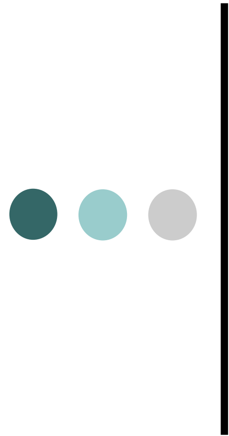


Hacking Memory
Introducción al Shellcode

Jeimy J. Cano, Ph.D, CFE
Universidad de los Andes

jcano@uniandes.edu.co

JCM-05 All rights reserved.

Agenda

- Introducción
- Conceptos Básicos
  - Administración de memoria
  - System Call
  - String format
- Stack Overflows
- Heap Overflows
- Shellcode
  - Conceptos básicos
  - Esqueleto de un código shell
  - Ejemplos de Shellcodes
    - Ejemplo en Sistemas Windows
    - Ejemplo en Sistemas Linux

JCM-05 All rights reserved.

Agenda

- Conceptos Avanzados de *hacking memory*
  - Kernel Overflows
  - Ingeniería Inversa
- ¿Técnicas Anti-reversing?
- Reflexiones finales
- Referencias

JCM-05 All rights reserved.

Introducción

- El manejo de la memoria en las aplicaciones actuales, en el entorno de ejecución, NO se hace de manera adecuada.
- La manipulación de las zonas de memoria en la ejecución de las aplicaciones genera efectos de borde inesperados
- La memoria que manejan las aplicaciones es tan vulnerable como el código mismo que las integra.
- La memoria es un narrador invisible de la ejecución de las aplicaciones
- Reconocer la administración de memoria del SO, es un factor crítico en el descubrimiento de vulnerabilidades de las aplicaciones.

JCM-05 All rights reserved.

Conceptos Básicos

➢ Administración de Memoria

Segmento	Descripción
	0xBFFFFFFF – High Address
Env pointer argc	
Stack Local variables, function calls ↓	LIFO Structure – Last In First Out Almacenamiento por *cortos* periodos de tiempo
↑ **Heap** Dynamic variables	FIFO Structure – First In First Out Almacenamiento por *largos* periodos de tiempo
BSS – Block Started by Symbol Uninitialized Data	Segmento de donde es posible la Escritura
Data Initialized Data	Segmento de donde es posible la Escritura
Text Compiled program code	Segmento de Sólo Lectura
Librerías Compartidas	
	0x8000000 – Low Address

BSS y Data: Reservado para variables Globales

JCM-05 All rights reserved.

Conceptos Básicos

- Registros

0xBFFFFFFF – High Address

Memory (stack side)
Env pointer argc
Stack Local variables, function calls ↓
Function () Local variables
↑ **Heap** Dynamic variables
Librerías Compartidas

0x8000000 – Low Address

Registers
EIP
EBP
ESP

Text segment
Function () In compiled code
Text Compiled program code

BSS – Block Started by Symbol
Uninitialized Data

Data
Initialized Data

JCM-05 All rights reserved.

Conceptos Básicos

System Call

- Es un conjunto de funciones que permiten a un tercero acceso a funciones específicas del sistema operacional como:
  - Obtener entradas
  - Producir salidas
  - Terminar procesos
  - Ejecutar binarios
- Permiten acceso directo al Kernel, en el cual se tiene acceso a funciones de bajo nivel.
- Interfases entre modo kernel y el modo usuario
- Están intimamente a funciones básicas como read, write, fetch, primitivas del sistema operacional
- En Linux – Interrupciones de Software – Int 0x80

JCM-05 All rights reserved.

Conceptos Básicos

String Format

- Son cadenas alfanuméricas que contienen símbolos especiales reconocidos por la función printf() y sus derivadas (sprintf(), fprint(), vprintf()).
- Con ellas es posible especificar la manera como serán visualizados los demás argumentos entregados a la función.
- Especificaciones de formato:
  - %d – presentado como número entero – Como valor
  - %u – presentado como número natural – Como valor
  - %x – presentado como número natural hexadecimal – Como valor
  - %s – presentado como cadena alfanumérica – Como referencia
  - %n – Número de caracteres impresos hasta el momento – Como referencia.

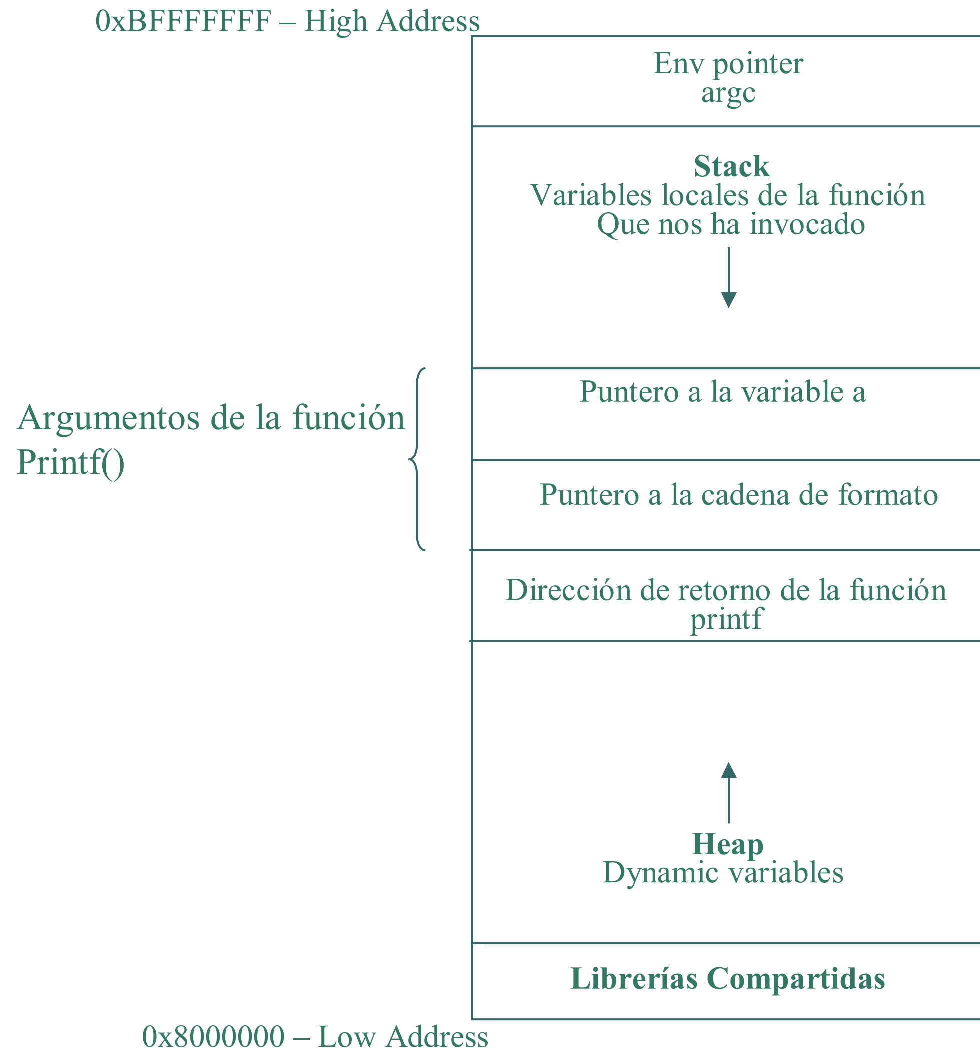
JCM-05 All rights reserved.

Conceptos Básicos

String Format - 2

- Ejemplo básico

```
Int main() {
 char *a = "Hola Mundo";
Printf("Mensaje a presentar: %s \n", a);
}
```



JCM-05 All rights reserved.

Conceptos Básicos

- String Format - 3

  - Utilizando especificaciones de formato

prog1

```
Int main(int argc, char **argv) {
 char f[256];
Strcpy (f, argv[1]);
Printf(f);
}
```

1

```
$ ./prog1 ‘ %x-%x-%x'
Bffffc4f-0-0
-------------------------------------------------
```

2

```
$ ./prog1 ' %n %n %n %n '
Segmentation fault
```

1. Se le pasa como parámetro al programa, una instrucción en comillas inversas, lo que sugiere la ejecución del programa y lo que se encuentra entre comillas, tomando una palabra de 4 bytes de la pila (pues no había argumento), y así sucesivamente

2. El programa ha tratado de escribir a una dirección de memoria al azar la cantidad de caracteres impresos.

Tomado de: SOBOLEWSKI, P. y NIDECKI, T. (2004) Abusos de las cadenas de formato. *Hakin9*. No.5. Noviembre/Diciembre.

JCM-05 All rights reserved.

Stack Overflows

- Características básicas
  - Generalmente se dan por manipulación o mal uso de *buffers*
  - Un *buffer* es un conjunto contiguo de espacio de memoria.

- Instrucciones de manejo del Stack
  - PUSH – Instrucción utilizada para colocar datos en el stack
  - POP – Instrucción para remover datos del Stack
  - ESP – Registro que define el límite del Stack, es decir apunta a la parte superior del Stack

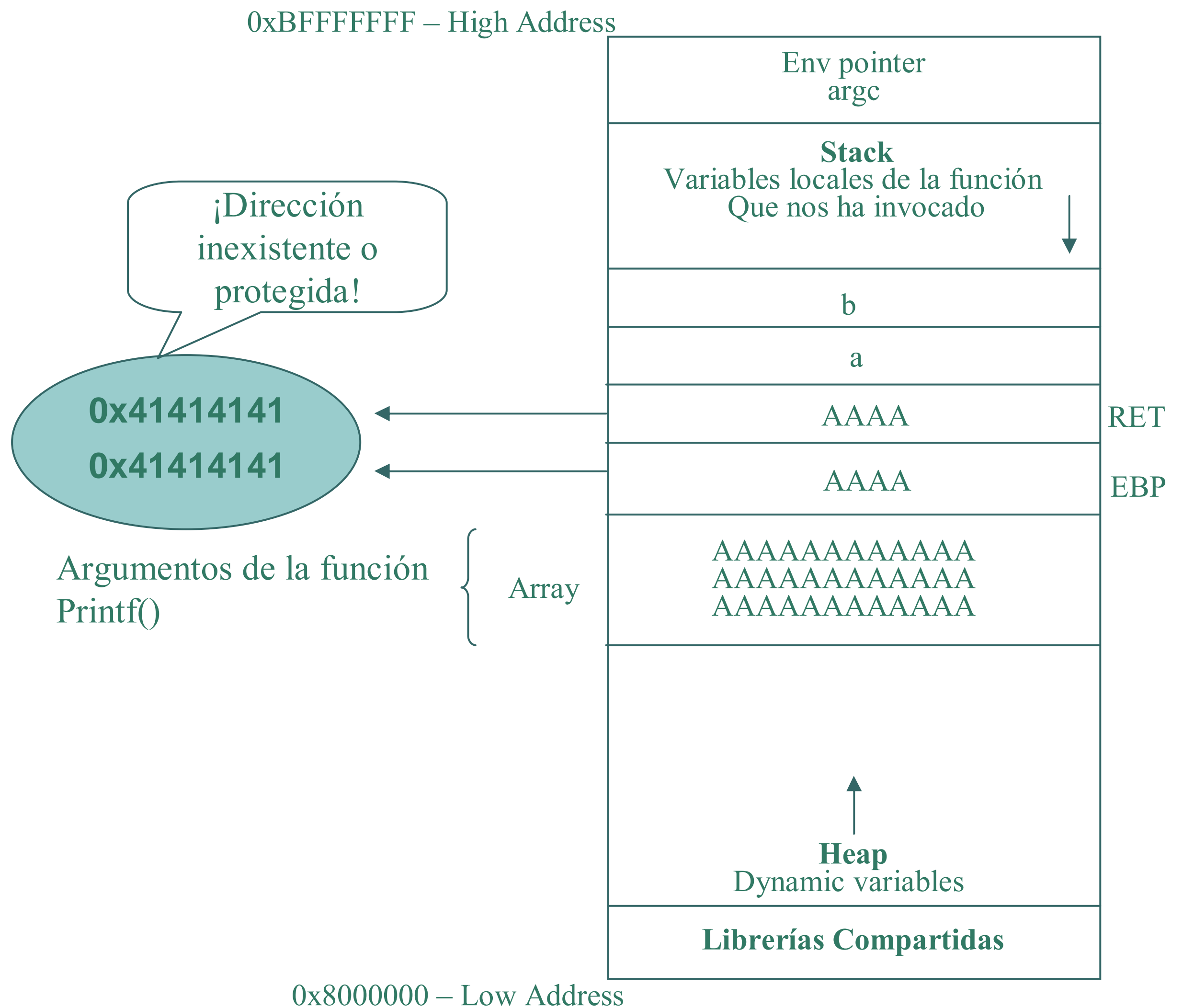
JCM-05 All rights reserved.

Stack Overflows

➢ Ejemplo

```
Void return_input (int a, int b) {
    char array [30];
     gets (array);
     printf (" %s \n", array)
}

main( ) {
 return_input (1,2 );
return 0;
}
```



JCM-05 All rights reserved.

Heap Overflows

- Características básicas
  - La zona de heap, es aquel lugar de memoria para manejo de variables que permanecen por largo tiempo, las cuales continuan existiendo aún la función haya retornado (y sus variables hayan sido elimadas del stack)
  - Funciones que manejan este segmento – malloc () y free (), los cuales se apoyan en brk() o mmap() (esta dos últimas system call).

- Consideraciones de funcionamiento
  - Cuando se usa malloc(), brk() asigna un bloque amplio de memoria en pedazos, el cual entrega al usuario según su solicitud.
  - Cuando se usa free(), decide sobre el nuevo pedazo de memoria liberado, para armar un bloque mas grande disponible.

JCM-05 All rights reserved.

Heap Overflows

➢ Ejemplo

```
int main( int argc, char** argv) {
    char *buf;
    char *buf2;
buf = (char*)malloc(1024);
buf2= (char*)malloc(1024);
printf ("buf=%p buf2=%p\n", buf, buf2);
strcpy (buf, argv[1]);
free(buf2)
}
```

```
$ ./prog1 AAAAAAA … (5000 veces)
malloc(1024)=0x080495b0
malloc(1024)=0x080499b8
printf("buf=%p buf2=%p",(…)buf= 0x080495b0
buf2=0x080499b8)=29
strcpy (0x080495b0, "AAAAA…." ) = 0x080495b0
free (0x080499b8) = <void>
Segmentation fault
```

1. Existen dos buffers contiguos defindos. El primero sobrepasa su tamaño y afecta al segundo. Particularmente sobre escribe la estructura de metadatos del segundo. Luego cuando se intenta liberar el segundo buffer posiblemente se esta teniendo acceso a un segmento de memoria inválido.

KOZIOL, J., AITEL, D., LITCHFIELD, D., ANLEY, C., EREN, S., MEHTA, N. y HASSELL, R. (2004) The shellcoder's handbook: discovering and exploiting security holes. John Wiley & Sons

JCM-05 All rights reserved.

Shellcodes

- Características básicas
  - Una pieza de código de máquina ejecutable o código script que tiene una sola misión: abrir un intérprete de comandos en el sistema objetivo.
  - Generalmente se crean en código ensamblador

- Consideraciones de funcionamiento
  - Los shellcodes son generalmente utilizados para explotar vulnerabilidades, basado en una técnica de vulneración como son :
    - Stack buffer overflows
    - Heap buffer overflows
    - Formar string
    - Integer overflow
    - Memory corruption, etc.
  - Utilizan el contenido de la memoria (Payload)

JCM-05 All rights reserved.

Shellcodes

- Esqueleto base de un código Shell
  - Obtener el EIP
    - Dirección base. Cualquier variable o función en el código shell será relativa a esta dirección. Para obtener esta dirección utilizamos las funciones CALL y POP.
  - Decodificar
    - Generalmente encontramos en memoria caracteres NULL, los cuales nos impiden ejecutar nuestro código. Lo que implica que debamos codificar nuestro código generalmente con XOR con un valor predifinido
  - Obtener las direcciones de las funciones requeridas
    - Identificar en memoria los API, para identificar la dirección de los procesos que requerimos.
  - Configurar el Socket de conexión
    - Generalmente se requiere ubicar donde estan (en memoria) las funciones de socket(), bind(), listen() y accept() o sus equivalentes.
  - Creación del shell

JCM-05 All rights reserved.

Shellcodes en Windows

- Ejemplo

Definición de la dirección donde se encuentra el shellcode
Código que descifra el resto

Determinar la dirección de dónde se encuentra kernel32.dll
Determinar la dirección de la función GetProcAddress
Determinar las direcciones de otras funciones de la librería Kernel32.dll
Carga de la librería ws2_32.dll y extracción de las direcciones de la función Winsock
Se establece la conexión TCP con el sistema (proceso) del intruso
Ejecución del intérprete de comandos con redirección de entrada/salida de la conexión establecida

Parte Cifrada

JCM-05 All rights reserved.

Shellcodes en linux

- Ejemplo – Desbordamiento de Pila

Dirección de retorno

Dirección anterior de EIP

Buf[ ]

Dirección del Shellcode

Shellcode

Opción 1

Dirección del Shellcode

Shellcode

NOP

Opción 2

JCM-05 All rights reserved.

Conceptos avanzados de *hacking memory*

- Kernel Overflows
  - Caso Solaris – LKM vfs_getvfssw()
    - El sistema operacional Solaris implementa mucha de su funcionalidad con Loadable Kernel Modules (LKM)
    - Vfs_getvfssw trata de cargar un file system virtual.
    - Cuando el Kernel recibe la petición para un file system que no ha sido cargado apropiadamente en el espacio del kernel, el kernel automáticamente busca automáticamente un módulo cargado para ese file system.
    - La vulnerabilidad esta en que el kernel carga el módulo de uno de los anteriormente cargados, obteniendo la capacidad para responder el requerimiento.
    - Esto implica que se carga el usuario (que inició el módulo previamente montado), teniendo este último comando de ejecución sobre el Kernel.

JCM-05 All rights reserved.

Conceptos avanzados de *hacking memory*

- Ingeniería Reversa
  - ✓ Es el proceso de extraer el conocimiento o diseño original de cualquier objeto elaborado por el hombre
  - ✓ Aplicado sobre software se pueden distinguir dos tendencias:
    - ☞ Sobre aspectos de seguridad
      - Identificar código malicioso
      - Analizar algoritmos de cifra
      - Auditoría de programas binarios
      - Revisar características de protección del software
    - ☞ Sobre aspectos de desarrollo de software.
      - Evaluar el adecuado funcionamiento de librerías y API's
      - Revisar el funcionamiento de algoritmos
      - Evaluar la calidad y robustez del software

JCM-05 All rights reserved.

Conceptos avanzados de *hacking memory*

Ingeniería Reversa

Herramientas generalmente utilizadas

- Generalmente se utilizan dos tipos de herramientas en esta técnica:
  - Offline Analysis
    - Se toma el ejecutable binario y se utiliza un desemsamblador o decompilador para convertir el código de máquina en código humanamente reconocible.
  - Live Analysis
    - Implica la misma conversión del código de máquina a código humanamente reconocible, pero el análisis se hace en ejecución misma del sistema. Puedes observar los datos internos del programa y cómo afecta el código.
- Dentro del listado de herramientas disponibles están:
  - Desemsambladores : IdaPro (www.datarescue.com), ILDasm (de Microsoft Intermedia Language)
  - Depuradores: OllyDbg (home.t-online.de/home/Ollydbg), Windbg (www.microsoft.com/whdc/devtools/debugging/default.mspx), SoftICE.

JCM-05 All rights reserved.

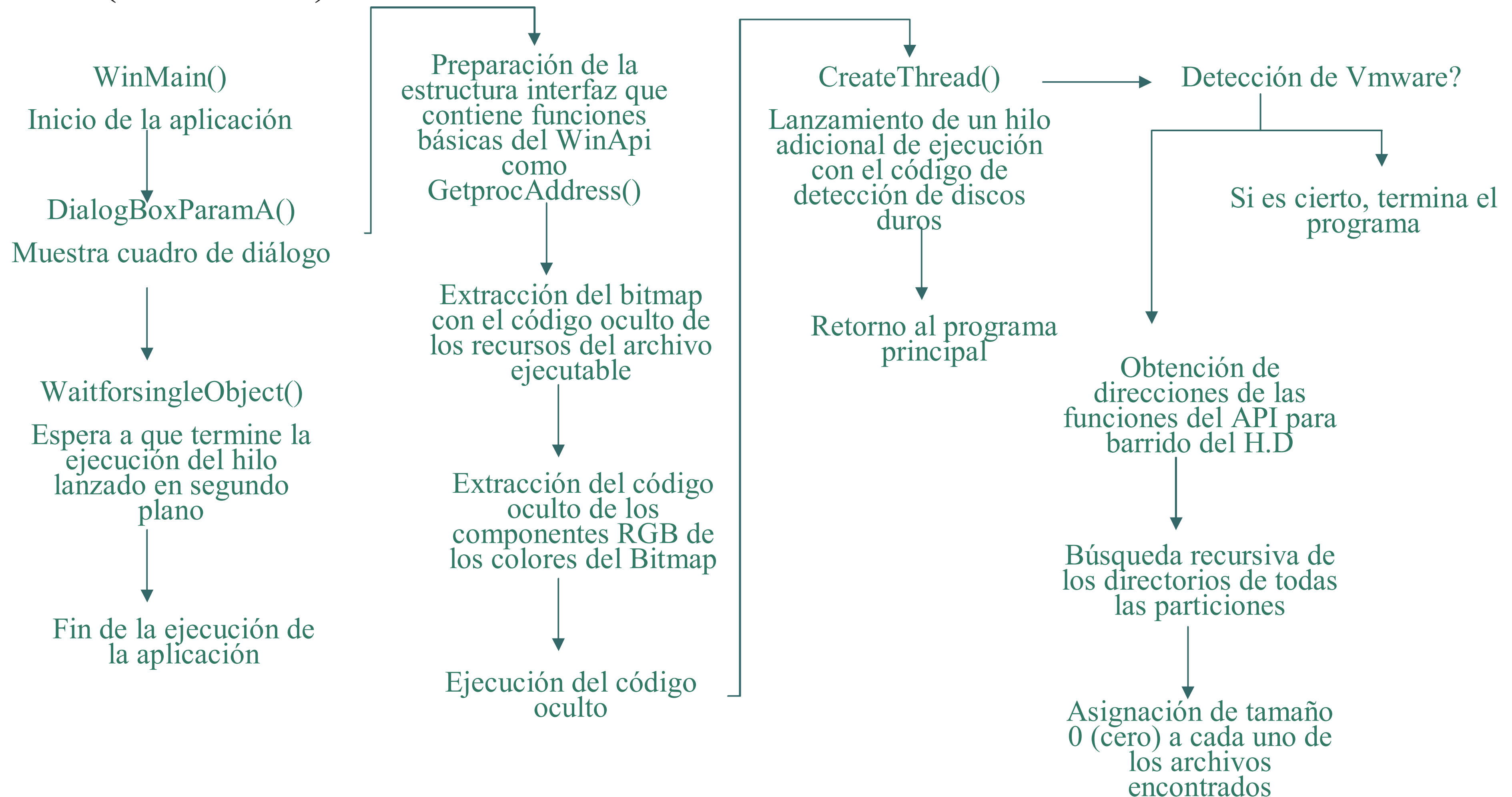
Conceptos avanzados de *hacking memory*

- Ingeniería Reversa
  - Caso – Análisis Formato ELF (Executable and linking Format)
  - ELF, es el formato de tres tipos de objetos binarios, típico para sistemas Linux: reasignables, ejecutables y compartidos.
    - Reasignables: Extensión .o,los cuales son enlazados con otros con el fin de crear el archivo ejecutable o librería compartida.
    - Ejecutables: Los listos para la inicialización, con las referencias resueltas.
    - Compartidos: Extensión .so, los cuales contienen el código y los datos que participan en el proceso de encadenamiento de los objetos en los dos contextos.

JCM-05 All rights reserved.

Conceptos avanzados de *hacking memory*

- Ingeniería Reversa – Un ejemplo malicioso en detalle (Patch.exe)



Tomado de: WOJCIK, B (2005) Análisis del funcionamiento de programas sospechosos. Hakin9. No.1

JCM-05 All rights reserved.

¿Técnicas Anti-reversing?

- Algunas aproximaciones para limitar la ingeniería reversa
  - Eliminar información simbólica
    - Eliminación de información textual del programa: Nombres de las clases, nombres de los objetos globales.
  - Opocar (Obfuscate) el código del programa
    - Hace menos entendible para los humanos el código fuente, manteniendo su funcionalidad.
    - Cambio de nombre de las variables, secuencias en el algoritmo.
  - Código embebido antidebugger
    - Desarrollar un programa que intencionalmente efectúa operaciones si detecta la presencia de un programa de debug.
    - En windows la función que se invoca en el API es IsdebuggerPresent

Tomado de: EILAM, E. (2005) Reversing. Secrets of reverse engineering. John Wiley & Sons. Cap.10

JCM-05 All rights reserved.

Reflexiones Finales

- *Es tiempo de revisar en profundida los códigos fuente, objeto y ejecutable de las aplicaciones. Pruebas de caja blanca*
- *Los ataques más frecuentes se asocian con el manejo de la memoría de la máquina víctima*
- *Es necesario desarrollar habilidades para desemsamblar código de aplicaciones y analizar las interrelaciones entre sus módulos.*
- *La proliferación de Shellcodes, buffer overflows, heap overflows, sugiere un manejo más formal de la memoria de las aplicaciones.*
- *Reconocer las prácticas de programación inseguras para avanzar en algunas estrategias para manejo y control de excepciones en los programas.*
- *Los ataques son menos tangibles, las evidencias están en memoria.*

JCM-05 All rights reserved.

Para finalizar:

"Cuando pruebe un sistema no olvide:

¿Cómo funciona el sistema?
¿Cómo no funciona el sistema?
¿Cómo reacciona ante una falla?
¿Cómo hacerlo fallar?"

Adaptado de:
Bruce Schneier.
Beyond Fear. Thinking Sensibly about security in an uncertain world. 2003.

JCM-05 All rights reserved.

Referencias

- PROCTOR, P. y BYRNES, F. C. (2002) The secured enterprise. Protecting your information assets. Prentice Hall.
- CANO, J. (2004) Inseguridad Informática. Un concepto dual en seguridad informática. http://www.virusprot.com/art47.html
- SCHNEIER, B. (2003) Beyond Fear. Thinking Sensibly about security in an uncertain world. Copernicus Books.
- KRAUSE, M., y TIPTON, H. (Editors) (2004) Handbook of Information Security Management. Auerbach.
- BEJTLICH, R. (2005) The Tao of Network Security Monitoring. Beyond Intrusion Detection. Addison Wesley.
- McNAB, C (2004) Network Security Assessment. O'Reilly.
- HONEYNET PROJECT (2004) Know your enemy. Learning about security threats. Addison Wesley.
- KOZIOL, J., AITEL, D., LITCHFIELD, D., ANLEY, C., EREN, S., MEHTA, N. y HASSELL, R. (2004) The shellcoder's handbook: discovering and exploiting security holes. John Wiley & Sons.
- EILAM, E. (2005) Reversing. Secrets of reverse engineering. John Wiley & Sons.
- ABEL, P. (1996) Lenguaje ensamblador y programación para PC IBM y compatibles. Pearson Educación.
- KASPERSKY, K. (2003) Hacker disassembling unconvered. A-List Publishing.
- CHUVAKIN, A. y PEIKARI, C. (2004) Security warrior. O'reilly.
- NELSON, B., PHILLIPS, A., ENFINGER, F. y STEUART, C. (2004) Guide to computer forensics and investigations. Thomson Learning.
- KRAUSE, W. y HEISER, J. (2002) Computer Forensics. Incident response essentials. Addison Wesley.

JCM-05 All rights reserved.

Referencias

- ALEPH ONE (1996) Smashing the Stack for fun and profit. Phrack. http://www.phrack.org/phrack/49/p49-14.
- SK (2004) History and advances in windows shellcode. Phrack Vol.0xXX, Issue 0x3e, Phile #0x07. http://www.phrack.org/62/p62-0x07_Advances_in_Windows_Shellcode.txt
- BLOMGREN, M. (2004) Introduction to shellcoding. How to exploit buffer overflows. http://tigerteam.se
- ZILLION (2002) Writing Shellcode. http://www.shellcode.com.ar/docz/bof/Writing_Shellcode.html (Consultado el 15-ene-2004)
- PINCUS, J. y BAKER, B. (2004) Beyond stack smashing: recent advances in exploiting buffer overrun. *IEEE Security and Privacy*. Julio/Agosto.
- HILL, S. (2004) Analysis of the exploitation processes. http://www.covertsystems.org.
- WOLAK, M. (2004) Escribimos el shellcode para los sistemas MS Windows. *Hakin9*. No.2. Mayo/Junio.
- WOLAK, M. (2004) Exploit remoto para el sistema windows 2000. *Hakin9*. No.4. Septiembre/Octubre.
- SOBOLEWSKI, P. (2004) Desarrollo de shellcodes en Python. *Hakin9*. No.5. Noviembre/Diciembre.
- SOBOLEWSKI, P. y NIDECKI, T. (2004) Abusos de las cadenas de formato. *Hakin9*. No.5. Noviembre/Diciembre.
- SOBOLEWSKI, P. (2004) Desbordamiento de la pila de Linux x86. *Hakin9*. No.4. Septiembre/Octubre
- JANICZEK, M. (2005) Ingeniería inversa del código ejecutable ELF en el análisis post intrusión. *Hakin9*. No.1. Enero/Febrero.
- WOJCIK, B (2005) Análisis del funcionamiento de programas sospechosos. *Hakin9*. No.1. Enero/Febrero.

JCM-05 All rights reserved.

Hacking Memory
Introducción al Shellcode

Jeimy J. Cano, Ph.D, CFE
Universidad de los Andes

jcano@uniandes.edu.co

JCM-05 All rights reserved.