Hacking Windows CE

san@nsfocus.com
san@xfocus.org

内容目录

Windows CE简介(1)

- **Windows CE**是**PDA**和手机上使用非常广泛的嵌入式操作系统
- **Windows**开发者可以很方便的开发 **Windows CE** 下的应用程序
- **Windows CE 5.0**是最新版本
- 本文基于**Windows CE.net(4.2)**
- **Windows Mobile Software for Pocket PC and Smartphone**都是基于**Windows CE**的核心
- **Windows CE**默认使用**little-endian**

Windows CE简介(2)

- **ARM架构**
  - **RISC**
  - **ARMv1 - v6**

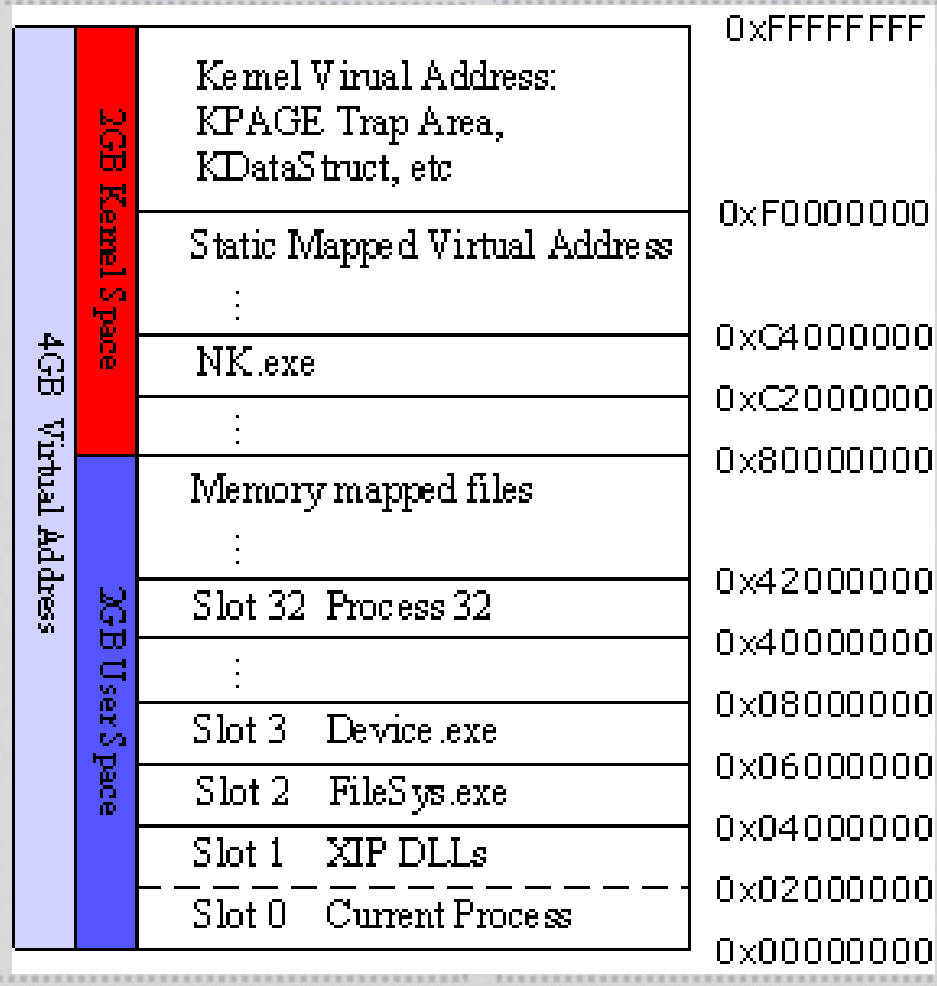
All Modes
Privelege Mode
Exception Mode

User	System	Supervisor	Abort	Undefined	Interrupt	Fast Interrupt
R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8	R8_fiq
R9	R9	R9	R9	R9	R9	R9_fiq
R10	R10	R10	R10	R10	R10	R10_fiq
R11	R11	R11	R11	R11	R11	R11_fiq
R12	R12	R12	R12	R12	R12	R12_fiq
R13	R13	R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14	R14	R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
PC	PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

内存管理(1)

- **Windows CE**使用**ROM (**只读**), RAM (**随机存取**)**
  - **ROM**在**Windows CE**系统中如一个小的只读硬盘
  - **RAM**在**Windows CE**系统分成两部分：程序内存和对象存储
- **Windows CE**是一个**32**位的操作系统，支持**4GB**的虚拟地址空间
- 上面**2GB**是内核空间

内存管理(2)

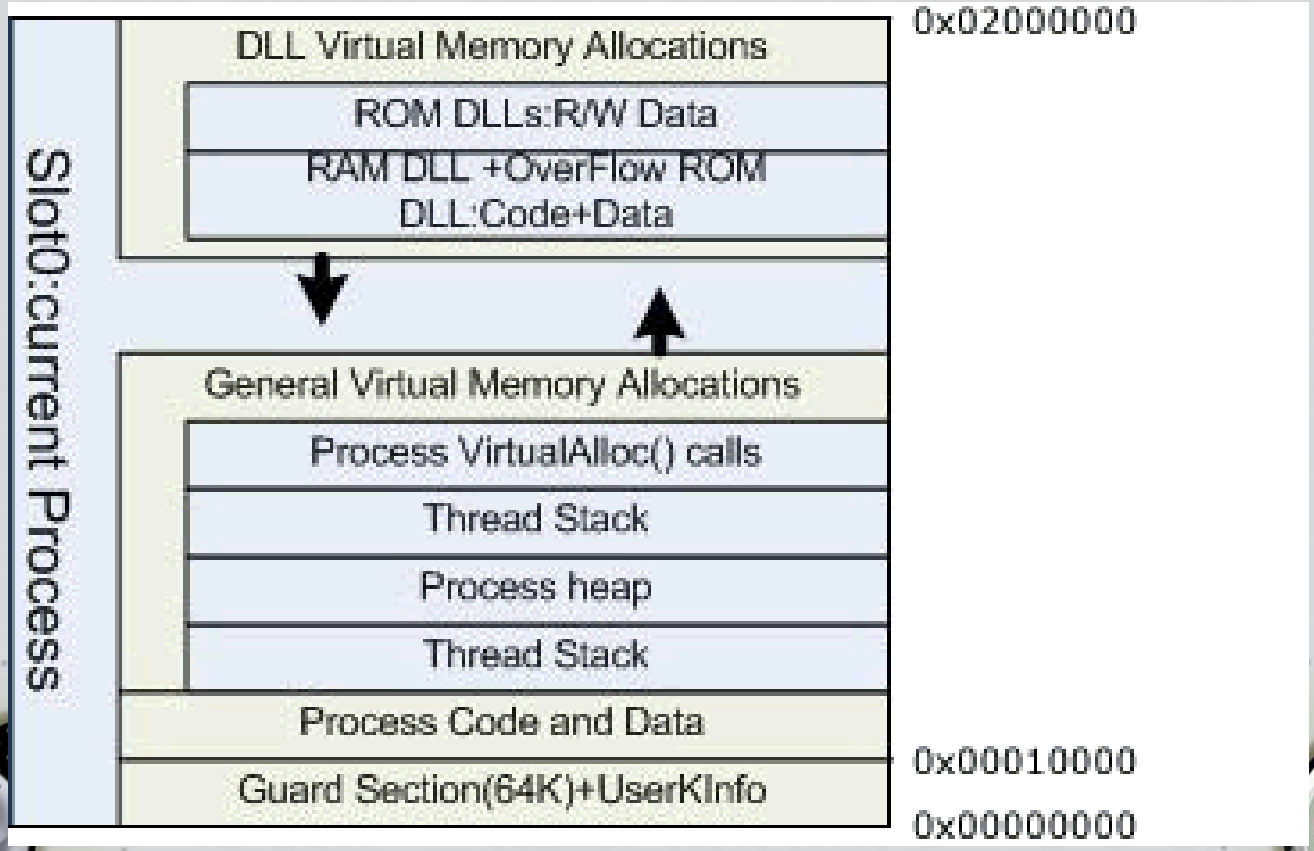

4GB Virtual Address
2GB Kernel Space
2GB User Space
Kernel Virual Address:
KPAGE Trap Area,
KDataStruct, etc
Static Mapped Virtual Address
NK.exe
Memory mapped files
Slot 32 Process 32
Slot 3 Device.exe
Slot 2 FileSys.exe
Slot 1 XIP DLLs
Slot 0 Current Process
0xFFFFFFFF
0xF0000000
0xC4000000
0xC2000000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

内存管理(3)

- 下面**2GB**是用户空间
  - **0x42000000-0x7FFFFFFF**之间的内存被用作大内存分配
  - **0x0-0x41FFFFFF**之间的内存被分成 **33**个**slot**，每个**32MB**

内存管理(4)

- **Slot 0**内存分布



进程和线程(1)

- **Windows CE**限制任何时间只允许**32**个进程同时运行
- 每个进程至少有一个主线程，即使它从来没有创建线程
- 每个进程可以创建任意个线程，只受可用内存的限制
- 每个线程都属于一个进程，并且共享其内存空间
- **SetProcPermissions**这个**API**可以给当前线程访问任意进程的权限
- 每个线程都有一个**ID**、一个私有**stack**和一套寄存器
- **Windows CE**下进程和线程的**ID**实际上就是其句柄

进程和线程(2)

- 当一个进程装载的时候，系统会做如下操作：
  - 分配下一个可用**slot**
  - **DLL**加载入**slot**
  - 接下来是**stack**和默认**heap**
  - 最后开始执行
- 当一个进程的线程活动的时候
  - 进程的**slot**会映射入**slot 0**
- 当进程不活动的时候，这将会映射回原始**slot**

进程和线程(3)

- 进程给每个线程分配**stack**，缺省大小是**64KB**，取决于程序编译时的链接参数
  - 最顶上的**2KB**用来做**stack**溢出保护
  - 剩余的才被使用
- 函数内的局部变量都是从**stack**中分配
- 函数结束的时候，线程的**stack**内存会被回收

API地址搜索技术(1)

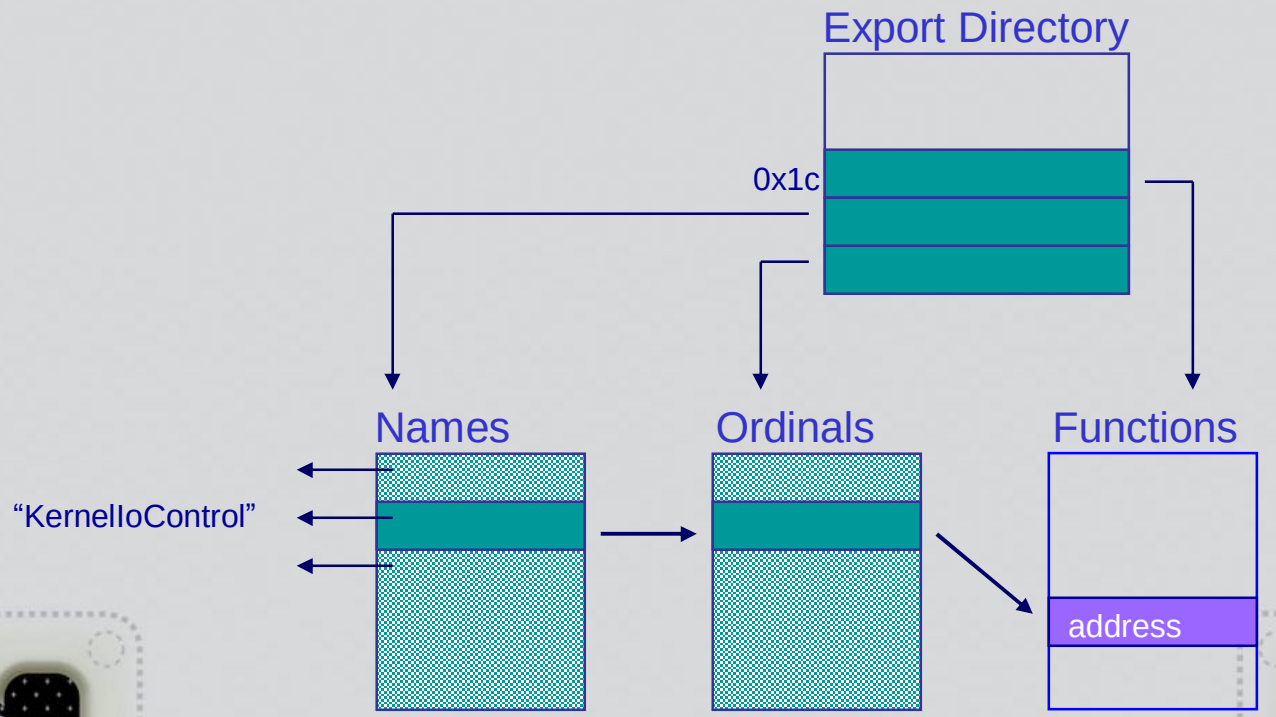
- 定位**coredll.dll**的加载基址
  - **struct KDataStruct kdata; // 0xFFFFC800: PUserKData**
  - **0x324 KINX_MODULES ptr to module list**
  - **LPWSTR lpszModName; /* 0x08 Module name */**
  - **PMODULE pMod; /* 0x04 Next module in chain */**
  - **unsigned long e32_vbase; /* 0x7c Virtual base address of module */**
  - **struct info e32_unit[LITE_EXTRA]; /* 0x8c Array of extra info units */**
    - **0x8c EXP Export table position**
- **PocketPC ROM**在编译的时候使用了**Enable Full Kernel Mode**选项
- 最终得到**coredll.dll**的加载基址和它的导出表相对地址

API地址搜索技术(2)

- 通过**IMAGE_EXPORT_DIRECTORY**结构来搜索**API**地址

```
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地址搜索技术(3)


Export Directory
0x1c
Names
Ordinals
Functions
“KernelIoControl”
address

Shellcode(1)

- **test.asm - 使用此技术的shellcode，源自WinCE4.Dust病毒**
  - **get_export_section**
  - **find_func**
  - 功能实现
- 它的功能时软重启**PDA**并且能打开一些**IPAQ**的蓝牙功能

Shellcode(2)

- 在写**shellcode**的时候，有些地方需要注意
  - **LDR**伪指令
    - **"ldr r4, =0xffffc800" => "ldr r4, [pc, #0x108]"**
    - **"ldr r5, =0x324" => "mov r5, #0xC9, 30"**
  - **r0-r3**寄存器保存**API**的第一个到第四个参数，如果参数大于**4**，那么其它参数会保存到**stack**里

Shellcode(3)

- **EVC**存在一些**bug**使得调试有些麻烦
  - **EVC**在函数结尾回收**stack**的时候会修改**stack**的内容
  - 断点的地方有时被改写成**0xE6000010**
  - 在使用断点或单步执行的时候，**EVC**允许代码修改**.text**段

System Call

- **Windows CE的API由系统调用实现**
- 有一个公式可以计算系统调用的地址
  - **0xf0010000-(256*apiset+apinr)*4**
- 用系统调用实现的**shellcode**比较简单，而且可以在用户模式下使用

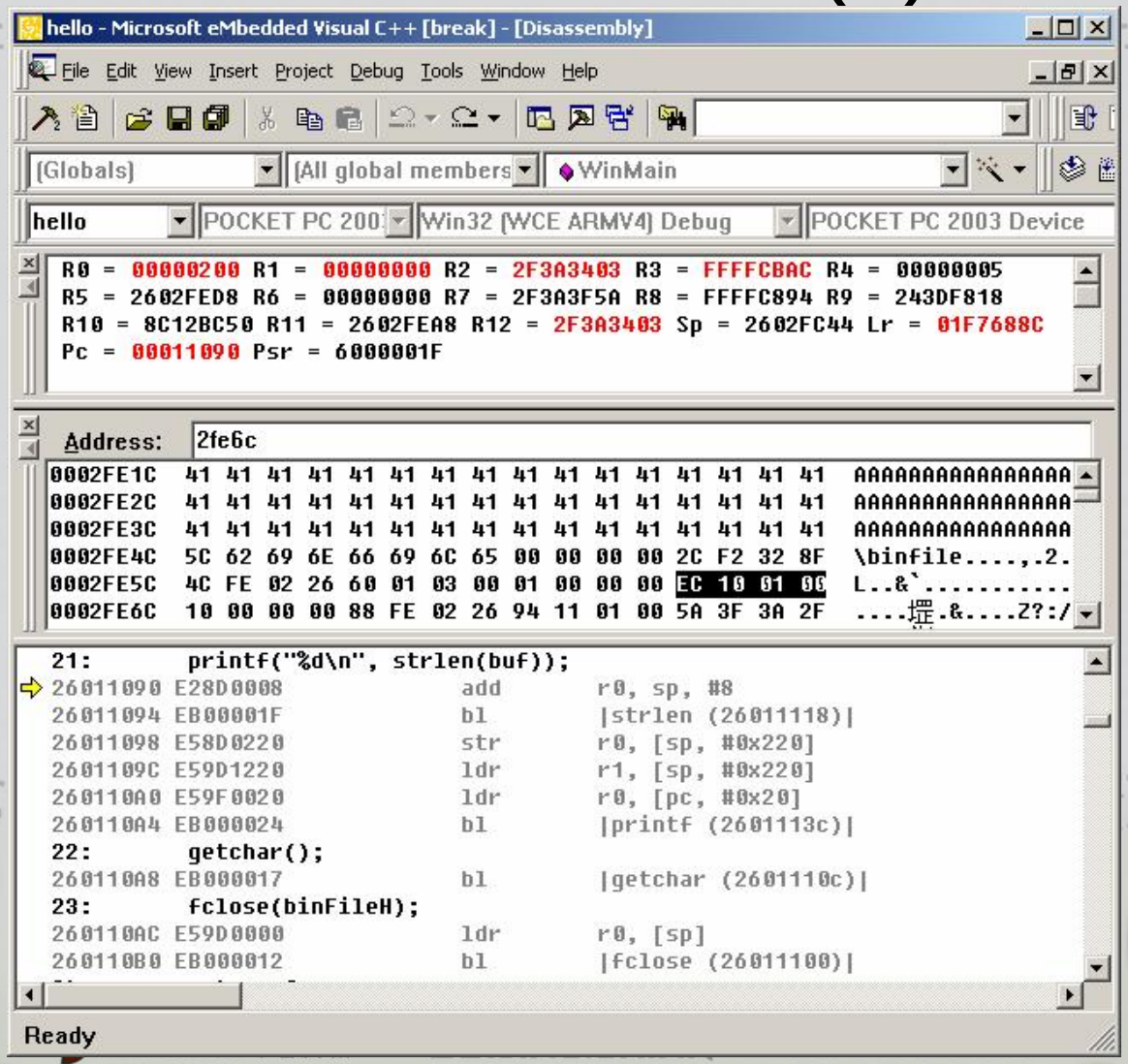
缓冲区溢出示例(1)

- **hello.cpp -** 有漏洞的程序
  - 通过**fread**读取根目录下的**"binfile"**文件到**"buf"**这个**stack**变量
  - **"buf"**变量会被溢出
- **ARM**汇编语言使用**bl**指令来调用函数
  - **"str lr, [sp, #-4]! " - hello()**函数的第一条指令
  - **"ldmia sp!, {pc} " - hello()**函数的最后一条指令
  - 覆盖**lr**寄存器保存在**stack**里的值，可以在函数返回的时候获得控制

缓冲区溢出示例(2)

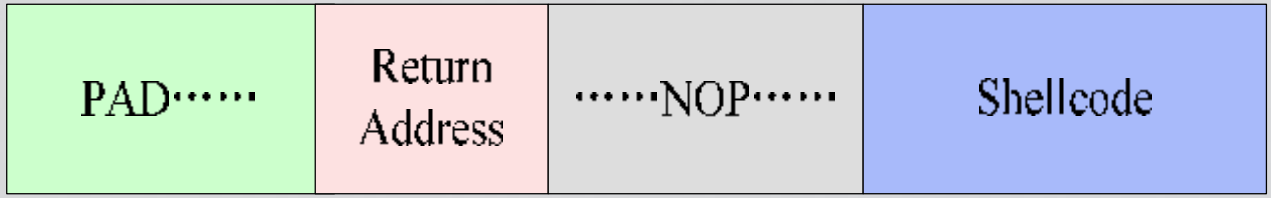
- 变量的内存地址和相应加载的**slot**相关，包括**stack**和**heap**
- 进程在每次启动的时候可能加载到不同的**slot**里，所以基地址会变化
- **slot 0**从当前进程的**slot**映射过来的，所以它的**stack**基地址是固定的

缓冲区溢出示例(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 41 AAAAAAAAAAAAAAAA
0002FE2C 41 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 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 ....&....Z?:/
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

缓冲区溢出示例(4)

- 失败的例子

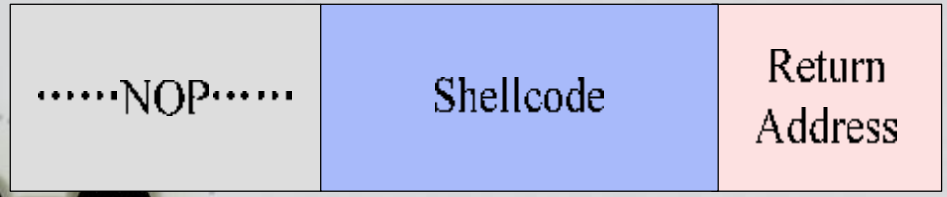


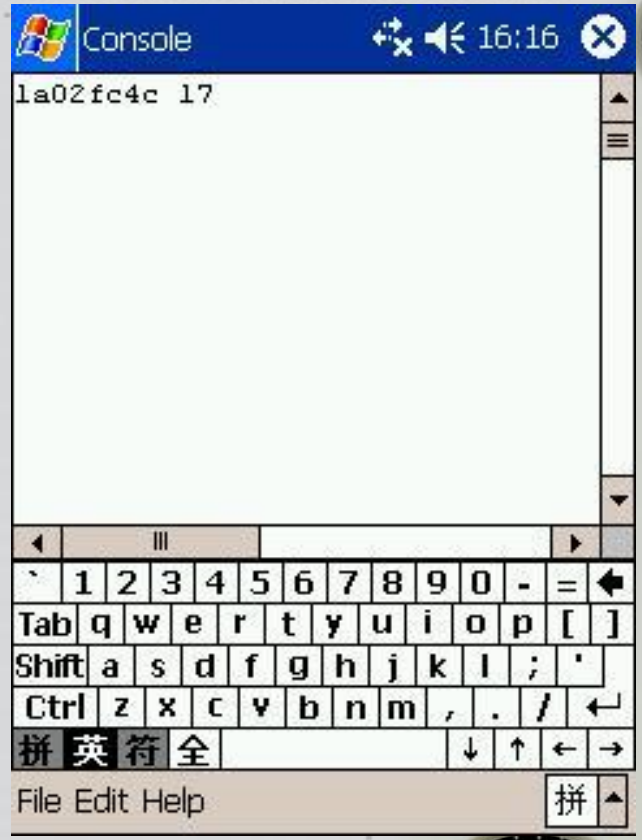
–当hello程序执行的时候，PDA僵死
–原因？

•Windows CE的stack非常小
•溢出可能破坏了栈顶的2KB保护

缓冲区溢出示例(5)

- 成功的例子
  - 当**hello**程序执行的时候，**PDA**重新启动，启动后蓝牙功能打开了
    - 说明程序执行了我们的**shellcode**





关于解码Shellcode(1)

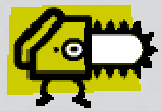
- 为什么需要解码**shellcode?**
  - 有些程序在字符串拷贝的时候会过滤一些特殊字符
  - 在**Windows CE**下要直接写一个不包含某种特殊字符的**shellcode**会比较困难和不方便

关于解码Shellcode(2)

- 现在常用的**ARM**处理器都是**Harvard**架构
  - 它分指令缓存和数据缓存
  - **ARM9**有**5**条流水线，**ARM10**有**6**条流水线
  - 自修改的代码变得不容易实现

关于解码Shellcode(3)

- 可用的案例
  - 只使用**store**没有用**load**来修改自身代码
  - 中间填充足够的空指令就可以得到解码后想要的效果
  - **ARM10**核心的处理器需要更多的填充指令
  - **Seth Fogie**的**shellcode**使用这种方法



关于解码Shellcode(4)

- 一个奇怪的案例
  - 先**load**一个编码的字节，解码后再用**store**保存回去
  - 填充指令没有效果
  - **SWI**指令在**Windows CE**下只是简单的执行**'movs pc,lr'**
  - 由于**PocketPC**上的应用程序跑在内核模式，所以使用**mcr**指令来操作协处理器来管理**cache**系统，但也没有成功

总结

- 上面讨论的代码是**Windows CE**下一个真实的缓冲区溢出实例
- 由于**ARM**处理器缓存的问题，解码**shellcode**并不是很好
- **Internet**和手持设备发展越来越快，所以对于**PDA**和手机的威胁越来越严重
- **Windows CE**下对系统漏洞的**patch**比较麻烦和危险

参考

- [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

Thank You!

san@nsfocus.com
san@xfocus.org