WEBSENSE.
SECURING PRODUCTIVITY™

Executable Encryption for Pocket PC and Smartphone Devices

Nicolas Brulez
Virus Researcher

VB2005

Agenda

- Introduction
- Encrypting Windows Files on Intel x86
- Why it doesn’t work on Windows Mobile (on ARM)?
    - WinCE / Windows Mobile PE Loader
    - Cache
- Encrypting Windows Mobile Files (ARM)
    - Messing with the PE Loader
    - Flushing the cache
    - Methods of Encryption
- Anti Debugging Techniques on ARM
- Conclusion

Introduction

- **Most Malwares are packed/encrypted nowadays**
- **Existing malwares for WinCE or Symbian are not encrypted and thus « easy » to analyse.**
- **Pocket PC and Smartphone executables are using Windows Mobile, and thus the PE File Format**
- **We might expect more malwares targeting devices using WinCE / Windows Mobile in the future**
- **Malware authors may pack/encrypt their new creations in order to protect their code**

WinCE Architecture

- Pocket PC and Smartphone Devices use RISC Processors
- There are different types of ARM processors : ARM, StrongARM, Xscale etc..
- WinCE is based on a revised and reduced version of Windows 2K
- The main system dll is the COREDLL.dll
- System DLLs are inside the ROM
- XIP : eXecute In Place (used to save memory)

ARM Architecture

- ARM General Registers: R0-R15
  - SP Register (R13) is the stack pointer
  - LR Register (R14) holds the return value (for function calls for instance)
  - PC Register (Program Counter or R15) holds the current instruction address+8 (Because of the 3 steps Pre Fetching)
  - Status Registers are R28-R29

www.arm.com for more information

Tools

- MS EVC 4 Debugger
- MS EVC ARM Assembler
- IDA Pro Pocket PC Debugger
- MASM for writing my Encryptor

Encrypting Windows Files on Intel x86

- We need good knowledge of the PE File Format
- ASM knowledge for writing the Loader
- We need to encrypt the code section (or any other section that can be encrypted)
- We can add import handling/protection, but this isn’t mandatory for most files (Some files need it though)

Encrypting Windows Files on Intel x86

- We need to update the PE Header depending of how we modified our file. (SizeOfImage, Sections Characteristics, Entry Point, Section Alignment etc)
- Once encrypted, the new entry point starts with the loader
- Loader will decrypt our sections in memory before jumping to the Original Entry Point. (OEP)

Why it doesn’t work on CE/ARM devices?

- Windows Mobile PE Loader:
  - Windows and Windows Mobile share the same file format but their PE loader is different
  - The Windows Mobile PE Loader is working differently
  - We have to be very careful on what we encrypt, and most importantly, decrypt

Why it doesn't work on CE/ARM devices?

- On Windows we can write inside the whole section virtual memory if we want to, not on Windows Mobile
- The encrypted file won't run
- The Raw Size is actually bigger than the Virtual Size on Windows Mobile files
- This is probably done because of the limited amount of memory we have on current devices

Why it doesn’t work on CE/ARM devices?

- *CACHE*
  - On x86 computers, we don’t have to worry about flushing the cache when we decrypt instructions
  - We do need to take care of it on ARM devices (like in the old days)
  - There aren’t much ways to clean the cache in a stable maner on Windows Mobile
  - On the other hand, we can use that as Anti debugging or Anti Emulation tricks

Encrypting Windows Mobile Files

- Messing with the PE Loader
  - The Windows PE Loader is very friendly
  - We can do almost anything with it :
    - EP before any sections
    - Fancy section raw size
    - Write anywhere inside section Virtual Memory
  - On the other hand, Windows Mobile PE Loader isn’t as nice

Encrypting Windows Mobile Files

- The quick and dirty Windows way won’t work
- We need to use the Virtual Size of the section, to know the number of bytes we need to encrypt
- We can also increase the VirtualSize to match the raw Size and it will work
- But It will also take more memory than the original application

Encrypting Windows Mobile Files

- FLUSHING THE CACHE
  - Unlike x86 computers, we need to flush the cache if we want to execute decrypted code (Else we will execute encrypted code, and our application will crash)
  - There are privileged instructions to do that on ARM devices, but we can’t call them from User Land as it simply crash the device (or even does a hard reset sometimes!)
  - The only way i could find was to use the old and nice FlushInstructionCache API function

Encrypting Windows Mobile Files

- Fortunately, it is possible to rip the code of this function to avoid Dynamic API function resolution. (Like we have on most packers on Windows)
- The WinDust Pocket PC Virus does Dynamic API Resolution
- The FlushInstructionCache function actually use a syscall ☺
- It might not work on future versions of Windows Mobile, but so far, so good

Encrypting Windows Mobile Files

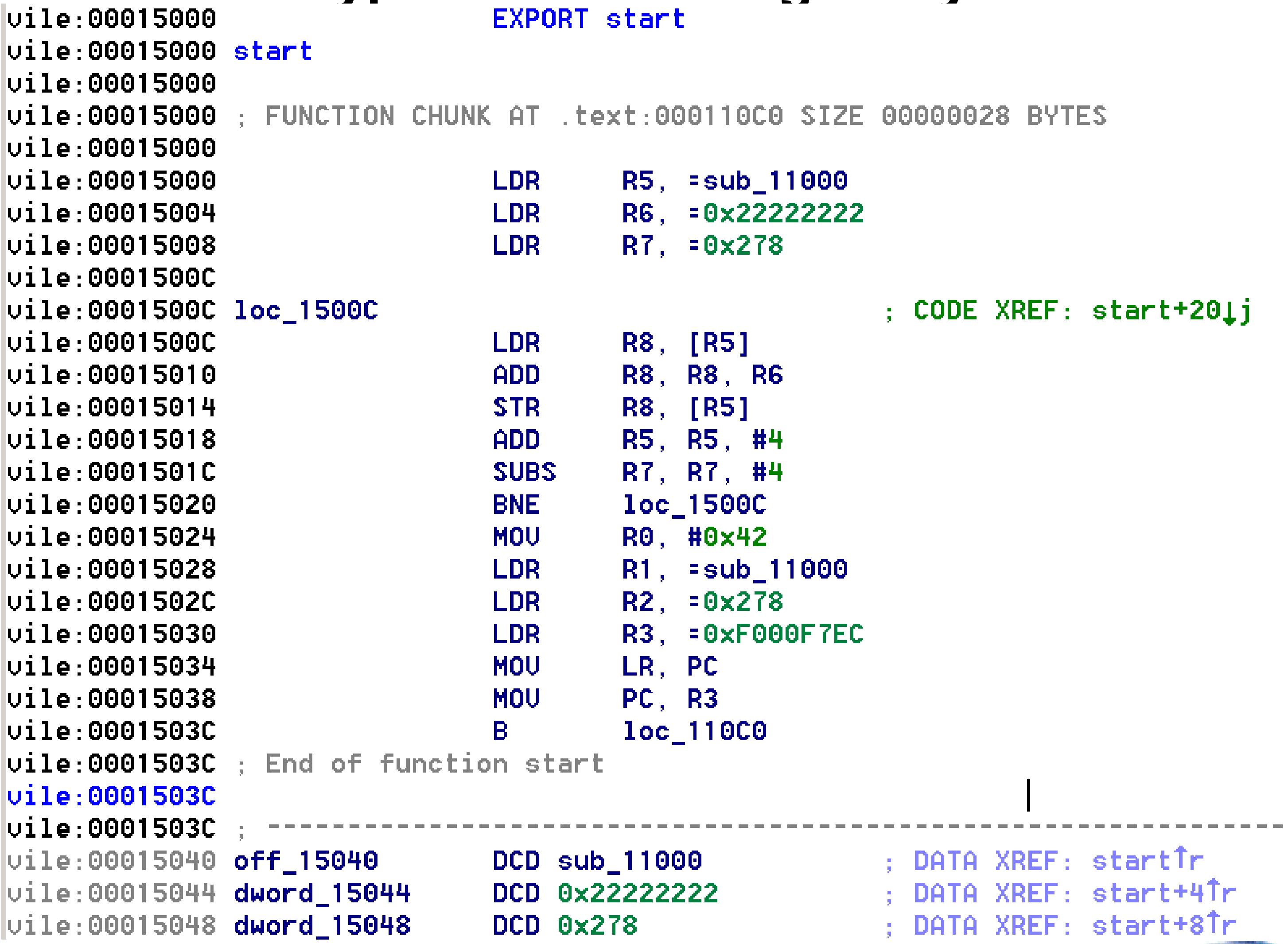
- For better compatibility, we need to use Dynamic API Function Resolution or add another Import Image Descriptor to the import table of the file we want to encrypt
- Windows Mobile will do the job for us
- On the other hand, this will add a weakness to the packer/protector as the API can be hooked.
- We could also put a breakpoint on it

Encrypting Windows Mobile Files

- I came up with two simple (but working) encryption methods while i was doing my research
  - Dword decryption with a single key and a loop (very similar to Windows Packers)
  - Dword based Encryption: Each Encrypted dword is moved inside the Cryptor section and we have a single key for every dwords of the section

Encrypting Windows Mobile Files

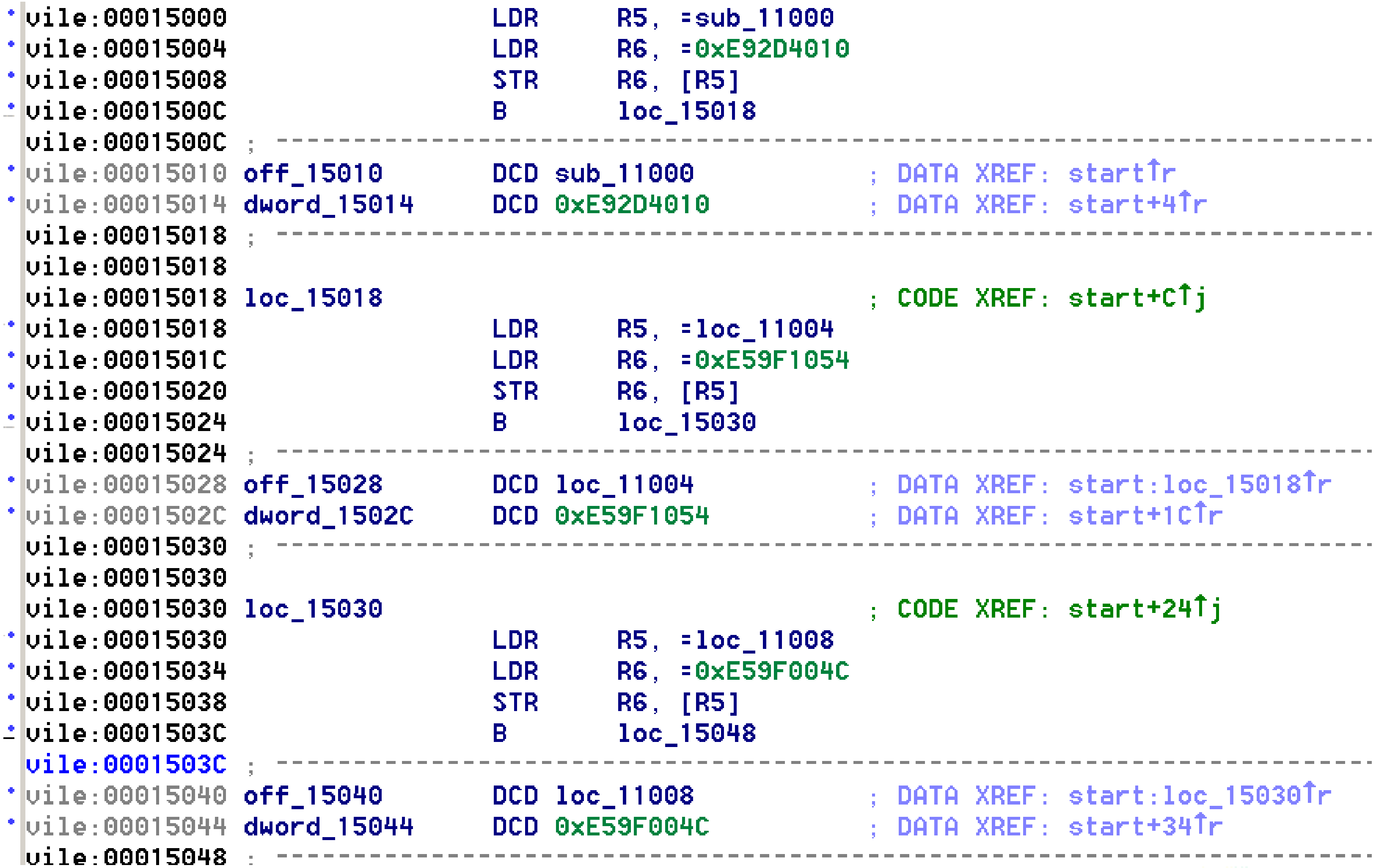
- **Dword decryption with a single key**



```
vile:00015000                 EXPORT start
vile:00015000 start
vile:00015000
vile:00015000 ; FUNCTION CHUNK AT .text:000110C0 SIZE 00000028 BYTES
vile:00015000
vile:00015000                 LDR     R5, =sub_11000
vile:00015004                 LDR     R6, =0x22222222
vile:00015008                 LDR     R7, =0x278
vile:0001500C
vile:0001500C loc_1500C                               ; CODE XREF: start+20↓j
vile:0001500C                 LDR     R8, [R5]
vile:00015010                 ADD     R8, R8, R6
vile:00015014                 STR     R8, [R5]
vile:00015018                 ADD     R5, R5, #4
vile:0001501C                 SUBS    R7, R7, #4
vile:00015020                 BNE     loc_1500C
vile:00015024                 MOV     R0, #0x42
vile:00015028                 LDR     R1, =sub_11000
vile:0001502C                 LDR     R2, =0x278
vile:00015030                 LDR     R3, =0xF000F7EC
vile:00015034                 MOV     LR, PC
vile:00015038                 MOV     PC, R3
vile:0001503C                 B       loc_110C0
vile:0001503C ; End of function start
vile:0001503C
vile:0001503C ; ---------------------------------------------------------------
vile:00015040 off_15040       DCD sub_11000           ; DATA XREF: start↑r
vile:00015044 dword_15044     DCD 0x22222222          ; DATA XREF: start+4↑r
vile:00015048 dword_15048     DCD 0x278               ; DATA XREF: start+8↑r
```

Encrypting Windows Mobile Files

- **Dwords moved in the Packer Section**



```
vile:00015000                  LDR      R5, =sub_11000
vile:00015004                  LDR      R6, =0xE92D4010
vile:00015008                  STR      R6, [R5]
vile:0001500C                  B        loc_15018
vile:0001500C ; ---------------------------------------------------------------------------
vile:00015010 off_15010        DCD sub_11000           ; DATA XREF: start↑r
vile:00015014 dword_15014      DCD 0xE92D4010          ; DATA XREF: start+4↑r
vile:00015018 ; ---------------------------------------------------------------------------
vile:00015018
vile:00015018 loc_15018                                ; CODE XREF: start+C↑j
vile:00015018                  LDR      R5, =loc_11004
vile:0001501C                  LDR      R6, =0xE59F1054
vile:00015020                  STR      R6, [R5]
vile:00015024                  B        loc_15030
vile:00015024 ; ---------------------------------------------------------------------------
vile:00015028 off_15028        DCD loc_11004           ; DATA XREF: start:loc_15018↑r
vile:0001502C dword_1502C      DCD 0xE59F1054          ; DATA XREF: start+1C↑r
vile:00015030 ; ---------------------------------------------------------------------------
vile:00015030
vile:00015030 loc_15030                                ; CODE XREF: start+24↑j
vile:00015030                  LDR      R5, =loc_11008
vile:00015034                  LDR      R6, =0xE59F004C
vile:00015038                  STR      R6, [R5]
vile:0001503C                  B        loc_15048
vile:0001503C ; ---------------------------------------------------------------------------
vile:00015040 off_15040        DCD loc_11008           ; DATA XREF: start:loc_15030↑r
vile:00015044 dword_15044      DCD 0xE59F004C          ; DATA XREF: start+34↑r
vile:00015048 ; ---------------------------------------------------------------------------
```

PROS AND CONS OF THOSE METHODS

- Encryption with a normal Loop
  - Pros
    - The Loader is very small
    - Very Fast
  - Cons
    - Same code for the whole section
    - Same key (easy to break it)
    - It is easy to bypass the decryption using breakpoints
    - We need to flush the cache
    - Some files won’t run correctly

PROS AND CONS OF THOSE METHODS

- Dwords Moved and Encrypted with different keys
  - Pros
    - Different block of code for every dword with different keys or algorithm if we want
    - We can write « Pseudo Polymorphic » loaders, especially with all the registers we have on ARM
    - If well done, there is no easy way to bypass the whole decryption (must not be linear of course)
    - Emulation is slower as it has a lot of operations done for every dwords
    - NO need to flush the cache
    - Worked on every files i tested it on

PROS AND CONS OF THOSE METHODS

- Cons
  - The encrypted file is quite bigger (but we could remove the first section completely to decrease file size)
  - Slower than a normal decryption loop

More Encryption?

- It is possible to use crypto to encrypt our code, now that we know how to flush the cache, and what need to be encrypted/decrypted
- Those two methods were just for testing purpose
- I have a more complex protector already working ☺

Anti Debugging Tricks on WinMobile / ARM

- The best Anti Debugging trick i could find was playing with the cache
- We can dynamically encrypt / modify instructions that should NOT be modified inside our protector
- If we don’t flush the cache, our application will run fine, as our modifications will be ignored
- What happens when we debug such code ?
- Debuggers flush the cache, and we end up executing garbage code or we could be redirected to fake routines!

Anti Debugging Tricks on Win Mobile / ARM

- We can use timing detection using GetTickCount API function or similar functions (Like we have on Windows already)

- When playing with the new IDA Remote Debugger for Pocket PC, i found interesting functions:
  - AttachDebugger
  - DebuggerConnect

- I invited anyone interested to do some research on those functions ☺

Conclusion

- Executable Encryption is possible on Windows Mobile and ARM devices
- More research needs to be done
- We need to work on unpacking tools for those packers (I already have a few ones)
- If you want more information, read my paper in the proceedings or email me
- The two examples here are for proof of concept, but i am working on ARMadillo for Pocket PC (and it works ;)

Questions?

- **If you have any questions, please talk SLOOOWLY, or just talk to me after the presentation. (Better :p)**

- **Thanks ☺**

nbrulez@websense.com

http://WebsenseSecurityLabs.com