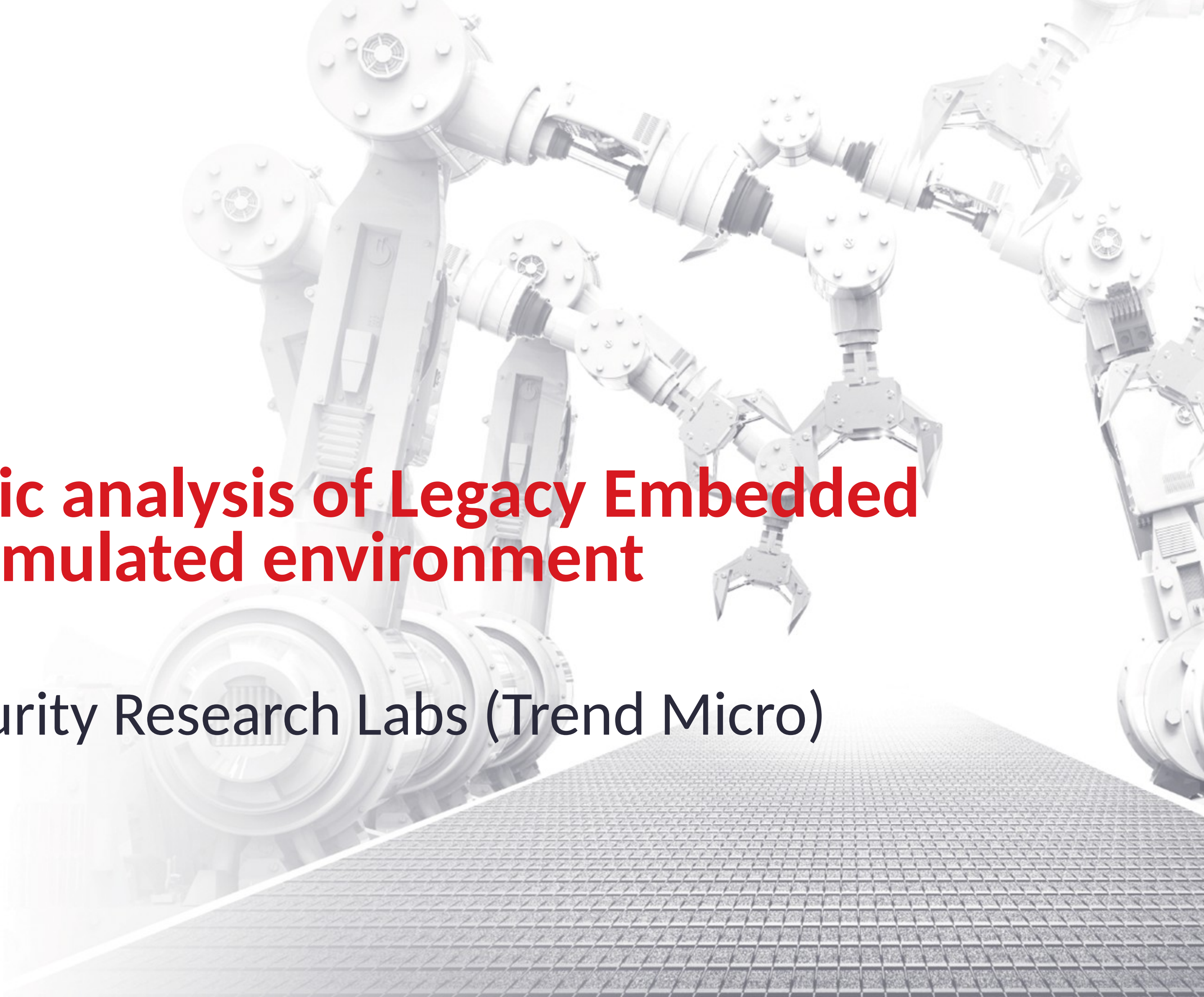


Enabling dynamic analysis of Legacy Embedded Systems in full emulated environment

Ta-Lun Yen
TXOne IoT/ICS Security Research Labs (Trend Micro)

$(whoami)

- @evanslify
- Researcher @ TXOne Networks (Trend Micro), 2019/11-present
- Reverse Engineering, protocol analysis, wireless, *hardware*

Outline

- Our goals & Background of Windows CE6
- CE6 Bootloader & power-on initialization
- Inside CE6 Application Loader & Memory management
- Reconstructing extracted binaries to dynamic execution
- Conclusion

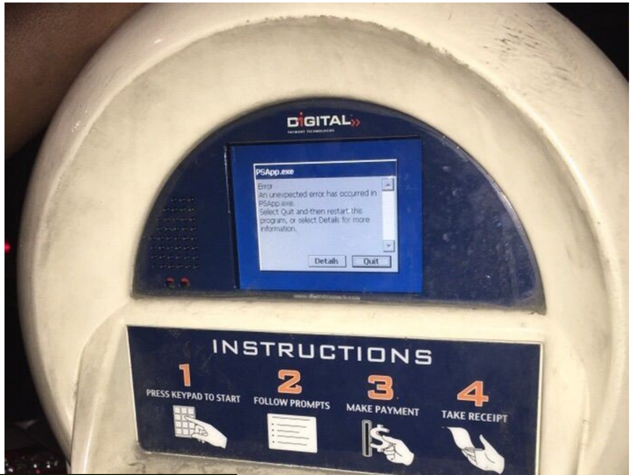
Our goal

- Emulate CE6 image from device with QEMU
- We don’t want to buy every hardware for research
  - We ended up buying one actually (for comparison)
- Serial ports & debugger is not present on every hardware

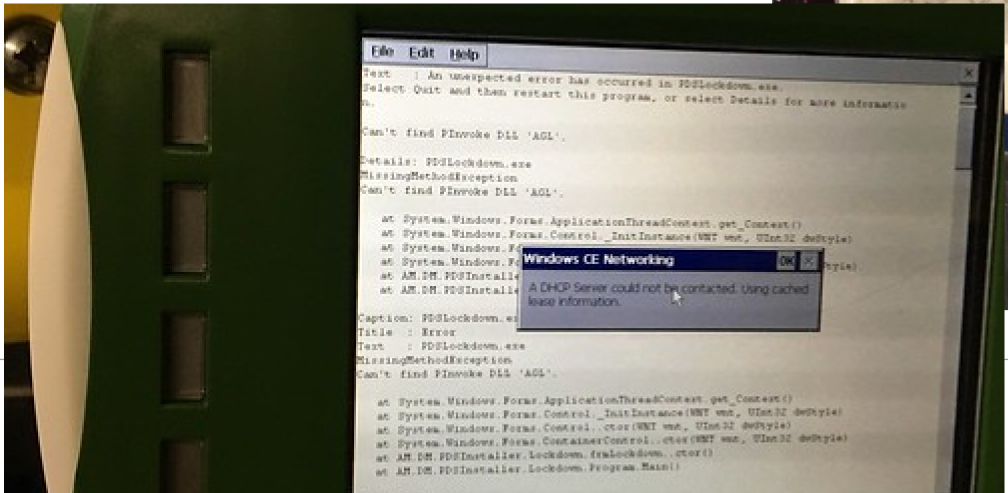
Background of Windows CE6

Horrors from the ancient

- WinCE hasn't been actively exploited (yet…)
- However, it runs everywhere
    - In cars, Parking meters, aircraft IFEs, subway turnstiles, medical devices, power plants...



```
PSApp.exe
Error
An unexpected error has occurred in
PSApp.exe.
Select Quit and then restart this
program, or select Details for more
information.
Details   Quit
```



```
Text    : An unexpected error has occurred in PDSLockdown.exe.
Select Quit and then restart this program, or select Details for more informatio
n.

Can't find PInvoke DLL 'AGL'.

Details: PDSLockdown.exe
MissingMethodException
Can't find PInvoke DLL 'AGL'.

   at System.Windows.Forms.ApplicationThreadContext.get_Context()
   at System.Windows.Forms.Control._InitInstance(WNT wnt, UInt32 dwStyle)
   at System.Windows.Fo
   at System.Windows.Fo                                              yle)
   at AM.DM.PDSInstalle
   at AM.DM.PDSInstalle

Caption: PDSLockdown.ex
Title   : Error
Text    : PDSLockdown.exe
MissingMethodException
Can't find PInvoke DLL 'AGL'.

   at System.Windows.Forms.ApplicationThreadContext.get_Context()
   at System.Windows.Forms.Control._InitInstance(WNT wnt, UInt32 dwStyle)
   at System.Windows.Forms.Control..ctor(WNT wnt, UInt32 dwStyle)
   at System.Windows.Forms.ContainerControl..ctor(WNT wnt, UInt32 dwStyle)
   at AM.DM.PDSInstaller.Lockdown.frmLockdown..ctor()
   at AM.DM.PDSInstaller.Lockdown.Program.Main()
```

```
Windows CE Networking   OK ×
A DHCP Server could not be contacted. Using cached
lease information.
```

Difference between {NT, CE}

- Microsoft Shared Source Initiative: (partial) source code
- Loosely adheres to NT APIs and behavior
- Real-time OS

Difference between {NT, CE}

- While having different APIs and behaviors between CE and NT...
- Some exploits and techniques might work on both CE & NT
  - ...with some efforts, e.g MS17-010 [1]

[1] https://www.fracturelabs.com/posts/2017/exploiting-ms17-010-on-windows-embedded-7-devices/

Current methods to study CE6 firmware

- File extraction
  - https://github.com/nlitsme/eimgfs (was dumprom)
- Dynamic debugger
  - CeGCC http://cegcc.sourceforge.net/
- Mass storage & extract files (unlikely for drivers)
- Limitations
  - You cannot run them in your environment with MS emulator or QEMU… until now

Round 1
Straight up & go to emulation

CE6 Booting process

- BIOS bootloader / DOS loader (loadcepc.exe)
- Similar to most embedded x86's
  - Hardware & platform initialization
  - Load & start the OS
  - Having access to serial / KITL would be great
- At this point, we assume its just like any x86 machine, and easy to QEMU

CE6 Firmware format

- “B000FF format”
  - .bin for properly packed format
    - Can be used with DOS
  - .nb0 for 1:1 RAM
    - Can only be used with BIOS

- Our target contains a .nb0, and we can convert it into a .bin
  - By specifying a address from the start of .nb0

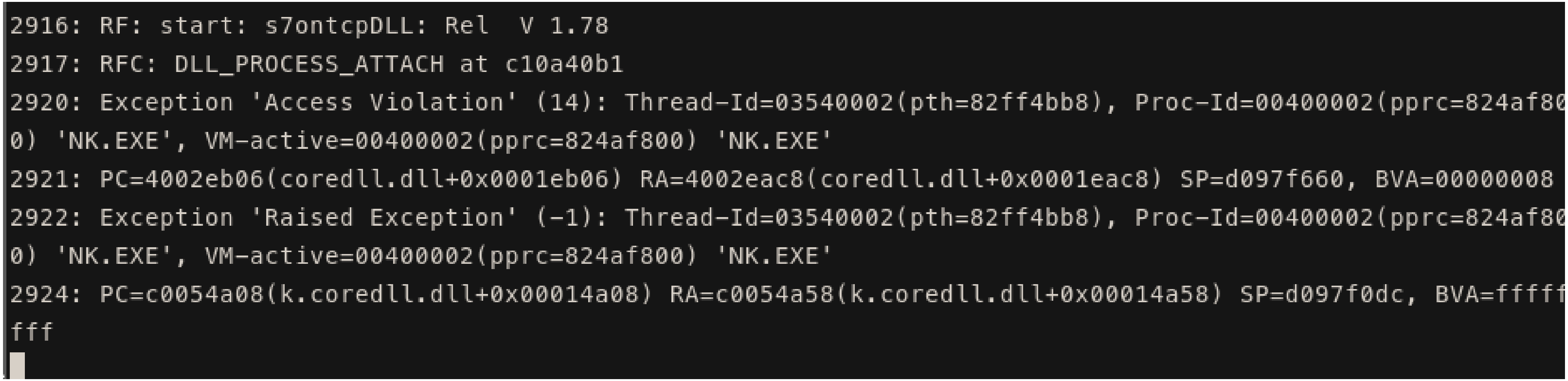
```
struct BIN_HEADER {
  char[7] Signature; // B000FF\n signature

  DWORD ImageStart;   // Image Start
  DWORD ImageLength; // Image Length
};

struct BIN_BLOCK {
  DWORD Address; // memory address
  DWORD Size;
  DWORD Checksum; // CRC32
};
```

Our 1st failed approach

- Kernel loads, partial initialization can be done
- But, it never fully boot to desktop



```
2916: RF: start: s7ontcpDLL: Rel  V 1.78
2917: RFC: DLL_PROCESS_ATTACH at c10a40b1
2920: Exception 'Access Violation' (14): Thread-Id=03540002(pth=82ff4bb8), Proc-Id=00400002(pprc=824af80
0) 'NK.EXE', VM-active=00400002(pprc=824af800) 'NK.EXE'
2921: PC=4002eb06(coredll.dll+0x0001eb06) RA=4002eac8(coredll.dll+0x0001eac8) SP=d097f660, BVA=00000008
2922: Exception 'Raised Exception' (-1): Thread-Id=03540002(pth=82ff4bb8), Proc-Id=00400002(pprc=824af80
0) 'NK.EXE', VM-active=00400002(pprc=824af800) 'NK.EXE'
2924: PC=c0054a08(k.coredll.dll+0x00014a08) RA=c0054a58(k.coredll.dll+0x00014a58) SP=d097f0dc, BVA=fffff
fff
```

Our 1st failed approach

- Hardware differences in QEMU and actual device
  - AMD Geode(!) vs. Q35/i440FX (QEMU)
- It is naive to assume this would work straightforward!
  - Need to have corresponding devices in QEMU
  - I/O points, special flash memory, etc
- Approach is very time-consuming
  - Patched multiple if-else, I/O checks, an graphics driver

What we learned

- QEMU-lating an image as-is is very, very difficult
- Device-specific modification must be made
- Binary patching on this scale is very unpleasant

Round 2
Application loader/Memory management

CE6 Application loader

- Straight up emulation does not work
  - What if we can move binaries from another image to our own?
  - All of drivers, libraries, etc
- Figure out if we can:
  - Extract driver & files from image
  - Build our own image
  - Make extracted files run in our image

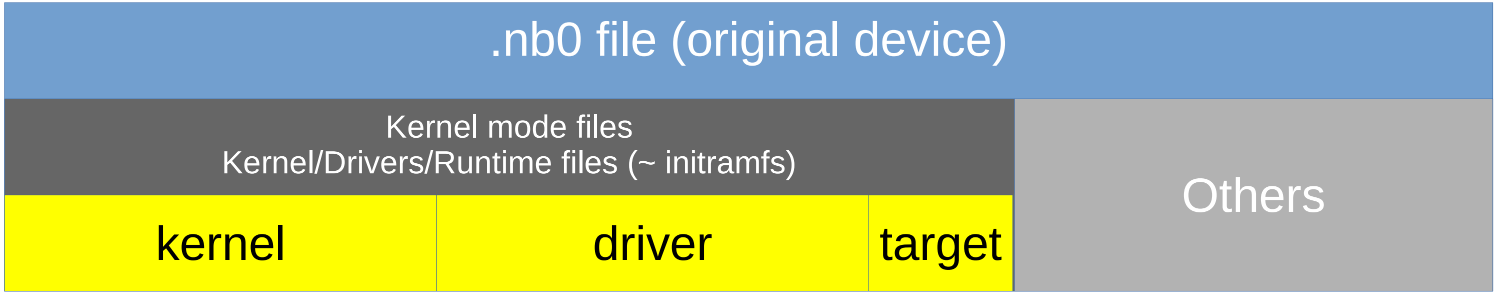
CE6 Application loader

- Straight up emulation does not work
  - What if we can move binaries from another image to our own?
  - All of drivers, libraries, etc
- Figure out if we can:
  - Extract driver & files from image → Yes, using eimgfs
  - Build our own image → Yes, CE6 SDK
  - Make extracted files run in our image → It crashed right away

CE6 Application loader

- Like NT…. Or not
- Kernel parses PE header, loads libraries, allocate memories, and run the PE
- **If ImageBase is fixed, and the address is already used, the kernel assigns a next free page.**
  - **Without .reloc, it will not fail (in CE6)**
  - **This causes kernel to crash most of the time**

Moving files from an image to another



.nb0 file (original device)

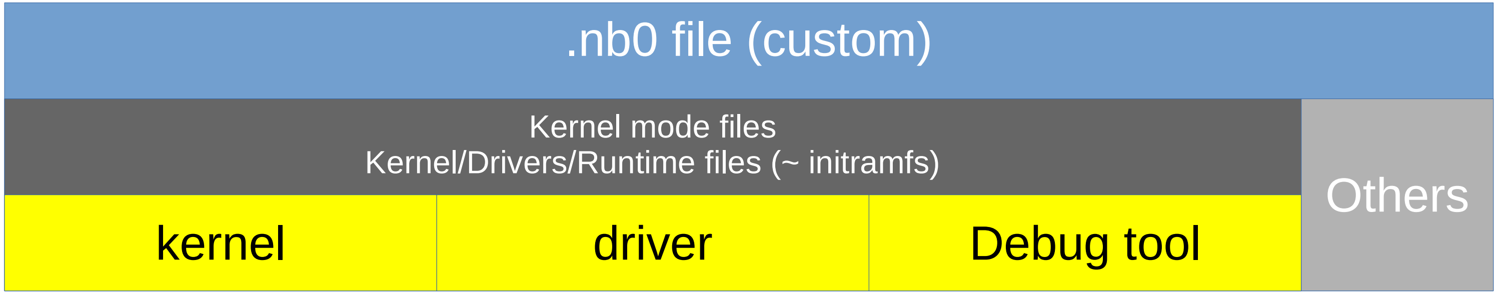
Kernel mode files
Kernel/Drivers/Runtime files (~ initramfs)

kernel

driver

target

Others



.nb0 file (custom)

Kernel mode files
Kernel/Drivers/Runtime files (~ initramfs)

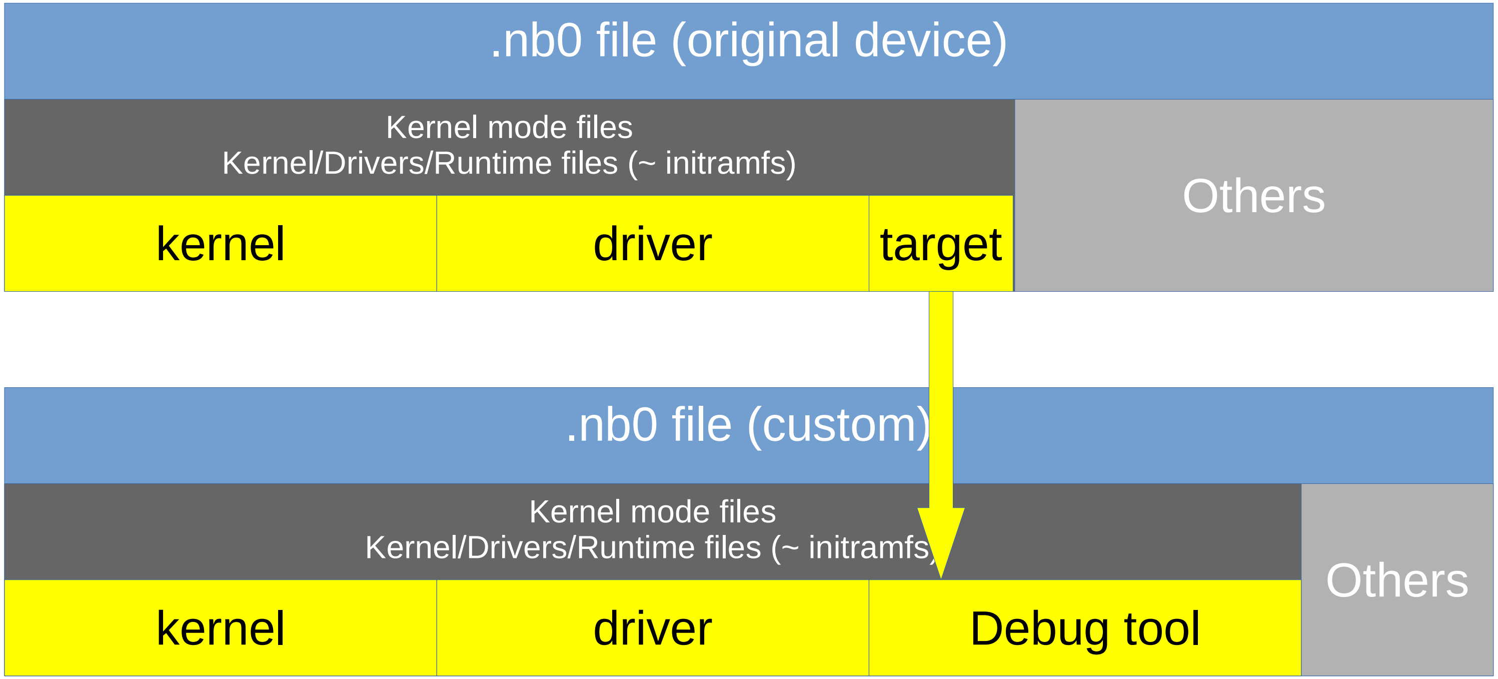
kernel

driver

Debug tool

Others

Moving files from an image to another



.nb0 file (original device)

Kernel mode files
Kernel/Drivers/Runtime files (~ initramfs)

kernel | driver | target

Others

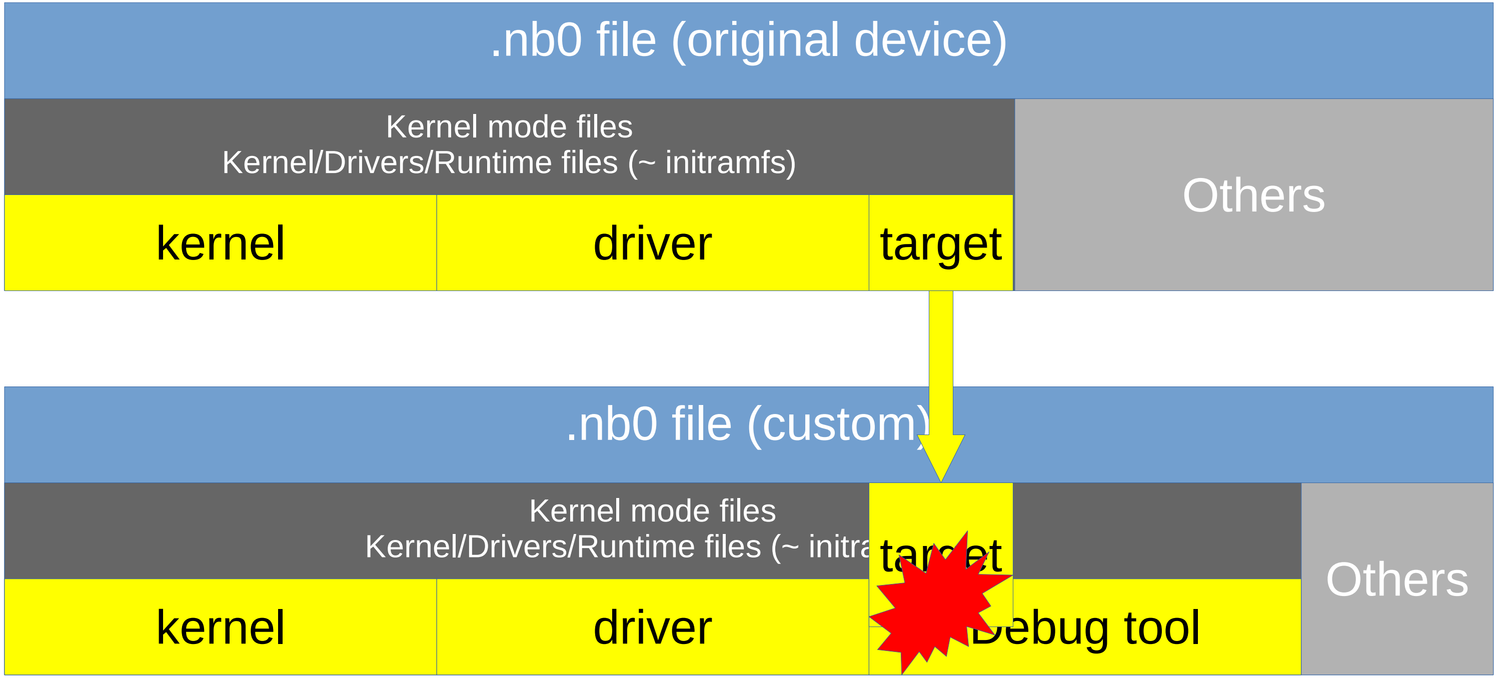
.nb0 file (custom)

Kernel mode files
Kernel/Drivers/Runtime files (~ initramfs)

kernel | driver | Debug tool

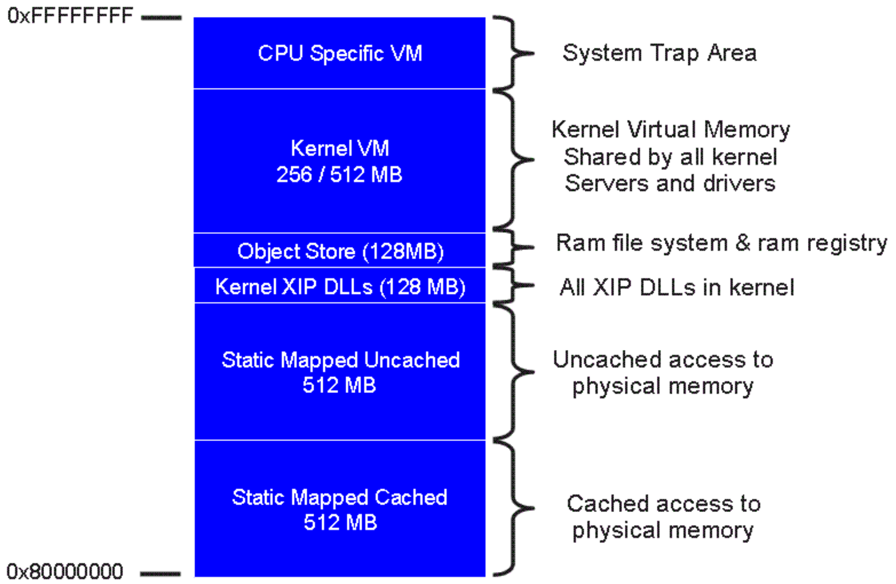
Others

Moving files from an image to another



CE6 Memory Management

- CE6 does not use “slots”
  - Each process has 1GB virtual memory
- Flashes are usually XIP, to save loading times
  - Most *drivers & frequently* used PE has fixed addresses



https://gist.github.com/udaken/f70b5a4c453fe64cb548a10dc85a27ed

CE6 & SDK: How it pack files

- Visual Studio + CE6 SDK
  - Everything is packed into B000FF format
  - Unessential segments, including .reloc is stripped
  - Optionally convert into .nb0
- cl.exe → link.exe → bundled image

requires .reloc

.reloc is stripped

What we want to do:

- Extract files using eimgfs and rebundling with our own environment
  - Access to KITL and WinDbg
  - Bundle our own files & tools
- Conclusion: .reloc must be reconstructed
  - .reloc is required for loader to edit addresses on the fly, should the binary is not loaded in originally intended address.
  - Image packer requires this information to write static addresses (binaries in .nb0/.bin have fixed addresses)

Our approach:
Static reconstruction of relocation information in PE

Our approach

- Try our best to reconstruct .reloc and make binaries work again
- Prior art: Dynamic analysis only [1]

[1] http://www.cs.columbia.edu/~vpappas/papers/reloc.raid14.pdf

Our approach

- We know where PE starts and where it ends
- Look for all addresses needs to be relocated, and re-write our .reloc segment.
  - ImageBase ~ (ImageBase+SizeOfImage)
- Brute-force search through entire binary
- .text (with code) and non-.text (without code) needs to be handled seperately

Our approach (code segment)

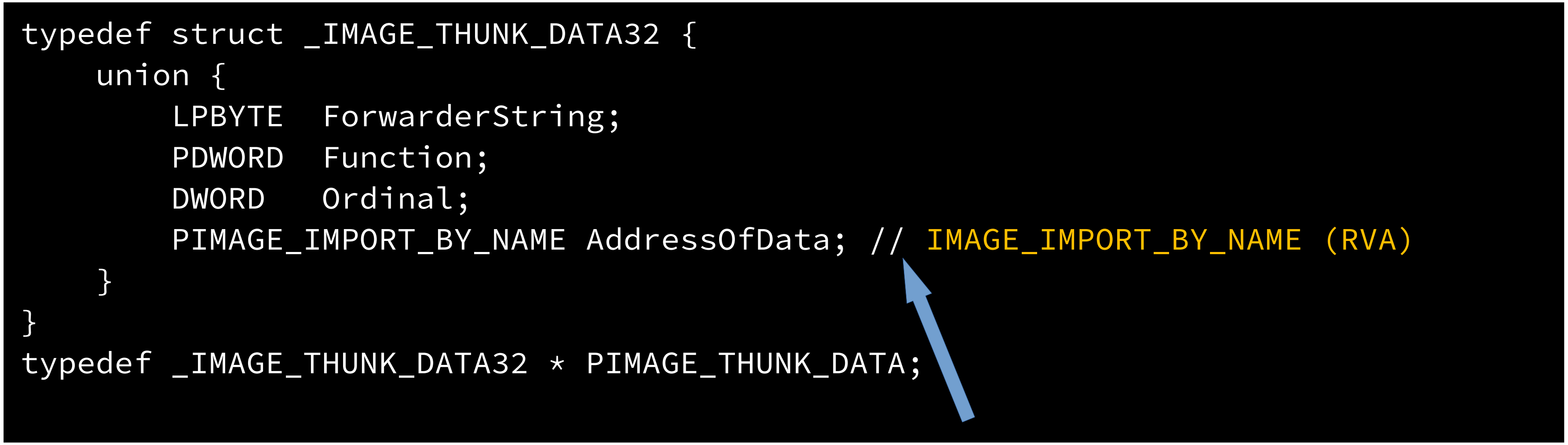
- Locate all function epilouge and prolouge
- Iterate through each function & check every instruction’s operand
  - If its referencing somewhere in the binary, relocate the address

Our approach (non-code segments)

- vtable, string tables, etc
- Conveniently 4-byte aligned
- Look for any 4-byte pointing into the PE

Our approach (quirks)

- It still doesn’t work… and missing a ton of .reloc entries
- Import Address Table



```
typedef struct _IMAGE_THUNK_DATA32 {
    union {
        LPBYTE  ForwarderString;
        PDWORD  Function;
        DWORD   Ordinal;
        PIMAGE_IMPORT_BY_NAME AddressOfData; // IMAGE_IMPORT_BY_NAME (RVA)
    }
}
typedef _IMAGE_THUNK_DATA32 * PIMAGE_THUNK_DATA;
```

AddressOfData can be char* and must be added to .reloc

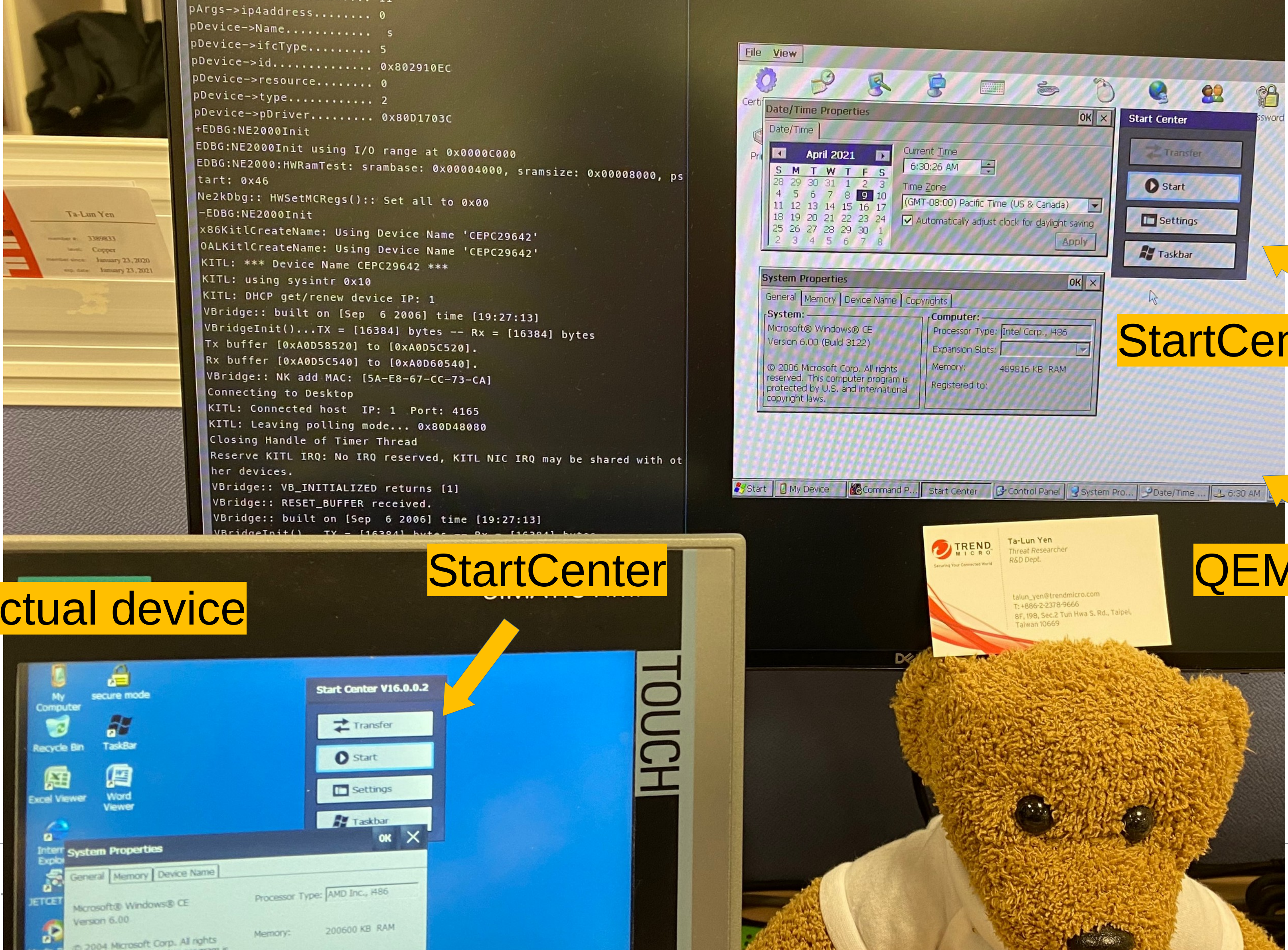
Our approach (finally)

- Rebuild our .reloc, and recompile our own CE image!

```
typedef struct _IMAGE_BASE_RELOCATION {
    DWORD   VirtualAddress;
    DWORD   SizeOfBlock;
//  WORD    TypeOffset[1];
} IMAGE_BASE_RELOCATION;


typedef struct {
  unsigned long  r_vaddr;   /* address of relocation      */
  unsigned long  r_symndx;  /* symbol we're adjusting for */
  unsigned short r_type;    /* type of relocation         */
} RELOC;   //COFF relocation table entry
```

Demo: We run your device without your hardware



```
pArgs->ip4address........ 0
pDevice->Name...........  s
pDevice->ifcType......... 5
pDevice->id............. 0x802910EC
pDevice->resource........ 0
pDevice->type........... 2
pDevice->pDriver......... 0x80D1703C
+EDBG:NE2000Init
EDBG:NE2000Init using I/O range at 0x0000C000
EDBG:NE2000:HWRamTest: srambase: 0x00004000, sramsize: 0x00008000, ps
tart: 0x46
Ne2kDbg:: HWSetMCRegs():: Set all to 0x00
-EDBG:NE2000Init
x86KitlCreateName: Using Device Name 'CEPC29642'
OALKitlCreateName: Using Device Name 'CEPC29642'
KITL: *** Device Name CEPC29642 ***
KITL: using sysintr 0x10
KITL: DHCP get/renew device IP: 1
VBridge:: built on [Sep  6 2006] time [19:27:13]
VBridgeInit()...TX = [16384] bytes -- Rx = [16384] bytes
Tx buffer [0xA0D58520] to [0xA0D5C520].
Rx buffer [0xA0D5C540] to [0xA0D60540].
VBridge:: NK add MAC: [5A-E8-67-CC-73-CA]
Connecting to Desktop
KITL: Connected host  IP: 1  Port: 4165
KITL: Leaving polling mode... 0x80D48080
Closing Handle of Timer Thread
Reserve KITL IRQ: No IRQ reserved, KITL NIC IRQ may be shared with ot
her devices.
VBridge:: VB_INITIALIZED returns [1]
VBridge:: RESET_BUFFER received.
VBridge:: built on [Sep  6 2006] time [19:27:13]
```

Date/Time Properties

Current Time: 6:30:26 AM

Time Zone: (GMT-08:00) Pacific Time (US & Canada)

☑ Automatically adjust clock for daylight saving

System Properties

System:
Microsoft® Windows® CE
Version 6.00 (Build 3122)

© 2006 Microsoft Corp. All rights reserved. This computer program is protected by U.S. and international copyright laws.

Computer:
Processor Type: Intel Corp., i486
Expansion Slots:
Memory: 489816 KB RAM
Registered to:

Start Center V16.0.0.2

System Properties

Microsoft® Windows® CE
Version 6.00

Processor Type: AMD Inc., i486
Memory: 200600 KB RAM

StartCenter

QEMU’d CE

StartCenter

Actual device

With our method...

- You can totally run bundled CE6 binaries without hardware!
- KITL, Serial outputs, WinDbg
- Around 98% accuracy (good enough to run)
  - Compared with original binary & reconstructed binary

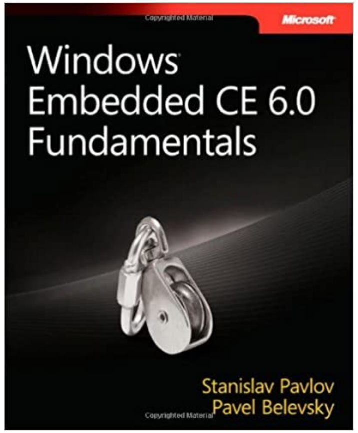
- We plan to open-source the tools we used later on

Suggestions for vendors & Remarks

- Anything bundled within firmware will be extracted & being looked at
- Proprietary format does not preventing breaking in
- Friendly community / researcher outreach is noble

Future work & Mentions

- Combine this with [insert any fuzzer here]
  - Yes, if ported to CE
  - For simple programs https://github.com/mauricek/wcecompat
- A good reference helps very much
- Thank you, MSFT, for shared-source initiative
  - It will be next to impossible to achieve this without it


Copyrighted Material

Microsoft

Windows
Embedded CE 6.0
Fundamentals

Stanislav Pavlov
Pavel Belevsky

Copyrighted Material

Questions?

- Send to “talun_yen at trendmicro dot com”