

每 天 都 有 奇 迹

M I R A C L E S E V E R Y D A Y

<http://www.nsfocus.com>



Hacking Windows CE

san@nsfocus.com

san@xfocus.org



Structure Overview

- Windows CE Overview
- Windows CE Memory Management
- Windows CE Processes and Threads
- Windows CE API Address Search Technology
- The Shellcode for Windows CE
- Windows CE Buffer Overflow Demonstration
- About Decoding Shellcode
- Conclusion
- Reference



Windows CE Overview(1)

- Windows CE is a very popular embedded operating system for PDAs and mobiles
- Windows developers can easily develop applications for Windows CE
- Windows CE 5.0 is the latest version
- This presentation is based on Windows CE.net(4.2)
- Windows Mobile Software for Pocket PC and Smartphone are also based on the core of Windows CE
- By default Windows CE is in little-endian mode





Windows CE Overview(2)

- ARM Architecture
 - RISC
 - ARMv1 - ARMv6
 - ARM7, ARM9, ARM10 and ARM11
 - 7 processor modes
 - 37 registers
 - 15 general-purpose registers are visible at any one time
 - r13(sp), r14(lr)
 - r15(pc) can access directly



Memory Management(1)

4GB Virtual Address	2GB Kernel Space	Kernel Virtual Address: KPAGE Trap Area, KDataStruct, etc	0xFFFFFFFF
		Static Mapped Virtual Address	0xF0000000
		⋮	⋮
		NK.exe	0xC4000000
	⋮	⋮	
	2GB User Space	Memory mapped files	0xC2000000
		⋮	⋮
		Slot 32 Process 32	0x80000000
		⋮	⋮
		Slot 3 Device.exe	0x42000000
		Slot 2 FileSys.exe	0x40000000
		Slot 1 XIP DLLs	0x08000000
		Slot 0 Current Process	0x06000000
			0x04000000
			0x02000000
			0x00000000





Memory Management(2)

- Windows CE uses ROM (read only memory), RAM (random access memory)
 - The ROM in a Windows CE system is like a small read-only hard disk
 - The RAM in a Windows CE system is divided into two areas: program memory and object store
- Windows CE is a 32-bit operating system, so it supports 4GB virtual address space
- Upper 2GB is kernel space, used by the system for its own data



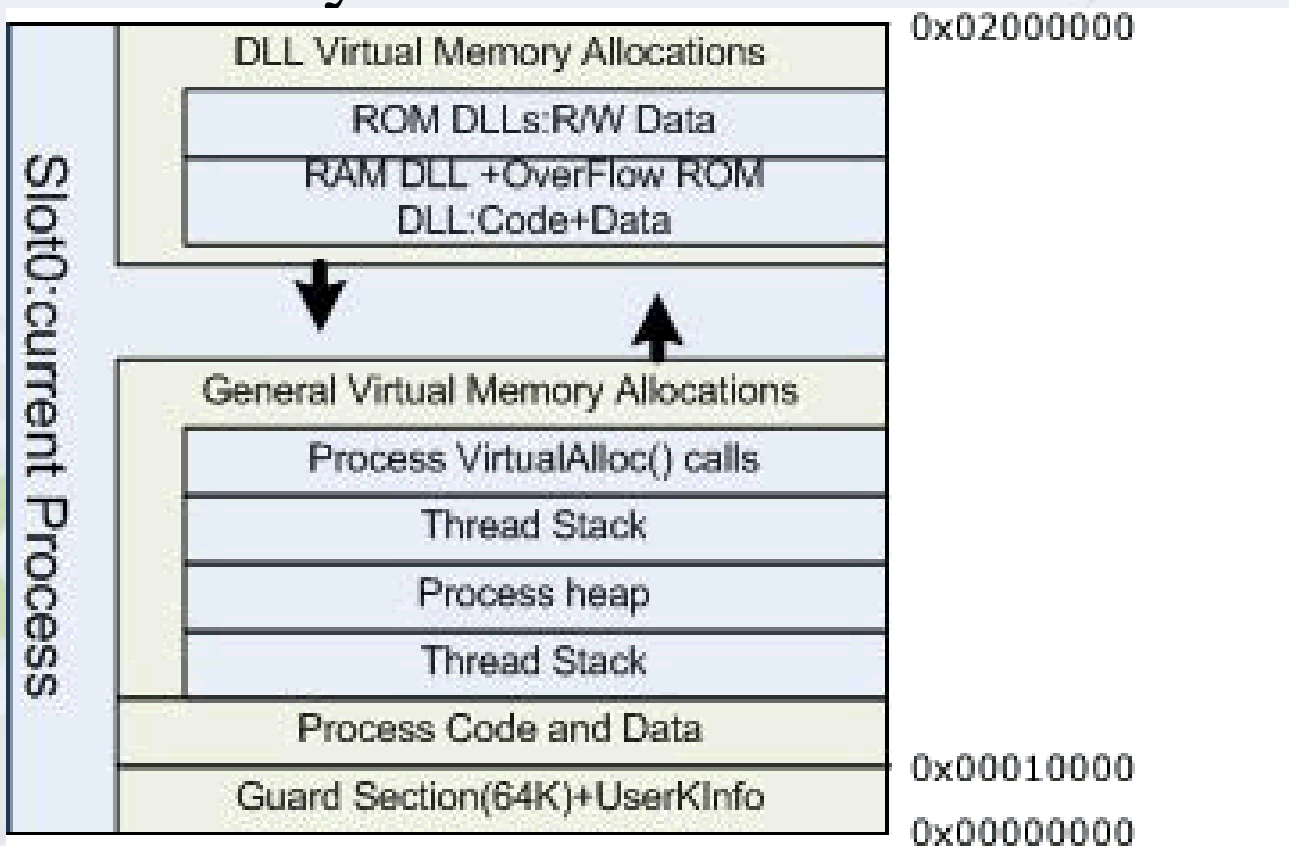


Memory Management(3)

- Lower 2GB is user space
 - 0x42000000-0x7FFFFFFF memory is used for large memory allocations, such as memory-mapped files
 - 0x0-0x41FFFFFF memory is divided into 33 slots, each of which is 32MB

Memory Management(4)

- Slot 0 layout





Processes and Threads(1)

- Windows CE limits 32 processes being run at any one time
- Windows CE restricts each process to its own code and data
- Every process at least has a primary thread associated with it upon starting (even if it never explicitly created one)
- A process can create any number of additional threads (only limited by available memory)
- Each thread belongs to a particular process (and shares the same memory space)
- Each thread has an ID, a private stack and a set of registers





Processes and Threads(2)

- When a process is loaded
 - Assigned to next available slot
 - DLLs loaded into the slot
 - Followed by the stack and default process heap
 - After this, then executed
- When a process' thread is scheduled
 - Copied from its slot into slot 0
- This is mapped back to the original slot allocated to the process if the process becomes inactive
- Kernel, file system, windowing system all run in their own slots



Processes and Threads(3)

- Processes allocate stack for each thread, the default size is 64KB, depending on the link parameter when the program is compiled
 - Top 2KB used to guard against stack overflow
 - Remained available for use
- Variables declared inside functions are allocated in the stack
- Thread's stack memory is reclaimed when it terminates



API Address Search(1)

- Locate the loaded address of the coredll.dll
 - struct KDataStruct kdata; // 0xFFFFFC800: kernel data page
 - 0x324 KINX_MODULES ptr to module list
 - LPWSTR lpszModName; /* 0x08 Module name */
 - PMODULE pMod; /* 0x04 Next module in chain */
 - unsigned long e32_vbase; /* 0x7c Virtual base address of module */
 - struct info e32_unit[LITE_EXTRA]; /* 0x8c Array of extra info units */
 - 0x8c EXP Export table position
- PocketPC ROMs were builded with Enable Full Kernel Mode option
- We got the loaded address of the coredll.dll and its export table position.





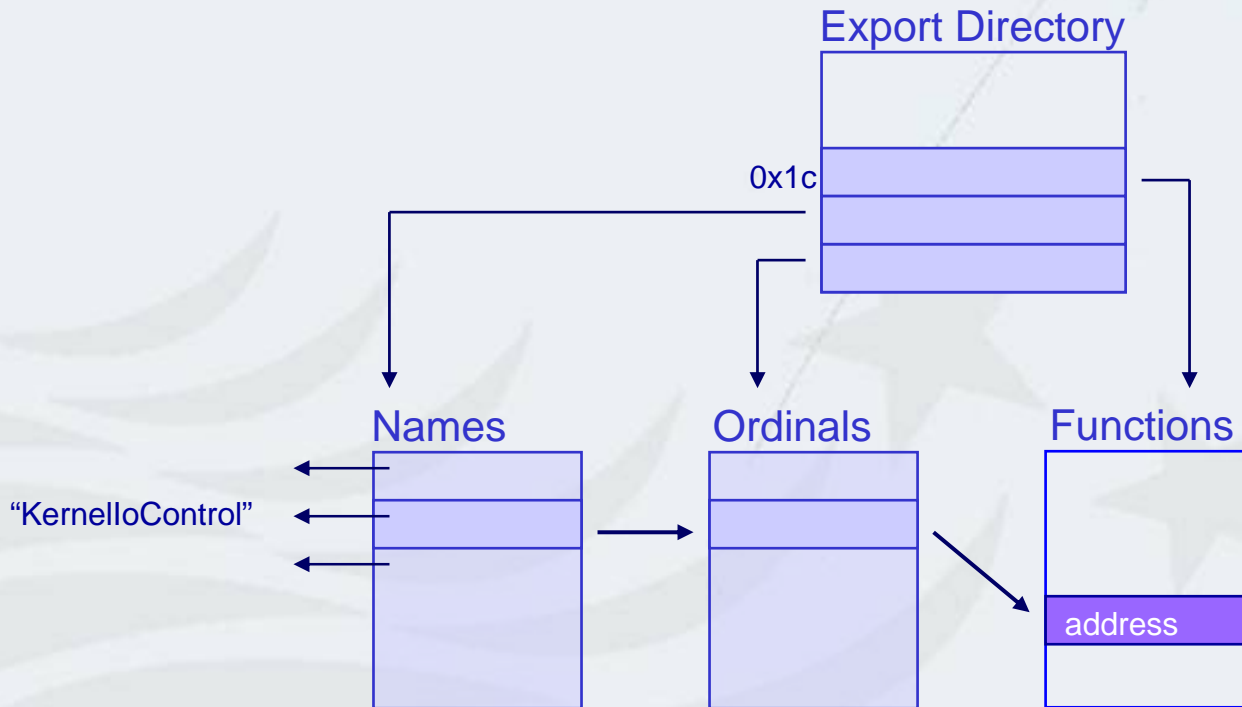
API Address Search(2)

- Find API address via IMAGE_EXPORT_DIRECTORY structure like Win32.

```
typedef struct _IMAGE_EXPORT_DIRECTORY
{
    .....
    DWORD   AddressOfFunctions;    // +0x1c RVA from base of
    image
    DWORD   AddressOfNames;       // +0x20 RVA from base of
    image
    DWORD   AddressOfNameOrdinals; // +0x24 RVA from base of
    image
                                           // +0x28
} IMAGE_EXPORT_DIRECTORY,
 *PIMAGE_EXPORT_DIRECTORY;
```




API Address Search(3)



Shellcode(1)

- test.asm - the final shellcode
 - get_export_section
 - find_func
 - function implement of the shellcode
- It will soft reset the PDA and open its bluetooth for some IPAQs(For example, HP1940)





Shellcode(2)

- Something to attention while writing shellcode
 - LDR pseudo-instruction
 - "ldr r4, =0xffffc800" => "ldr r4, [pc, #0x108]"
 - "ldr r5, =0x324" => "mov r5, #0xC9, 30"
 - r0-r3 used as 1st-4th parameters of API, the other stored in the stack



Shellcode(3)

- EVC has several bugs that makes debug difficult
 - EVC will change the stack contents when the stack releases in the end of function
 - The instruction of breakpoint maybe change to 0xE6000010 in EVC sometimes
 - EVC allows code modify .text segment without error while using breakpoint. (sometimes it's useful)

Buffer Overflow Demo(1)

- hello.cpp - the vulnerable program
 - Reading data from the "binfile" of the root directory to stack variable "buf" by fread()
 - Then the stack variable "buf" will be overflowed
- ARM assembly language uses bl instruction to call function
 - "str lr, [sp, #-4]!" - the first instruction of the hello() function
 - "ldmia sp!, {pc}" - the last instruction of the hello() function
 - Overwriting lr register that is stored in the stack will obtain control when the function returned



Buffer Overflow Demo(2)

- The variable's memory address allocated by program is corresponding to the loaded Slot, both stack and heap
- The process maybe loaded into the difference Slot at each start time, so the base address always alters
- Slot 0 is mapped from the current process' Slot, so its stack address is stable

Buffer Overflow Demo(3)

hello - Microsoft eMbedded Visual C++ [break] - [Disassembly]

File Edit View Insert Project Debug Tools Window Help

[Globals] [All global members] WinMain

hello POCKET PC 200 Win32 (WCE ARMV4) Debug POCKET PC 2003 Device

R0 = 00000200 R1 = 00000000 R2 = 2F3A3403 R3 = FFFFCBAC R4 = 00000005
R5 = 2602FED8 R6 = 00000000 R7 = 2F3A3F5A R8 = FFFFC894 R9 = 243DF818
R10 = 8C12BC50 R11 = 2602FEA8 R12 = 2F3A3403 Sp = 2602FC44 Lr = 01F7688C
Pc = 00011090 Psr = 6000001F

Address: 2fe6c

0002FE1C	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
0002FE2C	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
0002FE3C	41 41 41 41 41 41 41 41 41 41 41 41 41 41 41	AAAAAAAAAAAAAAAA
0002FE4C	5C 62 69 6E 66 69 6C 65 00 00 00 00 2C F2 32 8F	\binfile.....2.
0002FE5C	4C FE 02 26 60 01 03 00 01 00 00 00 EC 10 01 00	L...&`.....
0002FE6C	10 00 00 00 88 FE 02 26 94 11 01 00 5A 3F 3A 2F&.....??:/

21: printf("%d\n", strlen(buf));

```
26011090 E28D0008 add r0, sp, #8
26011094 EB00001F bl |strlen (26011118)|
26011098 E58D0220 str r0, [sp, #0x220]
2601109C E59D1220 ldr r1, [sp, #0x220]
260110A0 E59F0020 ldr r0, [pc, #0x20]
260110A4 EB000024 bl |printf (2601113c)|
```

22: getchar();

```
260110A8 EB000017 bl |getchar (2601110c)|
```

23: fclose(binFileH);

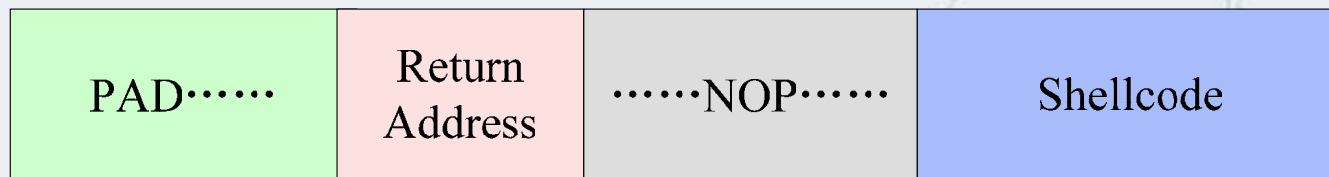
```
260110AC E59D0000 ldr r0, [sp]
260110B0 EB000012 bl |fclose (26011100)|
```

Ready



Buffer Overflow Demo(4)

- A failed exploit



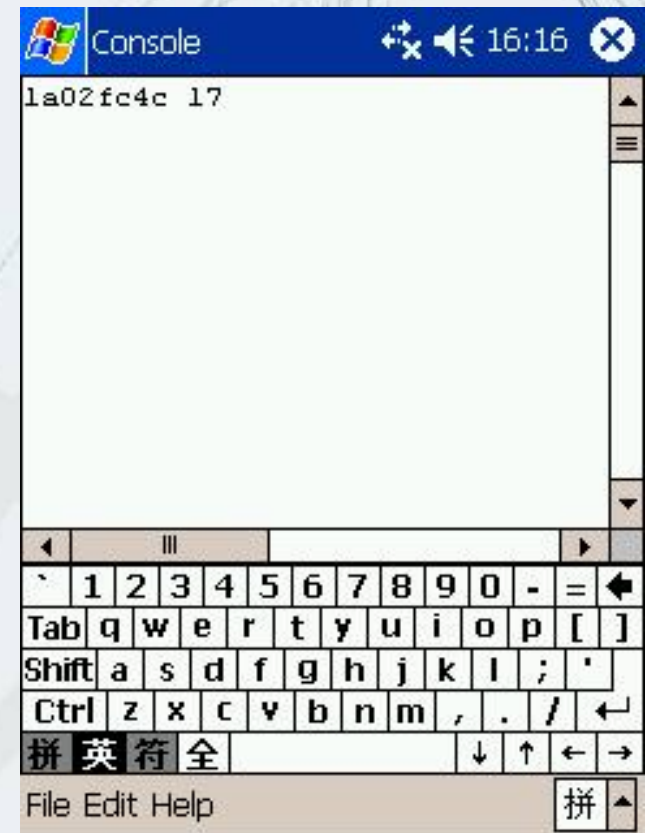
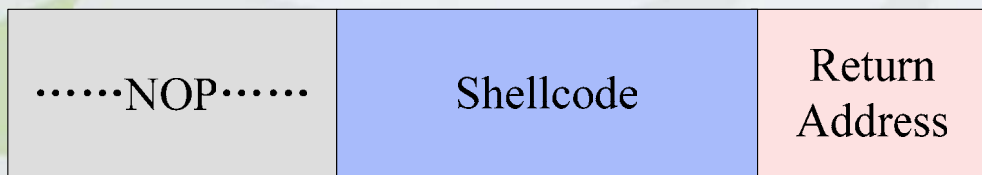
–The PDA is frozen when the hello program is executed

–Why?

- The stack of Windows CE is small
- Buffer overflow destroyed the 2KB guard on the top of stack boundary

Buffer Overflow Demo(5)

- A successful exploit -
exp.c
 - The PDA restarts when the
hello program is executed
 - The program flows to our
shellcode





About Decoding Shellcode(1)

- Why need to decode shellcode?
 - The other programs maybe filter the special characters before string buffer overflow in some situations
 - It is difficult and inconvenient to write a shellcode without special characters by API address search method in Windows CE





About Decoding Shellcode(2)

- The newer ARM processor has Harvard Architecture
 - ARM9 core has 5 pipelines and ARM10 core has 6 pipelines
 - It separates instruction cache and data cache
 - Self-modifying code is not easy to implement





About Decoding Shellcode(3)

- A successful example
 - only use store(without load) to modify self-code
 - you'll get what you want after padding enough nop instructions
 - ARM10 core processor need more pad instructions
 - Seth Fogie's shellcode use this method





About Decoding Shellcode(4)

- A puzzled example
 - load a encoded byte and store it after decoded
 - pad instructions have no effect
 - SWI does nothing except 'movs pc,lr' under Windows CE
 - On PocketPC, applications run in kernel mode. So we can use mcr instruction to control coprocessor to manage cache system, but it hasn't been successful yet





Conclusion

- The codes talked above are the real-life buffer overflow example in Windows CE
- Because of instruction cache, the decoding shellcode is not good enough
- Internet and handset devices are growing quickly, so threats to the PDAs and mobiles become more and more serious
- The patch of Windows CE is more difficult and dangerous





Reference

- [1] ARM Architecture Reference Manual
<http://www.arm.com>
- [2] Windows CE 4.2 Source Code
<http://msdn.microsoft.com/embedded/windowsce/default.aspx>
- [3] Details Emerge on the First Windows Mobile Virus
<http://www.informit.com/articles/article.asp?p=337071>
- [4] Pocket PC Abuse - Seth Fogie
<http://www.blackhat.com/presentations/bh-usa-04/bh-us-04-fogie/bh-us-04-fogie-up.pdf>
- [5] misc notes on the xda and windows ce
<http://www.xs4all.nl/~itsme/projects/xda/>
- [6] Introduction to Windows CE
<http://www.cs-ipv6.lancs.ac.uk/acsp/WinCE/Slides/>
- [7] Nasiry 's way
<http://www.cnblogs.com/nasiry/>
- [8] Programming Windows CE Second Edition - Doug Boling
- [9] Win32 Assembly Components
<http://LSD-PLaNET>

每 天 都 有 奇 迹

M I R A C L E S E V E R Y D A Y

Thank You!

san@nsfocus.com

san@xfocus.org



中联绿盟信息技术（北京）有限公司

NSFOCUS INFORMATION TECHNOLOGY CO.,LTD