¿Qué puedo hacer para proteger mi ejecutable de los malos? (2)

Protección de ejecutables contra *reversing*

- Código *anti-reversing*
  - *Anti-debugging*
  - *Anti-tracing*
  - Detección de modificaciones (e.g., CRC)
- Uso de protectores software (a.k.a. *packers*)
  - Comprimen un ejecutable ocultando:
    - Código original
    - *Entry Point* (EP): primera instrucción del ejecutable
  - $\surd \uparrow$ Protección vs. $\times \uparrow$ tiempo ejecución/ $\uparrow$ consumo memoria
  - Archivo autoextraíble:
    - Ejecutable (código) real + rutina de descompresión

Motivación (II): conocimientos previos

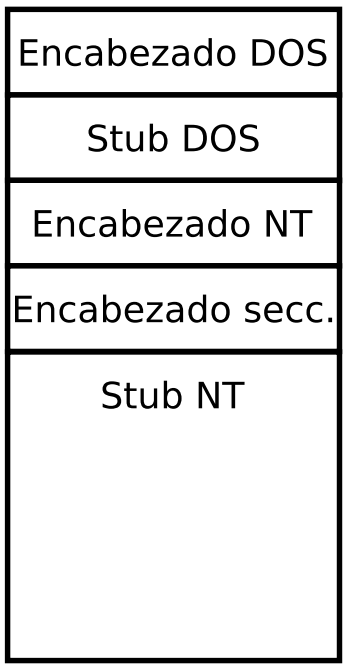
Conocimientos de las estructuras PE (1)



- Encabezados: tamaño constante
- EXEs, DLLs, OBJs
- Cabecera MZ (DOS): código a ejecutar si no es compatible con MS-DOS

Motivación (II): conocimientos previos

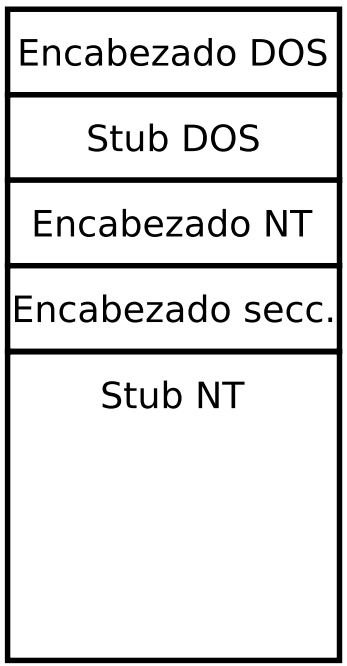
Conocimientos de las estructuras PE (1)



- Encabezados: tamaño constante
- EXEs, DLLs, OBJs
- Cabecera MZ (DOS): código a ejecutar si no es compatible con MS-DOS
  - Mark Zbikowski
  - 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

Motivación (II): conocimientos previos

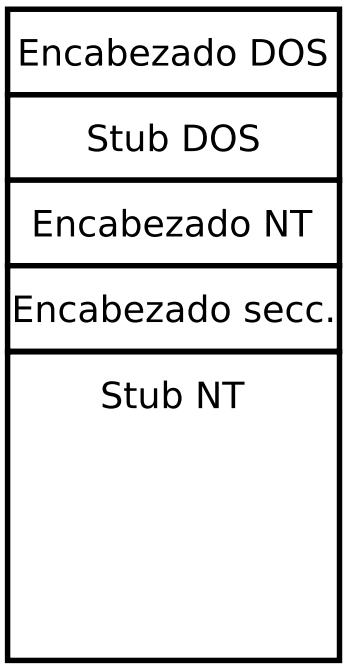
Conocimientos de las estructuras PE (2)



- 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

Motivación (II): conocimientos previos

Conocimientos de las estructuras PE (2)



- 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

Motivación (II): conocimientos previos

Conocimientos de las estructuras PE (3)

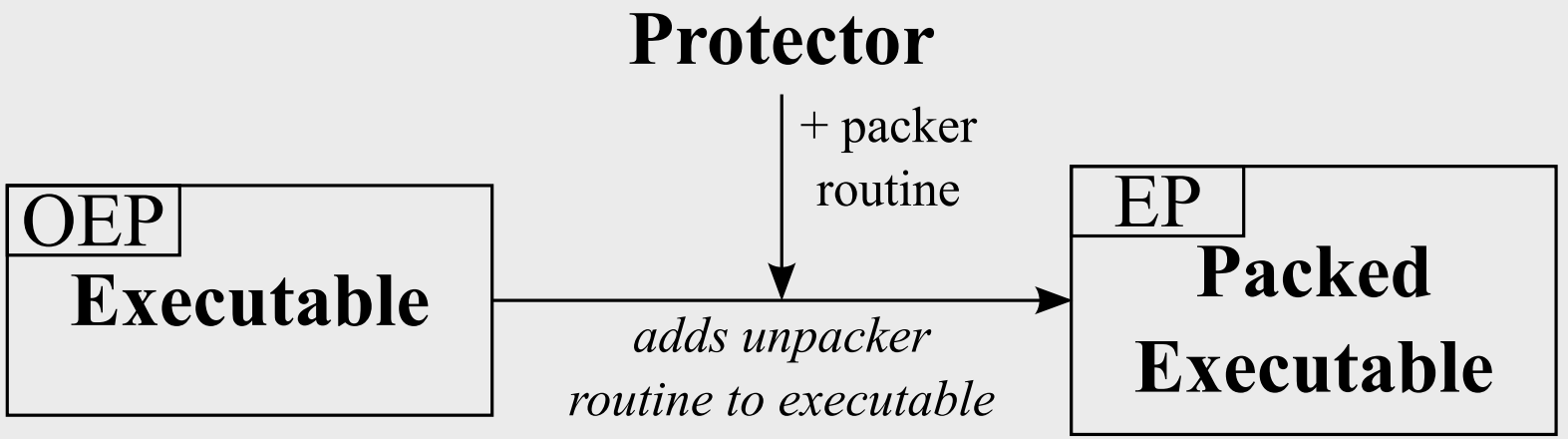
Referencias

- 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)

Motivación (II)

¿Cómo funciona un protector software?

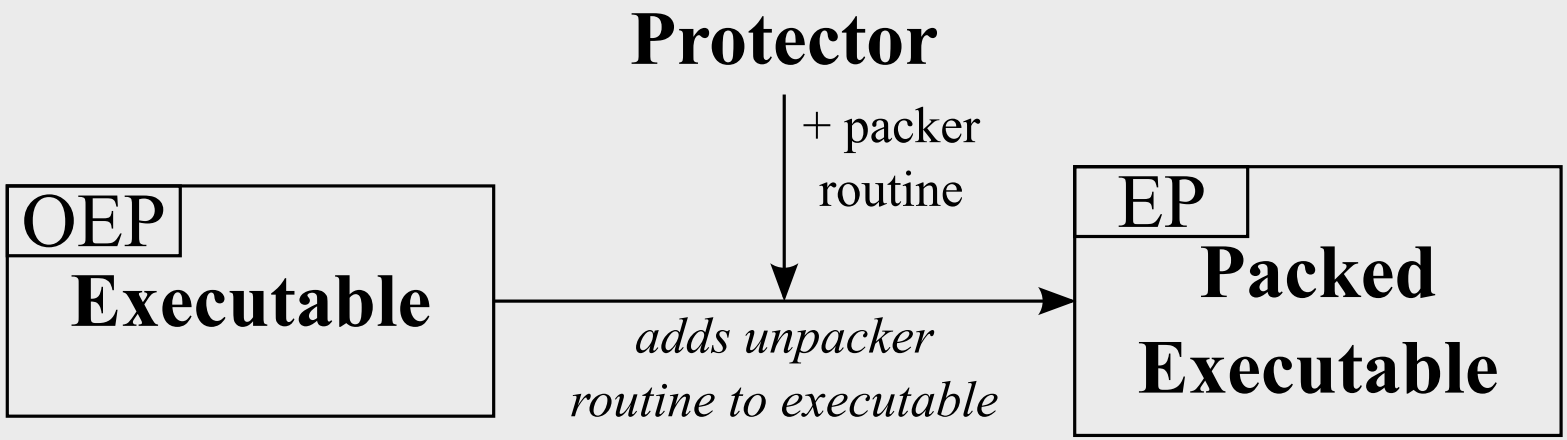
Empacando...



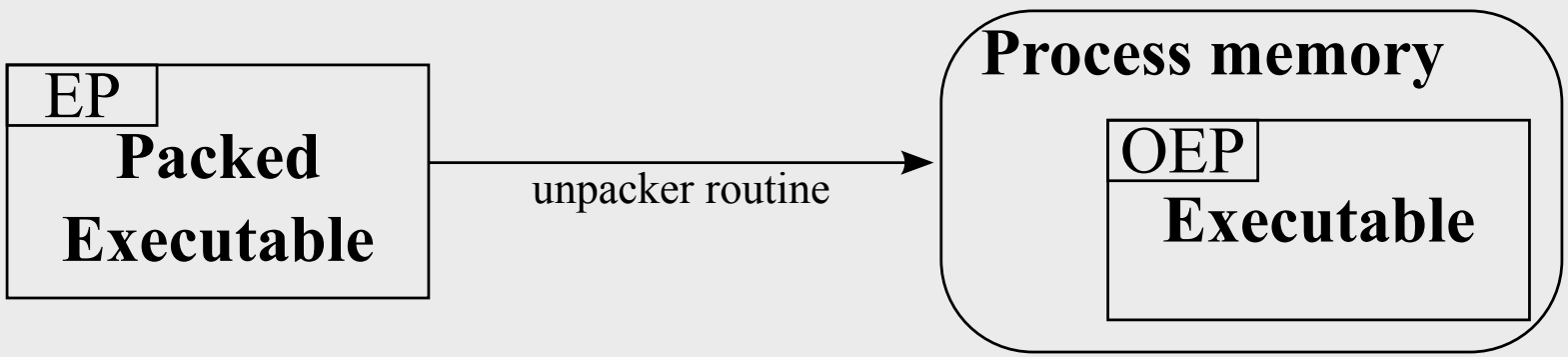
Motivación (II)

¿Cómo funciona un protector software?

Empacando...



Desempacando...



Motivación (III)

¿Cómo se revierte un ejecutable protegido?

Pasos para conseguir un desempacado

1. Identificar protector → metodología

Motivación (III)

¿Cómo se revierte un ejecutable protegido?

Pasos para conseguir un desempacado

1. Identificar protector → metodología
2. Encontrar el *Original Entry Point* (OEP)

Motivación (III)

¿Cómo se revierte un ejecutable protegido?

Pasos para conseguir un desempacado

1. Identificar protector → metodología
2. Encontrar el *Original Entry Point* (OEP)
   - *¿dónde acaba el código de desempacado?*

Motivación (III)

¿Cómo se revierte un ejecutable protegido?

Pasos para conseguir un desempacado

1. Identificar protector → metodología
2. Encontrar el *Original Entry Point* (OEP)
   - *¿dónde acaba el código de desempacado?*
3. Volcar (de memoria del proceso) el ejecutable a disco

Motivación (III)

¿Cómo se revierte un ejecutable protegido?

Pasos para conseguir un desempacado

1. Identificar protector → metodología
2. Encontrar el *Original Entry Point* (OEP)
   - *¿dónde acaba el código de desempacado?*
3. Volcar (de memoria del proceso) el ejecutable a disco
4. Reconstruir ejecutable volcado

Motivación (III)

¿Cómo se revierte un ejecutable protegido?

Pasos para conseguir un desempacado

1. Identificar protector → metodología
2. Encontrar el *Original Entry Point* (OEP)
   - *¿dónde acaba el código de desempacado?*
3. Volcar (de memoria del proceso) el ejecutable a disco
4. Reconstruir ejecutable volcado
   - *Cabecera del ejecutable*

Motivación (III)

¿Cómo se revierte un ejecutable protegido?

Pasos para conseguir un desempacado

1. Identificar protector → metodología
2. Encontrar el *Original Entry Point* (OEP)
   - *¿dónde acaba el código de desempacado?*
3. Volcar (de memoria del proceso) el ejecutable a disco
4. Reconstruir ejecutable volcado
   - *Cabecera del ejecutable*
   - *Tabla IAT (Import Address Table) [APIs]*

Motivación (III)

¿Cómo se revierte un ejecutable protegido?

Pasos para conseguir un desempacado

1. Identificar protector → metodología
2. Encontrar el *Original Entry Point* (OEP)
   - *¿dónde acaba el código de desempacado?*
3. Volcar (de memoria del proceso) el ejecutable a disco
4. Reconstruir ejecutable volcado
   - *Cabecera del ejecutable*
   - *Tabla IAT (Import Address Table) [APIs]*

Motivación (III)

¿Cómo se revierte un ejecutable protegido?

Pasos para conseguir un desempacado

1. Identificar protector → metodología
2. Encontrar el *Original Entry Point* (OEP)
   - *¿dónde acaba el código de desempacado?*
3. Volcar (de memoria del proceso) el ejecutable a disco
4. Reconstruir ejecutable volcado
   - *Cabecera del ejecutable*
   - *Tabla IAT (Import Address Table) [APIs]*

Dificultad en encontrar OEP

- Ofuscación de código del ejecutable protegido
- Código basura (*junk code*)
- Técnicas de ocultación de OEP (*Stolen Bytes*, Asprotect)

Motivación (IV)

Objetivo: encontrar el OEP de una forma rápida

Objetivo

- Mejorar el proceso de desprotección de empacados
- Encontrar OEP rápidamente

Motivación (IV)

Objetivo: encontrar el OEP de una forma rápida

Objetivo

- Mejorar el proceso de desprotección de empacados
- Encontrar OEP rápidamente
  - Entre el 79% y 92% de *malware* distribuido protegido
  - Centrarse en el código del *malware*
- Usando DBI (Dynamic Binary Instrumentation)

Trabajo Relacionado (I)

Quién y qué: desempacado automático

Debugging

- Universal PE Unpacker
  - Plugin para el IDA Pro $\geq$ 4.9
- PolyUnpack
  - Comparación de trazas ejecutable estático y en ejecución
  - *Single-step*

Trabajo Relacionado (II)

Quién y qué: desempacado automático

Emulación o virtualización (hardware)

- Renovo (Kang et al.)
  - Basado en QEMU
  - Traductor dinámico del binario
- Azure
  - Basado en KVM (*Kernel-based Virtual Machine*)
  - *Intel Virtualization Technology* (IVT)
- Pandora's Bosch
  - Basado en Bosch (emulador PC open source). Intérprete
  - Tesis doctoral Lutz Böhne
- *Secure and advanced unpacking using computer emulation* (Sébastien Josse), Journal in Computer Virology, Springer, 2007.

Trabajo Relacionado (III)

Quién y qué: desempacado automático

Windows Driver

- OllyBonE
  - OllyDBG plugin
  - Rompe cuando se provoca fallos de página
- Generic Unpacker
  - Hook en `ntos!SwapContext`
- OmniUnpack
  - Analiza traza de ejecución
  - Kernel driver + user component

Trabajo Relacionado (IV)

Quién y qué: desempacado automático

Otras

- RL!Depacker
- GUnPacker
- Otros específicos para protectores

Instrumentación Dinámica de Ejecutables

- MmmBop (Piotr Bania)
  - Hook en `KiUserExceptionDispatcher` y `NtContinue`
- ParaDyn
  - Usa framework DBI Dyninst
- Saffron (Quist)
  - Usa framework DBI PIN
  - Incorpora gestor de fallos de página (estilo OllyBonE)

Instrumentación Dinámica de Ejecutables (I)

¿WTF es *Instrumentación Dinámica de Ejecutables*?

Qué es

- DBI (*Dynamic Binary Instrumentation*)
- Analiza comportamiento de ejecutable durante ejecución
- Diferente (puede) según entrada
- Análisis estático vs. dinámico

Instrumentación Dinámica de Ejecutables (I)

¿WTF es *Instrumentación Dinámica de Ejecutables*?

Qué es

- DBI (*Dynamic Binary Instrumentation*)
- Analiza comportamiento de ejecutable durante ejecución
- Diferente (puede) según entrada
- Análisis estático vs. dinámico
  - PUEDE ocurrir vs. QUÉ ocurre
- NO SE EJECUTAN todos los posibles caminos (!)

Instrumentación Dinámica de Ejecutables (II)

¿Y cómo funciona la *Instrumentación Dinámica de Ejecutables*?

Cómo funciona

- Instrumentación JIT (Just-in-Time)
- Granularidad
  - Instrucción
  - Bloques (*basic block*, BBL)
  - Rutinas
  - Secciones
  - Imagen
- Dos componentes
  - Mecanismo de instrumentación: dónde y cómo
  - Código a ejecutar en el punto instrumentado
- Soporte de *multithreading*!

Instrumentación Dinámica de Ejecutables (III)

Frameworks DBI

Frameworks DBI: características y tipos

- APIs para desarrollo de herramientas DBA (*Dynamic Binary Analysis*)
- Buena documentación (algunos)
- Multiplataforma (incluso Android!)
- Soporte

Instrumentación Dinámica de Ejecutables (III)

Frameworks DBI

Frameworks DBI: características y tipos

- APIs para desarrollo de herramientas DBA (*Dynamic Binary Analysis*)
- Buena documentación (algunos)
- Multiplataforma (incluso Android!)
- Soporte
- Ejemplos:
  - PIN (de Intel)
  - Valgrind (Universidad de Cambridge)
  - DynamoRIO (HP + MIT)
  - Dyninst (Universidad de Wisconsin–Madison y Universidad de Maryland)

Instrumentación Dinámica de Ejecutables (IV)

Ejemplo de uso: PIN

Ejemplo PIN (en C): conteo de instrucciones

- Contar el número de instrucciones que se ejecutan en un programa*

```
#include "pin.H"

...

int main(int argc, char* argv[])
{
PIN_Init(argc, argv);
INS_AddInstrumentFunction(instrumentationFunc, 0);
PIN_AddFiniFunction(endFunc, 0);
}
```

Instrumentación Dinámica de Ejecutables (IV)

Ejemplo de uso: PIN

Ejemplo PIN (en C): conteo de instrucciones

- Contar el número de instrucciones que se ejecutan en un programa*

```
#include "pin.H"

...

int main(int argc, char* argv[])
{
PIN_Init(argc, argv);
INS_AddInstrumentFunction(instrumentationFunc, 0);
PIN_AddFiniFunction(endFunc, 0);
}
```

```
static UINT64 icount = 0;


void docount(){ icount++;}

void instrumentationFunc(INS ins, void *v)
{
INS_InsertCall(
      ins,
      IPOINT_BEFORE,
      (AFUNPTR)docount,
      IARG_END
      );
}
```

Recordatorio: hacer demos ejemplo.

Instrumentación Dinámica de Ejecutables (IV)

Ejemplo de uso: PIN

Ejemplo PIN (en C): conteo de instrucciones

- Contar el número de instrucciones que se ejecutan en un programa*

```
#include "pin.H"

...

int main(int argc, char* argv[])
{
PIN_Init(argc, argv);
INS_AddInstrumentFunction(instrumentationFunc, 0);
PIN_AddFiniFunction(endFunc, 0);
}
```

```
static UINT64 icount = 0;

void docount(){ icount++;}

void instrumentationFunc(INS ins, void *v)
{
INS_InsertCall(
      ins,
      IPOINT_BEFORE,
      (AFUNPTR)docount,
      IARG_END
      );
}
```

Recordatorio: hacer demos ejemplo.

Recordatorio: ¿has hablado algo de las diferencias de instrucciones?

Herramienta DBA para encontrar OEP (I)

En busca del OEP perdido

Idea (no nueva :()

- Usar DBI para encontrar el OEP

¿Cómo?

- Buscar @ins a ejecutar...

Herramienta DBA para encontrar OEP (I)

En busca del OEP perdido

Idea (no nueva :()

- Usar DBI para encontrar el OEP

¿Cómo?

- Buscar @ins a ejecutar...
- ...que antes haya sido escrita

Herramienta DBA para encontrar OEP (I)

En busca del OEP perdido

Idea (no nueva :()

- Usar DBI para encontrar el OEP

¿Cómo?

- Buscar @ins a ejecutar...
- ...que antes haya sido escrita
- Más informalmente:
  - Guardar histórico de @s memoria escritas
  - Chequear toda ejecutada con histórico

Herramienta DBA para encontrar OEP (I)

En busca del OEP perdido

Idea (no nueva :()

- Usar DBI para encontrar el OEP

¿Cómo?

- Buscar @ins a ejecutar...
- ...que antes haya sido escrita
- Más informalmente:
  - Guardar histórico de @s memoria escritas
  - Chequear toda ejecutada con histórico
  - Primera coincidencia → ¿bingo? (o línea, veremos...)

Herramienta DBA para encontrar OEP (II)

Pseudocódigo: jugada de bingo

booleano candidato = **falso**

...

procedimiento acciónAEjecutar(INS instrucción)

principio

si (*número de operandos en memoria de instrucción*) ≥ 1

$\wedge$ (*es instrucción de escritura*)

$\wedge$ ($\neg$ *candidato*))

entonces

Añadir instrucción a candidatas

si_no si ($\neg$ candidato)

Buscar @instrucción en candidatas

Si hay coincidencia $\rightarrow$ *instrucción es candidata*

candidato = **verdadero**

finSi

fin

Herramienta DBA para encontrar OEP (II)

Pseudocódigo: jugada de línea

...
procedimiento acciónAEjecutar(INS instrucción)
principio
  si (*número de operandos en memoria de instrucción*) ≥ 1
    $\wedge$ (*es instrucción de escritura*)
)
  entonces
    Añadir instrucción a candidatas
  si_no
    Buscar @instrucción en candidatas
    Si hay coincidencia $\rightarrow$ *instrucción es candidata*
  finSi
fin

Experimentos (I)

Programa de prueba

- Aplicación *UnpackME*
- Desarrollada en ASM 32bits

Experimentos (I)

Programa de prueba

- Aplicación *UnpackME*
- Desarrollada en ASM 32bits
- EP: `0x401000`

`Recordatorio: mostrar características (PEiD).`

Experimentos (I)

Programa de prueba

- Aplicación *UnpackME*
- Desarrollada en ASM 32bits
- EP: `0x401000`

`Recordatorio: mostrar características (PEiD).`

Proteger con diferentes packers... y probar!

Experimentos (II)

Algunos resultados

Packer	Version	¿Cantamos bingo?
UPX	3.0.8w	$\surd$

Experimentos (II)

Algunos resultados

Packer	Version	¿Cantamos bingo?
UPX	3.0.8w	$\surd$
FSG	2.0	$\surd$

Experimentos (II)

Algunos resultados

Packer	Version	¿Cantamos bingo?
UPX	3.0.8w	$\surd$
FSG	2.0	$\surd$
NeoLite	2.0	$\surd$

Experimentos (II)

Algunos resultados

Packer	Version	¿Cantamos bingo?
UPX	3.0.8w	$\surd$
FSG	2.0	$\surd$
NeoLite	2.0	$\surd$
XComp	0.98	$\surd$

Experimentos (II)

Algunos resultados

Packer	Version	¿Cantamos bingo?
UPX	3.0.8w	$\surd$
FSG	2.0	$\surd$
NeoLite	2.0	$\surd$
XComp	0.98	$\surd$
ASPack	2.28	$\surd$

Experimentos (II)

Algunos resultados

Packer	Version	¿Cantamos bingo?
UPX	3.0.8w	$\surd$
FSG	2.0	$\surd$
NeoLite	2.0	$\surd$
XComp	0.98	$\surd$
ASPack	2.28	$\surd$
PECompact	1.56	$\times *$

Experimentos (II)

Algunos resultados

Packer	Version	¿Cantamos bingo?
UPX	3.0.8w	$\surd$
FSG	2.0	$\surd$
NeoLite	2.0	$\surd$
XComp	0.98	$\surd$
ASPack	2.28	$\surd$
PECompact	1.56	$\times *$
Petite	2.3	$\times * *$

Experimentos (II)

Algunos resultados

Packer	Version	¿Cantamos bingo?
UPX	3.0.8w	$\surd$
FSG	2.0	$\surd$
NeoLite	2.0	$\surd$
XComp	0.98	$\surd$
ASPack	2.28	$\surd$
PECompact	1.56	$\times *$
Petite	2.3	$\times **$
PESpin	1.33	(rompe)

`Recordatorio: hacer un par de demos de "desempacado automático".`

Experimentos (II)

Algunos resultados

Packer	Version	¿Cantamos bingo?
UPX	3.0.8w	$\surd$
FSG	2.0	$\surd$
NeoLite	2.0	$\surd$
XComp	0.98	$\surd$
ASPack	2.28	$\surd$
PECompact	1.56	$\times *$
Petite	2.3	$\times **$
PESpin	1.33	(rompe)

Recordatorio: hacer un par de demos de "desempacado automático".

Recordatorio: enseña el CExe, pa' reírnos un rato.

Experimentos (III)

Discusión de los resultados

Algunos problemas encontrados

- Problema de tamaño mínimo ejecutable a proteger
  - ¿Cifrado de bloque con tamaño mínimo? ¿padding?
- Análisis de OEP candidatos (según packer)

Experimentos (III)

Discusión de los resultados

Algunos problemas encontrados

- Problema de tamaño mínimo ejecutable a proteger
  - ¿Cifrado de bloque con tamaño mínimo? ¿padding?
- Análisis de OEP candidatos (según packer)
  - ¿Meterle más lógica a la DBA?

Experimentos (III)

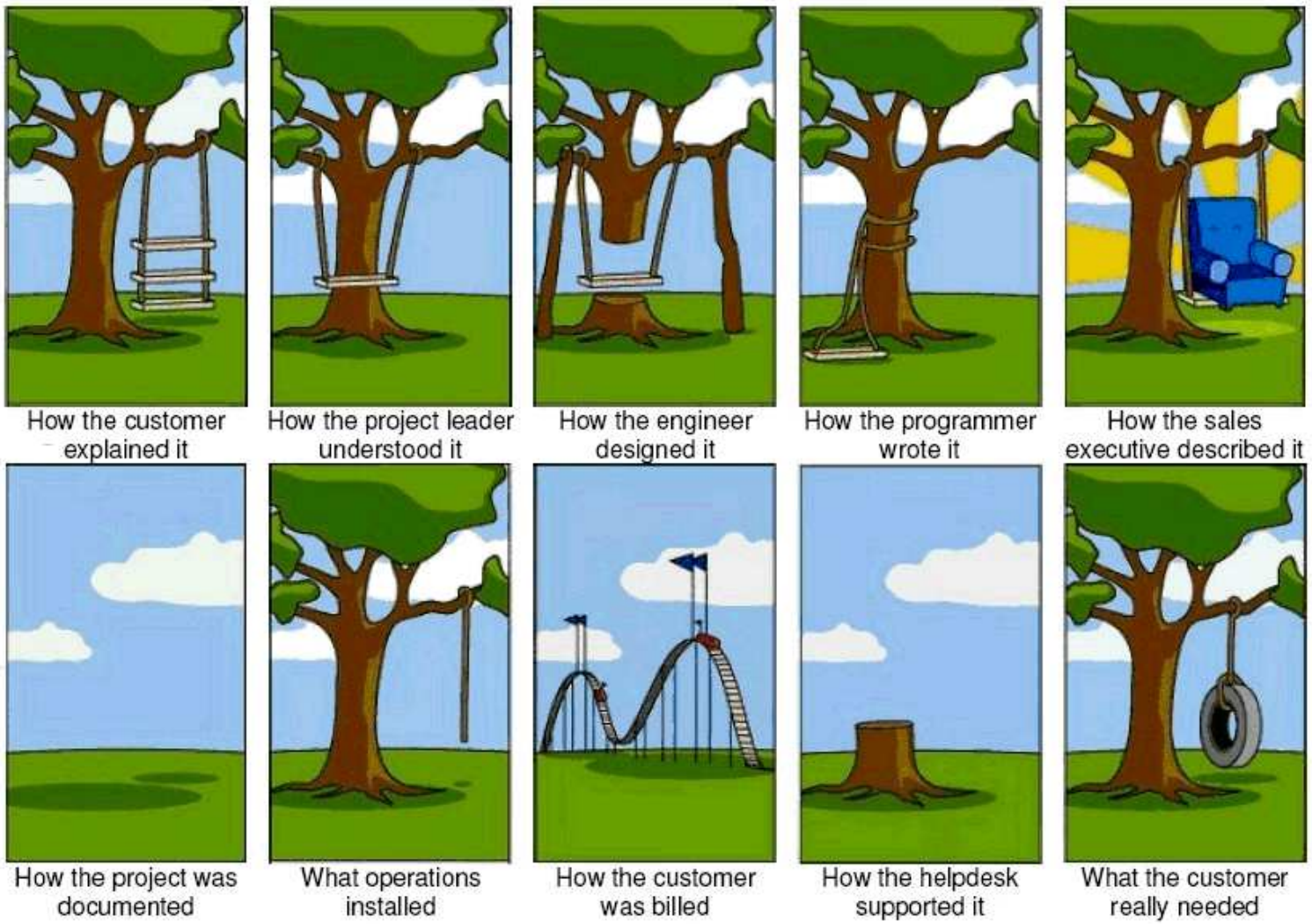
Discusión de los resultados

Algunos problemas encontrados

- Problema de tamaño mínimo ejecutable a proteger
  - ¿Cifrado de bloque con tamaño mínimo? ¿padding?
- Análisis de OEP candidatos (según packer)
  - ¿Meterle más lógica a la DBA?
- Algunos packers (ASProtect, Petite): ejecución entremezclada
  - ¿Usar secciones del ejecutable como límites?
  - Algunos se "incrustran" en los huecos vacíos del código real

Experimentos (IV)

Qué quería yo, y qué hemos conseguido. . .



Conclusiones (I)

Algunas conclusiones

- Frameworks DBI: rápido y sencillo → alto potencial
- NO es necesario conocimientos de programación S.O. avanzados
  - Podemos centrarnos en la "miga": la DBA
- Desventajas:
  - Conocimiento de la API
  - Tiempo de ejecución

Conclusiones (II)

Trabajo futuro

Conclusiones (II)

Trabajo futuro

- Construir herramienta DBA (y GPL) para desempacado genérico

Conclusiones (II)

Trabajo futuro

- Construir herramienta DBA (y GPL) para desempacado genérico
- ¿Cómo se comporta DBI con técnicas anti-debugging de los ejecutables? (e.g., SEH, generación de excepciones. . .)
  - ¿Y con las de los packers?

Conclusiones (II)

Trabajo futuro

- Construir herramienta DBA (y GPL) para desempacado genérico
- ¿Cómo se comporta DBI con técnicas anti-debugging de los ejecutables? (e.g., SEH, generación de excepciones. . .)
  - ¿Y con las de los packers?
- ¿Y si me roban los primeros bytes? :)

Conclusiones (II)

Trabajo futuro

- Construir herramienta DBA (y GPL) para desempacado genérico
- ¿Cómo se comporta DBI con técnicas anti-debugging de los ejecutables? (e.g., SEH, generación de excepciones. . .)
  - ¿Y con las de los packers?
- ¿Y si me roban los primeros bytes? :)
- ¿Y con código polimórfico?

Conclusiones (II)

Trabajo futuro

- Construir herramienta DBA (y GPL) para desempacado genérico
- ¿Cómo se comporta DBI con técnicas anti-debugging de los ejecutables? (e.g., SEH, generación de excepciones. . .)
  - ¿Y con las de los packers?
- ¿Y si me roban los primeros bytes? :)
- ¿Y con código polimórfico?
- ¿Y con packers emuladores (e.g. Themida, ExeCryptor)?

(Más) aplicaciones a Ingeniería Inversa

Conclusiones (II)

Trabajo futuro

- Construir herramienta DBA (y GPL) para desempacado genérico
- ¿Cómo se comporta DBI con técnicas anti-debugging de los ejecutables? (e.g., SEH, generación de excepciones. . .)
  - ¿Y con las de los packers?
- ¿Y si me roban los primeros bytes? :)
- ¿Y con código polimórfico?
- ¿Y con packers emuladores (e.g. Themida, ExeCryptor)?

(Más) aplicaciones a Ingeniería Inversa

- Detección de acceso a variables “protegidas”

Conclusiones (II)

Trabajo futuro

- Construir herramienta DBA (y GPL) para desempacado genérico
- ¿Cómo se comporta DBI con técnicas anti-debugging de los ejecutables? (e.g., SEH, generación de excepciones. . .)
  - ¿Y con las de los packers?
- ¿Y si me roban los primeros bytes? :)
- ¿Y con código polimórfico?
- ¿Y con packers emuladores (e.g. Themida, ExeCryptor)?

(Más) aplicaciones a Ingeniería Inversa

- Detección de acceso a variables "protegidas"
- Extracción de direcciones dinámicas

Conclusiones (II)

Trabajo futuro

- Construir herramienta DBA (y GPL) para desempacado genérico
- ¿Cómo se comporta DBI con técnicas anti-debugging de los ejecutables? (e.g., SEH, generación de excepciones. . .)
  - ¿Y con las de los packers?
- ¿Y si me roban los primeros bytes? :)
- ¿Y con código polimórfico?
- ¿Y con packers emuladores (e.g. Themida, ExeCryptor)?

(Más) aplicaciones a Ingeniería Inversa

- Detección de acceso a variables "protegidas"
- Extracción de direcciones dinámicas
- . . .

Agradecimientos y referencias

Referencias

- IndefiniteStudies
- Piotr Bania
- Gente de CLS

Agradecimientos y referencias

Referencias

- IndefiniteStudies
- Piotr Bania
- Gente de CLS
- Asociación RootedCON

Agradecimientos y referencias

Referencias

- IndefiniteStudies
- Piotr Bania
- Gente de CLS
- Asociación RootedCON

¡A trabajar!

→ Gracias por escucharme!

⇉ Nos vemos en la NcN! :)

Mejora en el Proceso de Desempacado usando Técnicas DBI

Ricardo J. Rodríguez

rjrodriguez@unizar.es

tw: @RicardoJRdez – http://www.ricardojrodriguez.es

Universidad Zaragoza

3 de Marzo de 2012

RootedCON 2012

Madrid, Spain