

The Future of RE:

# Dynamic Binary Visualization

{ christopher.domas // REcon 2013 // 22.06.2013 }

## & Myself

- ☒ Chris Domas
- ☒ Embedded Systems Engineer
- ☒ Cyber Innovation Unit
- ☒ National Security Division
- ☒ The Battelle Memorial Institute

## & Battelle

- ☒ World's largest non-profit R&D organization
- ☒ Manages leading national laboratories
- ☒ Awesome

# Myself

# Reverse Engineering

& Information Analysis

# In other words

& What IS this?

- How we conceptualize binary information
- How we use our conceptualization for analysis

# The ever-present issues

66	52	66	8B	D0	FA	B4	1B	D0	C0		push	dx
66	D1	C8	E6	74	E4	75	F6	C4	80		mov	dx, ax
74	08	86	C4	E6	74	E4	75	86	C4		cli	
8A	E6	66	5A	C3	66	52	66	8B	D0		mov	ah, 1Bh
86	C4	B4	1B	D0	C0	66	D1	C8	FA		rol	al, 1
66	50	8A	E0	8A	C2	86	C4	E6	74		ror	ax, 1
86	C4	E6	75	66	58	F6	C4	80	74		out	74h, al
06	86	C4	E6	74	E4	75	66	8B	C2		in	al, 75h
66	5A	C3	E6	72	E4	73	C3	66	50		test	ah, 80h
86	C4	E6	72	86	C4	E6	73	66	58		jz	\$+0xA
C3	66	9C	66	50	66	51	FA	66	8B		xchg	al, ah
CB	66	33	DB	8A	C1	E8	87	FF	FF		out	74h, al
FF	02	D8	80	D7	00	FE	C1	38	E9		in	al, 75h
76	EE	66	59	66	58	66	9D	C3	66		xchg	al, ah
50	66	53	80	E4	80	B3	10	0A	DC		mov	ah, dh

& Flexible  
 & Exact  
 & Complex

& Rigid  
 & Rules  
 & Succinct

# The RE Dichotomy

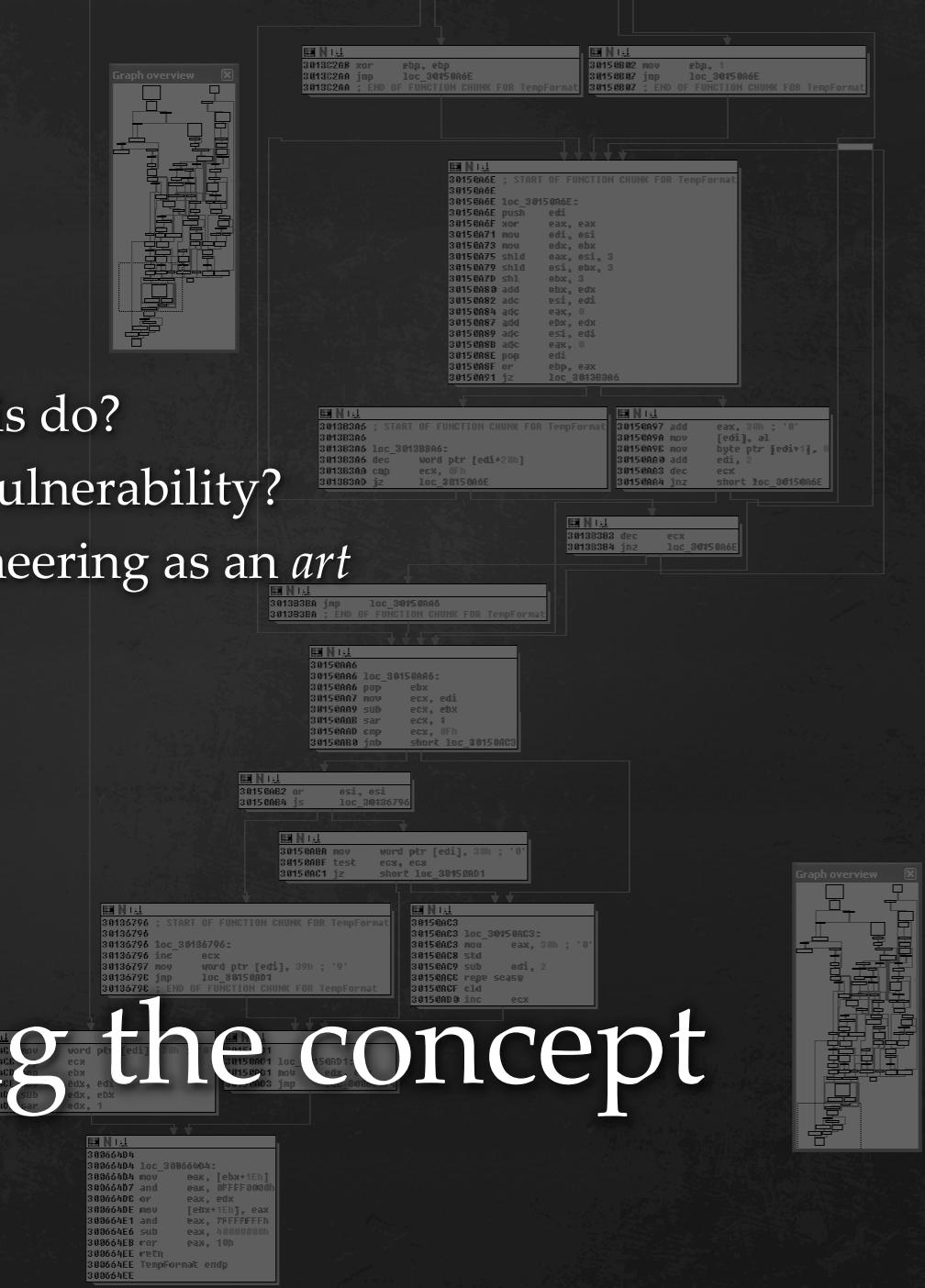
```

xor    edx, edx      ; int
lea    rcx, [rsp+78h] ; void *
       [rsp+arg_68], ebx
lea    r8d, [rdx+60h] ; size_t
call   memset
xor    ecx, ecx      ; hWnd
call   cs:GetDC
mov    rdi, rax
cmp    rax, rbx
jz    loc_1000015A4
lea    rsi, stru_1000101A0
mov    edx, 5Ah        ; int
mov    rcx, rax        ; HDC
mov    [rsp+arg_68], 68h
mov    [rsp+78h], rbp
mov    [rsp+88h], rsi
call   cs:GetDeviceCaps
mov    ecx, cs:nNumber
mov    r8d, 2D0h
edx, eax
call   cs:MulDiv
mov    rdx, rdi        ; hWnd
xor    ecx, ecx
eax
mov    dword ptr [rsp+94h], 1000041h
mov    eax, 2000h
[rsp+0C8h], ax
call   cs:ReleaseDC
call   cs:ChooseFontW
cmp    eax, ebx
jz    loc_1000015A4
rcx, cs:hCursor ; hCursor
call   cs:SetCursor
rcx, rsi        ; LOGFONTW *
call   cs>CreateFontIndirectW
rdi
rax, rbx
short loc_100001B8C
mov    rcx, cs:wParam ; HGDIOBJ
call   cs>DeleteObject
rcx, cs:qword_1000100A0 ; hWnd
r9, r14        ; lParam
mov    r8, rdi        ; wParam
mov    edx, 30h        ; Msg
mov    cs:wParam, rdi

```

- & What does this do?
- & Where's the vulnerability?
- & Reverse engineering as an *art*

# Investigating the concept



# Reverse Engineering



Let's fix this.

- ⌘ Bridge the analysis gap
- ⌘ Change RE from an ART to a SCIENCE

& Visual RE

❖ Address the conceptualization  
& Statistical Exploration

❖ Address the analysis

How?

## & Idea:

- ⌘ Take a computationally difficult task
- ⌘ Translate it to a problem our brains do naturally

# Visual RE

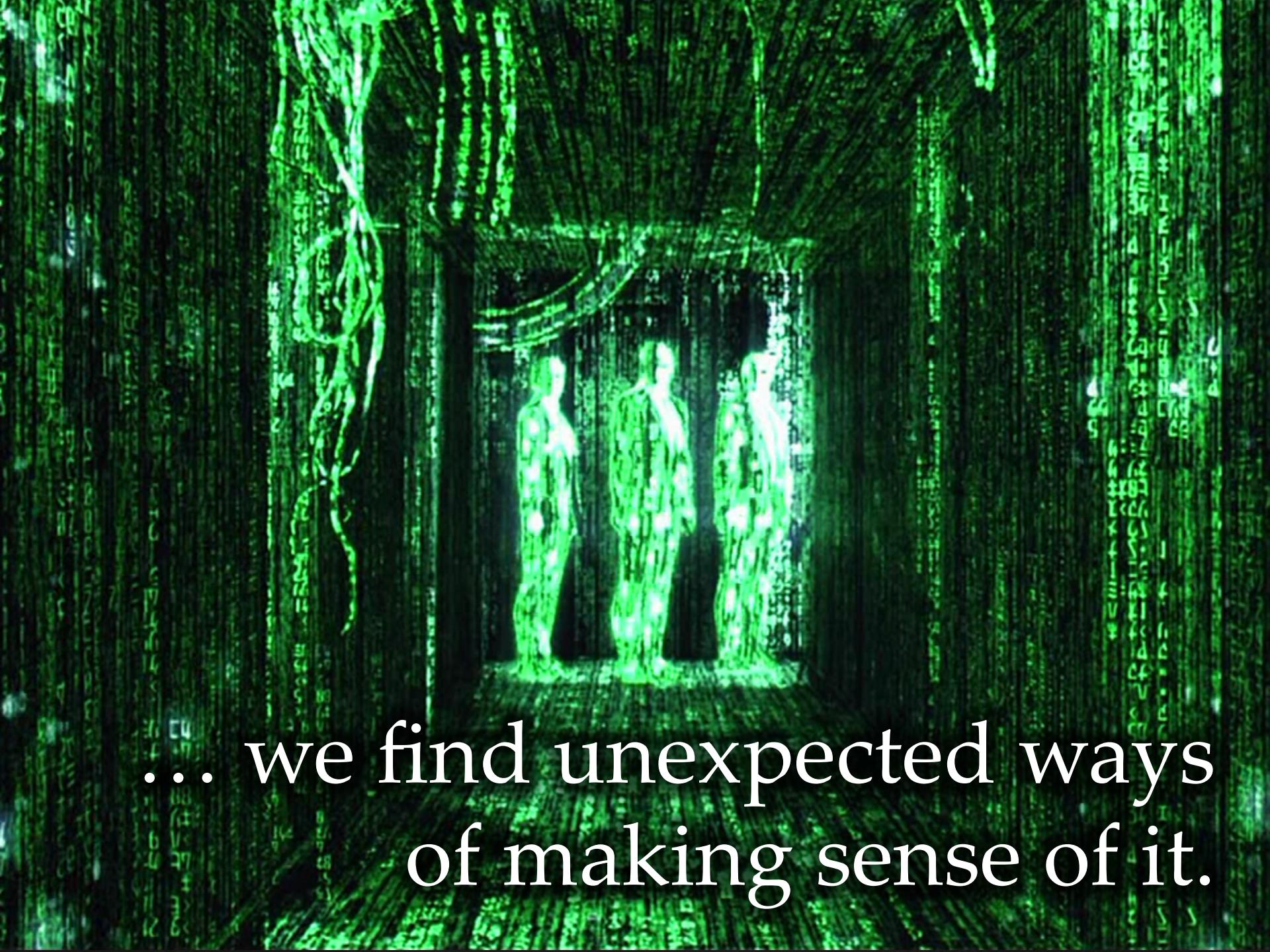
☞ In other words

☞ how to traverse a hundred thousand pages of

in 20 seconds

•...what?

If we change the way we process binary information...

A scene from the movie The Matrix. Neo, played by Keanu Reeves, stands in front of a large server unit. He is looking at the floor, and his body is slightly hunched over. The background is filled with a dense, vertical rain of green binary code (0s and 1s) that falls across the entire frame. The server unit behind him has several circular ports and a control panel with various buttons and lights.

... we find unexpected ways  
of making sense of it.

- & obfuscated file headers
- & no headers
- & multiple binaries embedded in a single blob
- & unique instruction sets
- & proprietary data formats
- & overwhelming complexity
- & steganography
- & memory dumps
- & rapid RE
- & triage
- & firmware
- & forensics

# Why bother?

- ¶ Our best RE tools are completely dependent on known structure
  - ☒ Information is evolving faster than our tools can keep up
- ¶ Gate's Law
  - ☒ Software is getting slower more rapidly than hardware becomes faster
  - ☒ The amount of information we need to analyze is growing exponentially

# Why bother?

& Greg Conti

✉ United States Military Academy

✉ Black Hat 2010

& Aldo Cortesi

✉ Nullcube

✉ corte.si

# Some interesting ideas

& Idea:

- ☒ Even in unstructured data, there is structure
- ☒ Sequential bytes have an implicit relationship

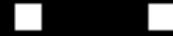
# Digraphs (conti)

## **recon**

re	(72, 65)
ec	(65, 72)
co	(63, 6F)
on	(6F, 6E)

# Digraphs (conti)

(0x20, 0x20)



(0x20, 0x35 )

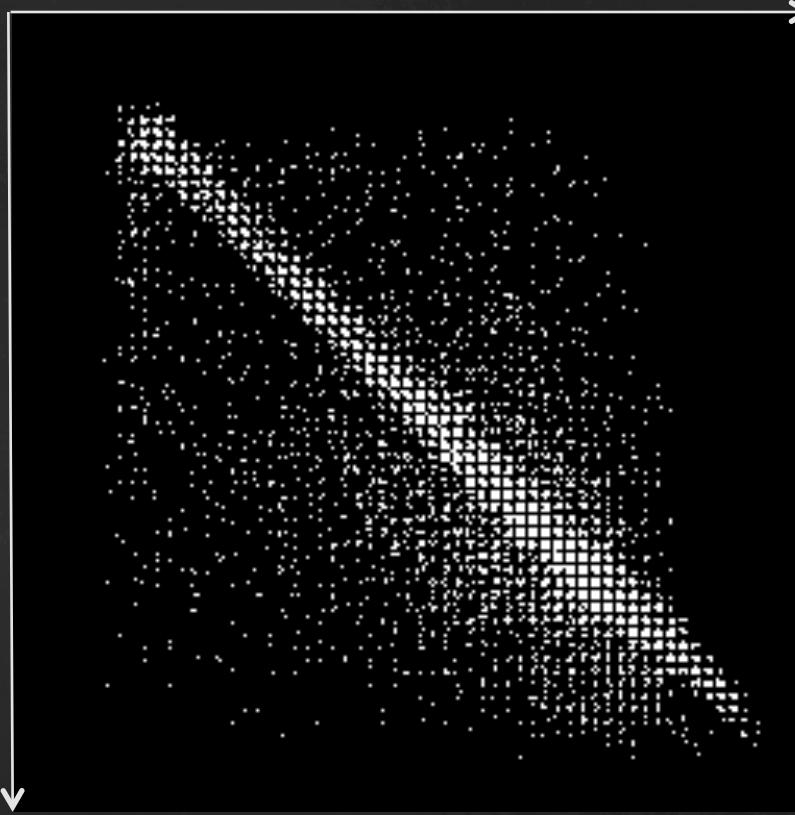
# Digraphs (conti)

& ASCII



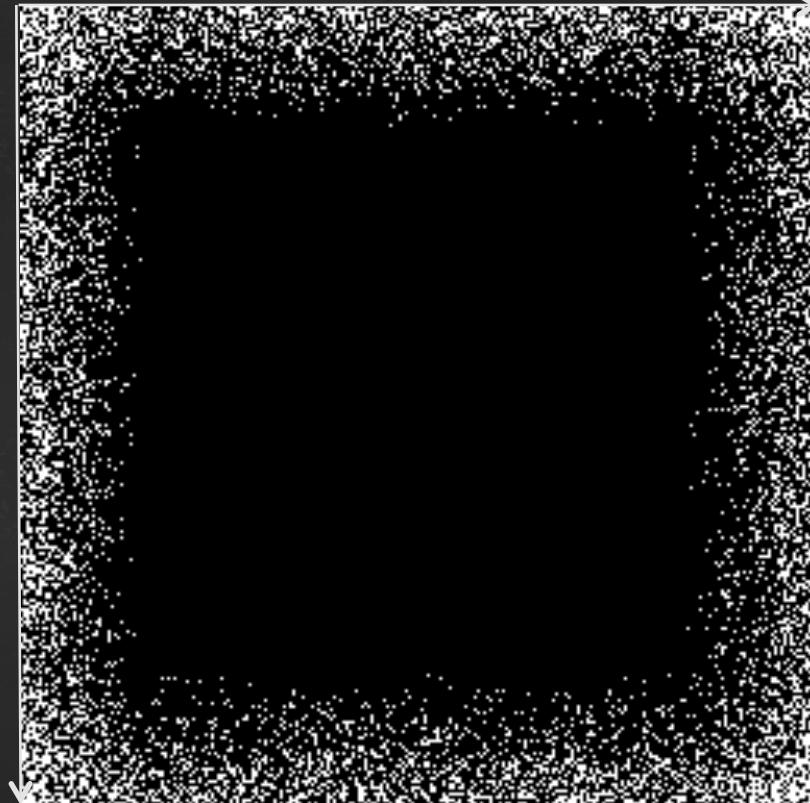
# Digraphs (contd)

& Image



# Digraphs (cont)

& Audio



Digraphs (conti)

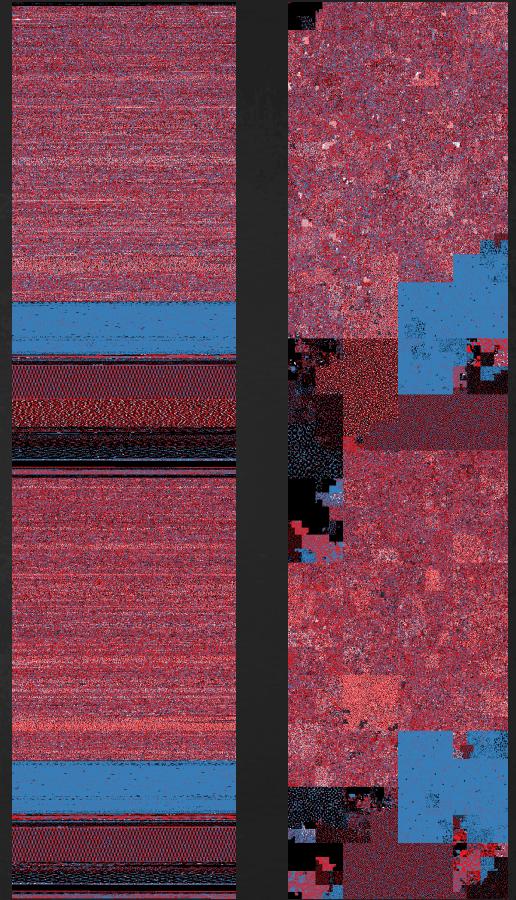
## & Introducing the Hilbert Curve



- & Continuous fractal space filling curve
- & Maps 1D space to an alternate dimensional space
- & *Preserves Locality*
- & Unexpected utility in computer science

# Hilbert Translation (cortesi)

- Traditional byte plot “Zig-Zag” order on left
- Hilbert byte plot on right
- Shaded by Byte Class



# Hilbert Translation (cortesi)

- Build on these concepts
- Goal
  - understand data
  - *independent of format*
    - Unknown code
    - Format deviations
    - Proprietary structure
  - a priori

# Our software

- ☒ ..cantor.dust..
- ☒ Named after this guy →
- ☒ Illustrating concepts
  - ☒ Doesn't need to be this software

# Introducing ..cantor.dust..



& An embarrassing mistake

..cantor.dust.. // background

66	52	66	8B	D0	FA	B4	1B	D0	C0	66	D1	C8	E6	74	E4	75	F6	C4	80	74	08	86	C4	E6	74	E4	75	86
C4	8A	E6	66	5A	C3	66	52	66	8B	D0	86	C4	B4	1B	D0	C0	66	D1	C8	FA	66	50	8A	E0	8A	C2	86	C4
E6	74	86	C4	E6	75	66	58	F6	C4	80	74	06	86	C4	E6	74	E4	75	66	8B	C2	66	5A	C3	E6	72	E4	73
C3	66	50	86	C4	E6	72	86	C4	E6	73	66	58	C3	66	9C	66	50	66	51	FA	66	8B	CB	66	33	DB	8A	C1
E8	87	FF	FF	FF	02	D8	80	D7	00	FE	C1	38	E9	76	EE	66	59	66	58	66	9D	C3	66	50	66	53	80	E4
80	B3	10	0A	DC	66	52	66	8B	D0	FA	B4	1B	D0	C0	66	D1	C8	E6	74	E4	75	F6	C4	80	74	08	86	C4
E6	74	E4	75	86	C4	8A	E6	66	5A	C3	66	52	66	8B	D0	86	C4	B4	1B	D0	C0	66	D1	C8	FA	66	50	8A
E0	8A	C2	86	C4	E6	74	86	C4	E6	75	66	58	F6	C4	80	74	06	86	C4	E6	74	E4	75	66	8B	C2	66	5A
C3	E6	72	E4	73	C3	66	50	86	C4	E6	72	86	C4	E6	73	66	58	C3	66	9C	66	50	66	51	FA	66	8B	CB
66	33	DB	8A	C1	E8	87	FF	FF	FF	02	D8	80	D7	00	FE	C1	38	E9	76	EE	66	59	66	58	66	9D	C3	66
50	66	53	80	E4	80	B3	10	0A	DC	66	52	66	8B	D0	FA	B4	1B	D0	C0	66	D1	C8	E6	74	E4	75	F6	C4
80	74	08	86	C4	E6	74	E4	75	86	C4	8A	E6	66	5A	C3	66	52	66	8B	D0	86	C4	B4	1B	D0	C0	66	D1
C8	FA	66	50	8A	E0	8A	C2	86	C4	E6	74	86	C4	E6	75	66	58	F6	C4	80	74	06	86	C4	E6	74	E4	75
66	8B	C2	66	5A	C3	E6	72	E4	73	C3	66	50	86	C4	E6	72	86	C4	E6	73	66	58	C3	66	9C	66	50	66
51	FA	66	8B	CB	66	33	DB	8A	C1	E8	87	FF	FF	FF	02	D8	80	D7	00	FE	C1	38	E9	76	EE	66	59	66
58	66	9D	C3	66	50	66	53	80	E4	80	B3	10	0A	DC	66	52	66	8B	D0	FA	B4	1B	D0	C0	66	D1	C8	E6
74	E4	75	F6	C4	80	74	08	86	C4	E6	74	E4	75	86	C4	8A	E6	66	5A	C3	66	52	66	8B	D0	86	C4	B4
1B	D0	C0	66	D1	C8	FA	66	52	66	8B	D0	FA	B4	1B	D0	C0	66	D1	C8	E6	74	E4	75	F6	C4	80	74	08
86	C4	E6	74	E4	75	86	C4	8A	E6	66	5A	C3	66	52	66	8B	D0	86	C4	B4	1B	D0	C0	66	D1	C8	FA	66
50	8A	E0	8A	C2	86	C4	E6	74	86	C4	E6	75	66	58	F6	C4	80	74	06	86	C4	E6	74	E4	75	66	8B	C2
66	5A	C3	E6	72	E4	73	C3	66	50	86	C4	E6	72	86	C4	E6	73	66	58	C3	66	9C	66	50	66	51	FA	66
8B	CB	66	33	DB	8A	C1	E8	87	FF	FF	FF	02	D8	80	D7	00	FE	C1	38	E9	76	EE	66	59	66	58	66	9D
C3	66	50	66	53	80	E4	80	B3	10	0A	DC	66	52	66	8B	D0	FA	B4	1B	D0	C0	66	D1	C8	E6	74	E4	75
F6	C4	80	74	08	86	C4	E6	74	E4	75	86	C4	8A	E6	66	5A	C3	66	52	66	8B	D0	86	C4	B4	1B	D0	C0
66	D1	C8	FA	66	50	8A	E0	8A	C2	86	C4	E6	74	86	C4	E6	75	66	58	F6	C4	80	74	06	86	C4	E6	74

..cantor.dust.. // background

- & An embarrassing mistake
- & Spare time

..cantor.dust.. // background

& Interface  
& Patterns

..cantor.dust.. // demo

- ¶ Visualizations
- ¶ Tuple systems
  - ↳ Why
  - ↳ Time/Space
  - ↳ Coordinate Systems
  - ↳ Fun

..cantor.dust.. // demo

& Visualizations

& Metric map

& Entropy

& Uses

& Encryption Keys

& Packing

& Obfuscation

& Examples

& Malware

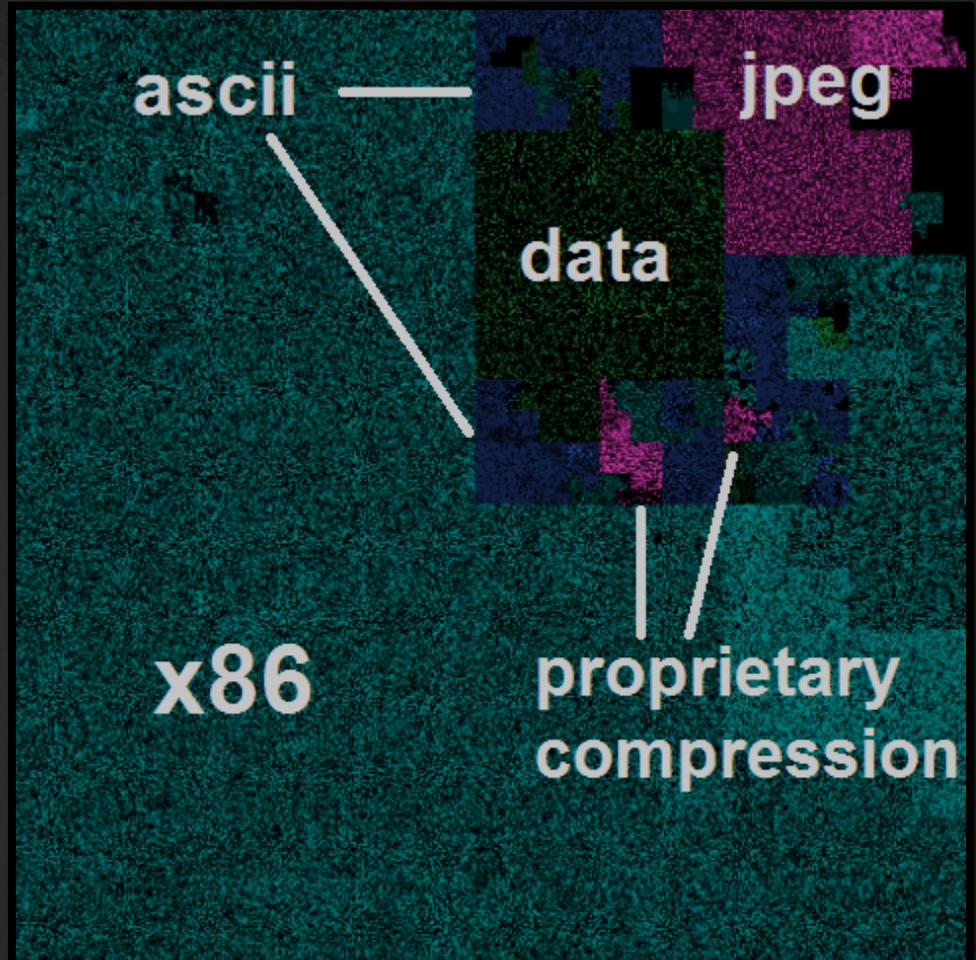
& Key check

..cantor.dust.. // demo

& Notepad.exe dissection

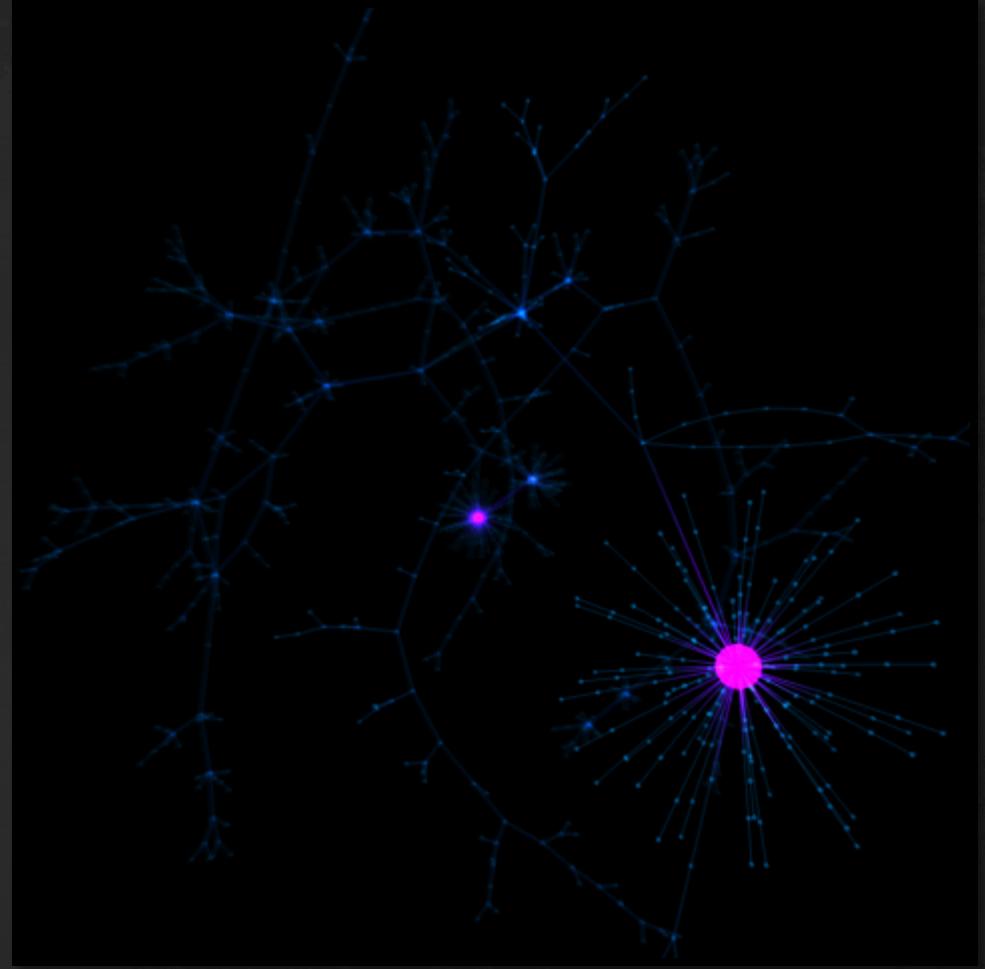
..cantor.dust.. // demo

- ❖ Binary region identification through statistical analysis
  - ❖ Naïve Bayes classification of n-gram models
  - ❖ Identify regions of an object based on supplied templates
- ❖ Uses:
  - ❖ Custom/proprietary formats
  - ❖ Unique Instruction Sets
  - ❖ Data/code features



..cantor.dust.. // classificaton

- ❖ Conventional parsing
  - ❖ Recursive descent
  - ❖ Linear sweep
- ❖ A new approach?
  - ❖ “Probabilistic parsing”
  - ❖ Identify structure based on statistical *patterns*, not grammar definitions



..cantor.dust.. // parsing

¶ “Attacking Intel BIOS”

¤ Invisible Things Lab

¤ Rafal Wojtczuk and Alexander Tereshkin

¤ Black Hat 2009

..cantor.dust.. // case study

& Goal:

☒ Bypass EFI signing protection

..cantor.dust.. // case study

## & Background:

- ⌘ EFI is a BIOS replacement
- ⌘ Update images can be signed and checked before they are flashed

..cantor.dust.. // case study

## & Background

- ☒ Update image “envelope”
  - ☒ Signed module
  - ☒ Signed module
  - ☒ Signed module
  - ☒ Unsigned module

..cantor.dust.. // case study

## & Background

- ☒ Update image “envelope”
  - ☒ Signed module
  - ☒ Signed module
  - ☒ Signed module
  - ☒ Unsigned module ← Attack Vector!

..cantor.dust.. // case study

- ↳ Why have an unsigned module?
  - ↗ Boot Splash Logo
  - ↗ Can be changed by the OEMs

..cantor.dust.. // case study

## & Exploit!

- ☒ Goal: Unsigned Code Execution
  - ☒ Can't update code, can update bitmap
- ☒ Vulnerability identified in bitmap parser for splash screen
  - ☒ In EFI template code used by most IBVs
- ☒ Flash the bitmap with an invalid image
  - ☒ Overflow with width & height
- ☒ Cause an overflow, get execution

..cantor.dust.. // case study

## & Observations

- ☒ No one is really “EFI”
  - ☒ There’s just “more” or “less” EFI for now
  - ☒ This means
    - ☒ Less structure
    - ☒ More headache

..cantor.dust.. // case study

& ITL covered the exploit  
& Let's figure out everything else

..cantor.dust.. // case study

# case study // step 1

- ❖ Choose a victim
  - ❖ Targeting my junk laptop
  - ❖ Don't care if I brick it
- ❖ Grab the update executable from vendor's website



## & Quick RE to extract the firmware image

- ❖ Ultra-secret secure flags have been encrypted by adding 1 to each character
- ❖ We're looking for "WRITEROMFILE"
- ❖ a.k.a. "XSJUFSPNGJMF"

'S'	.rdata:00430A94	0000000B	C	USER32.dll
'S'	.rdata:00430BDC	0000000D	C	ADVAPI2.dll
'S'	.data:00431030	00000008	C	DMBTJD
'S'	.data:00431038	0000000D	C	SFQPSUTUBUVT
'S'	.data:00431048	0000000D	C	OPSCVSFTVMUT
'S'	.data:00431058	0000000B	C	SCVSFTVMUT
'S'	.data:00431064	0000000E	C	XSJUFLSPNGJMF
'S'	.data:00431074	0000000D	C	XSJUFSPNGJMF
'S'	.data:00431084	0000000D	C	XSJUFIFYGJMF
'S'	.data:00431094	0000000D	C	XSJUFIESGJMF
'S'	.data:004310A4	0000000A	C	XJQFDMFB0
'S'	.data:004310B0	00000008	C	XJQFBMM
'S'	.data:004310B8	00000008	C	WFSCPFT
'S'	.data:004310C0	00000008	C	QSHCPPU
'S'	.data:004310C8	00000009	C	OPSFCPPU
'S'	.data:004310D4	00000008	C	OPQBVTF
'S'	.data:004310DC	00000005	C	JOGP
'S'	.data:004310E4	0000000A	C	GPSDFUZQF
'S'	.data:004310F0	00000008	C	GPSDFJU
'S'	.data:004310F8	00000008	C	GBDUPSZ
'S'	.data:00431100	00000005	C	FEWV

# case study // step 2

## ¶ RE image

- ☒ Find a custom decompression routine
- ☒ But only one module
- ☒ Where are the others?

```
decomp_module proc near
    push    edi
    add     edx, esi

loc_FFFF90D7:
    xor     ecx, ecx
    cmp     esi, edx
    jnb    short loc_FFFF9140
    lodsb
    test   al, 0Fh
    jnz    short loc_FFFF9108
    cmp     al, 0E0h ; 'a'
    jz     short loc_FFFF90FC
    cmp     al, 0F0h ; '='
    jz     short loc_FFFF90F4
    shr     al, 4
    mov     cl, al
    inc     ecx

loc_FFFF90F0:
    rep    movsb
    jmp    short loc_FFFF90D7
```

# case study // step 3

## & Use ..cantor.dust..

- ☒ Crop out the known module
- ☒ Use as a template for statistical analysis
- ☒ Identify regions of the binary image with patterns that match the template



case study // step 4

- ¶ Decompress the located modules
  - ☒ Use Chris Eagle's x86emu
  - ☒ No need to understand the algorithm

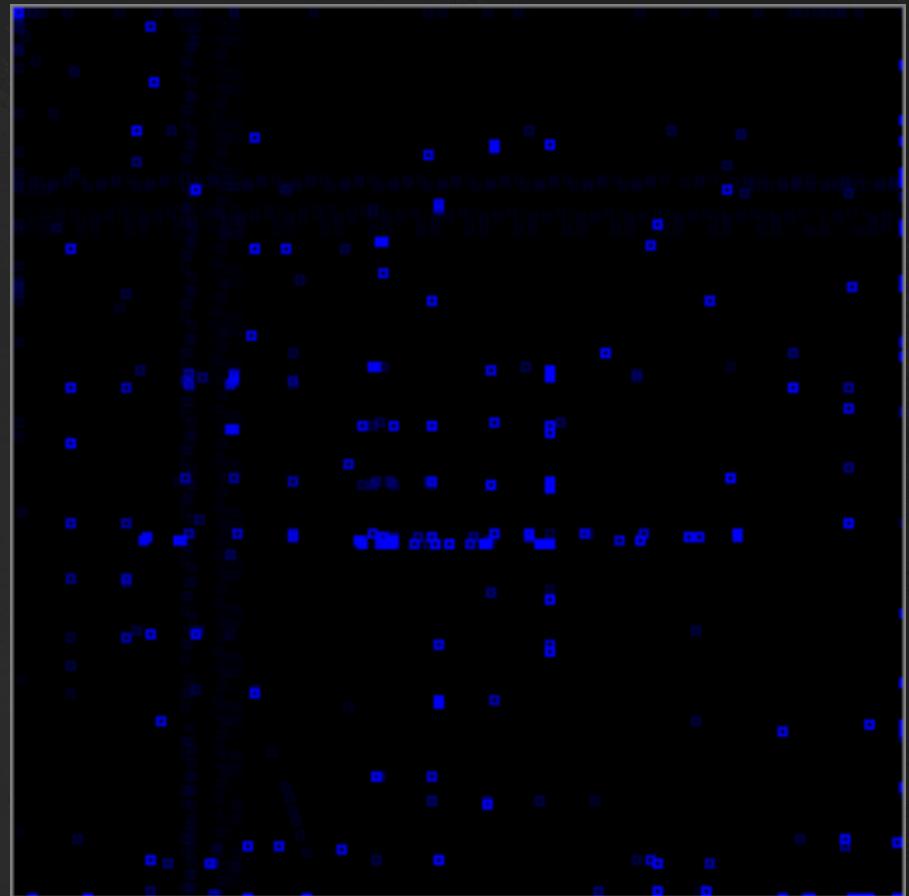
case study // step 5

& Modules contain  
proprietary headers

case study // step 6

```
root@deltaleph-ubuntu:~/modules# file *
mod_01: data
mod_02: data
mod_03: data
mod_04: DOS executable (COM)
mod_07: data
mod_09: 8086 relocatable (Microsoft)
mod_0c: data
mod_0d: data
mod_0e: data
mod_0f: data
mod_10: data
mod_11: data
mod_13: data
mod_14: data
mod_16: data
mod_19: data
mod_1a: data
mod_25: data
mod_3c: data
mod_3f: data
mod_40: data
mod_41: data
mod_42: data
mod_43: data
mod_45: data
mod_48: data
mod_49: data
```

- ❖ How do we find the splash screen?
  - ❖ Too time consuming at a binary level!
  - ❖ Use visual abstractions to locate the “bitmap data” fingerprint



case study // step 6

- Examine in a hex editor – follows the bitmap format, minus the 'BM' signature
- Easy to find width/height

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
00 0h:	C8	5F	00	00	00	00	00	76	00	00	00	28	00	00	00	E_.....v....(...	
00 1h:	90	01	00	00	7A	00	00	00	01	00	04	00	00	00	00	.....z.....	
00 2h:	00	00	00	00	3E	0B	00	00	3E	0B	00	00	00	00	00	.....>...>.....	
00 3h:	00	00	00	00	04	01	00	00	07	02	00	00	D5	49	00	00	
00 4h:	87	2E	00	00	55	1D	00	00	1D	0A	00	00	F2	53	00	00	
00 5h:	ED	51	00	00	E9	50	00	00	E6	4F	00	00	E2	4E	00	00	
00 6h:	DF	4D	00	00	B3	3D	00	00	2E	10	00	00	0E	05	00	00	
00 7h:	00	00	00	00	FF	.....VVVVVVVVVVVVV											
00 8h:	FF	VVVVVVVVVVVVVVVV															
00 9h:	FF	VVVVVVVVVVVVVVVV															
00 Ah:	FF	VVVVVVVVVVVVVV															
00 Bh:	FF	VVVVVVVVVVVVVV															
00 Ch:	FF	FF	FF	FF	FF	FF	FD	C4	0F	FF	FF	FF	FF	FF	FF	VVVVVVVVVVVVVV	
00 Dh:	FF	VVVVVVVVVVVVVV															
00 Eh:	FF	VVVVVVVVVVVVVV															
00 Fh:	FF	VVVVVVVVVVVVVV															
01 0h:	FF	VVVVVVVVVVVVVV															
01 1h:	FF	VVVVVVVVVVVVVV															
01 2h:	FF	VVVVVVVVVVVVVV															
01 3h:	FF	VVVVVVVVVVVVVV															
01 4h:	FF	VVVVVVVVVVVVVV															
01 5h:	FF	VVVVVVVVVVVVVV															

case study // step 7

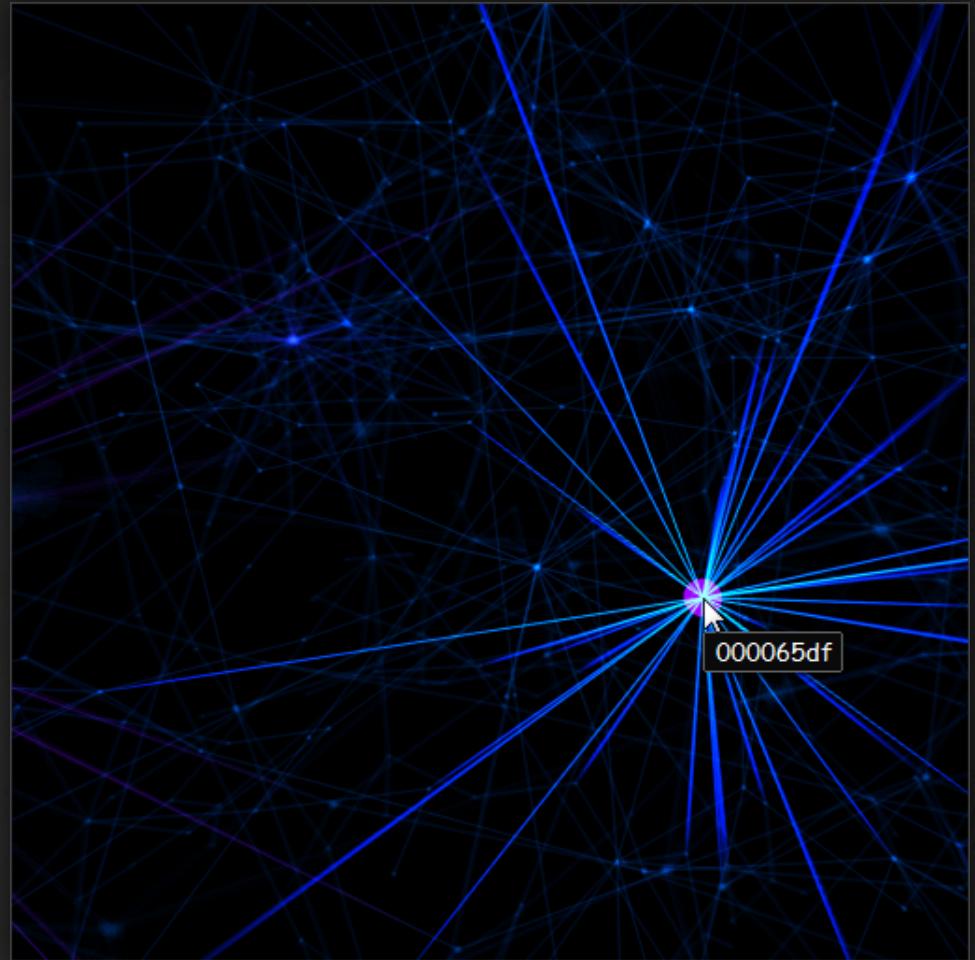
# case study // step 8

& Repackage the splash screen with modified information

- ¶ Try to flash
- ¶ Image is not accepted
  - ☒ Modules are signed
  - ☒ Bitmap is not
  - ☒ “Envelope” is checksum’d
  - ☒ Need to fix the checksum
- ¶ Drop first module into IDA
  - ☒ There’s no known structure to this module
  - ☒ IDA tells us nothing

case study // step 9

- Use probabilistic parsing to identify key functions
- Find debug\_printf
- Find calls to debug\_printf
- Isolate CRC checksum routine
  - Identify CRC algorithm
  - Locate checksum position in image



case study // step 10

- Repackage the image with proper checksum
- Success!

case study // step X



# case study // results

¶ Without ..cantor.dust..?

¤ Took me 37 hours

case study // results

& Hope I've shown...

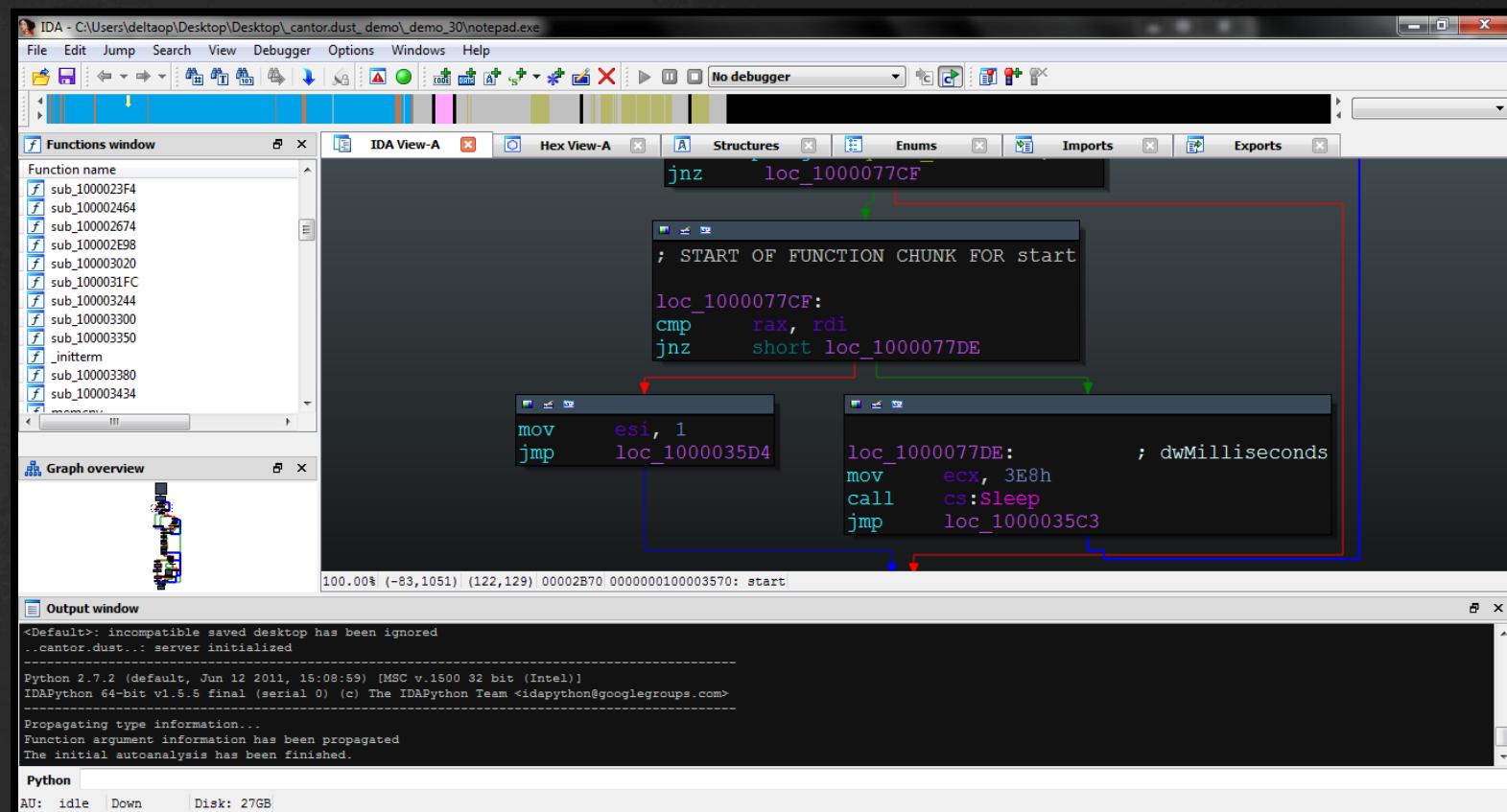
& Useful:  
☒ User interface

..cantor.dust.. // future

& Useful:

☒ Complete integration

..cantor.dust.. // future



..cantor.dust.. // future

IDA - C:\Users\delta10p\Desktop\Desktop\_cantor.dust\_demo\_\demo\_30\notepad.exe

File Edit Jump Search View Debugger Options Windows Help

No debugger

Imports      Exports

IDB View-A

```
.rdata:000000010000EF5A db 0
.rdata:000000010000EF5B db 0
.rdata:000000010000EF5C db 30h ; 0
.rdata:000000010000EF5D db 4
.rdata:000000010000EF5E db 0
.rdata:000000010000EF5F db 0
.rdata:000000010000EF60 db 0
.rdata:000000010000EF61 db 0
.rdata:000000010000EF62 db 0
.rdata:000000010000EF63 db 0
.rdata:000000010000EF64 db 0
.rdata:000000010000EF65 db 0
.rdata:000000010000EF66 db 0
.rdata:000000010000EF67 db 0
.rdata:000000010000EF68 db 1
.rdata:000000010000EF69 db 12h
.rdata:000000010000EF6A db 5
.rdata:000000010000EF6B db 0
.rdata:000000010000EF6C db 12h
.rdata:000000010000EF6D db 42h ; B
.rdata:000000010000EF6E db 0EH
.rdata:000000010000EF6F db 70h ; p
.rdata:000000010000EF70 db 0DH
.rdata:000000010000EF71 db 60h ; ` 
.rdata:000000010000EF72 db 0Ch
```

0000DD5F 000000010000EF5F: .rdata:000000010000EF5F

AU: idle Down Disk: 27GB

..cantor.dust.. // future

IDA - C:\Users\delta10p\Desktop\Desktop\_cantor.dust\_demo\_30\notepad.exe

File Edit Jump Search View Debugger Options Windows Help

No debugger

IDA View-A, Structures, Hex View-A Imports Exports

IDA View-A

Address	Value
100001000	100001000
100001200	100001200
100001234	100001234
100001364	100001364
1000013E0	1000013E0
100001500	100001500
100001690	100001690
10000180C	10000180C
100001DD0	100001DD0
100001F70	100001F70
100002054	100002054
1000021D8	1000021D8

Line 1 of 104

0000DAE1 000000010000ECE1: .rdata:000000010000ECE1

Imports Exports

37. tagMSG:00000000

0000DAE1 000000010000ECE1: .rdata:000000010000ECE1

The screenshot shows the IDA Pro debugger interface. The main window displays assembly code in the center, with memory dump and network analysis views on the left and right respectively. A status bar at the bottom indicates the current state.

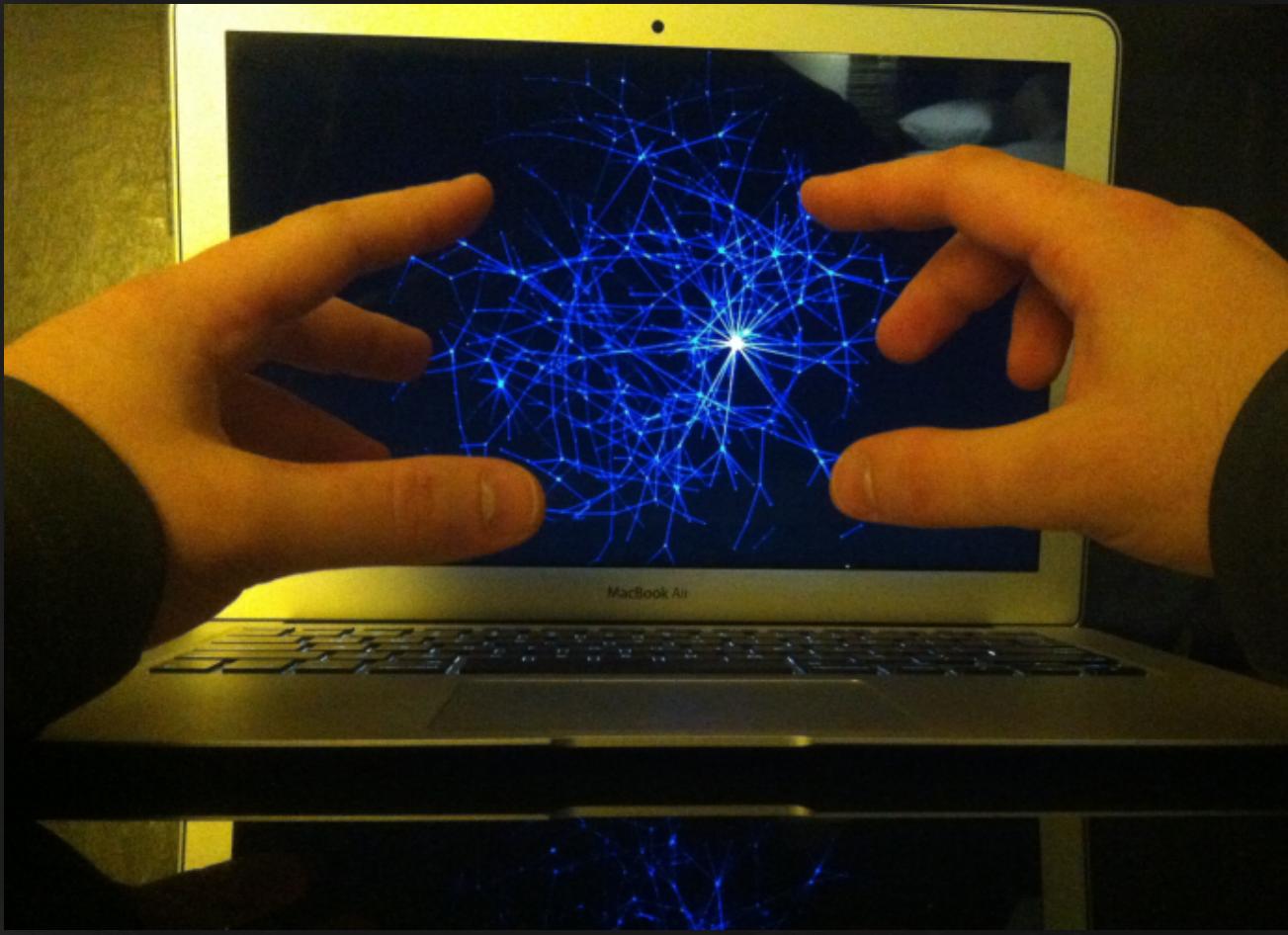
..cantor.dust.. // future

& Pointless:  
☒ Hands free

..cantor.dust.. // future



..cantor.dust.. // future



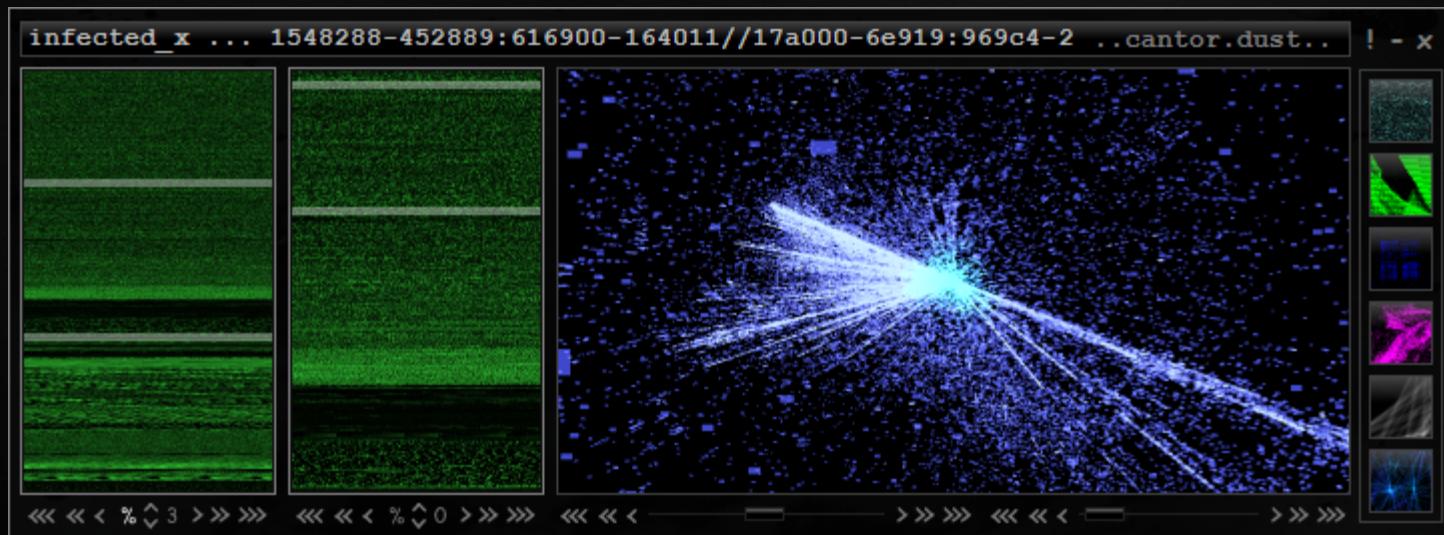
..cantor.dust.. // future

A promotional image for the movie Minority Report. Tom Cruise's character, John Anderton, is shown from the waist up, wearing a black t-shirt. He has his arms raised, palms facing forward, with glowing blue energy rings around his fingers. The background is a dark, futuristic setting with blurred digital displays showing numbers like '22:100' and '00:1'.

..cantor.dust.. // future

..minority.report..

- ¶ We need new ways to understand and analyze information
- ☒ Otherwise RE will stagnate



# Conclusions

# Conclusions

- ¶ We need new ways to understand and analyze information
  - ☒ Otherwise RE will stagnate
- ¶ Use ~~this software~~
- ¶ Start thinking differently

& Getting a copy  
✉ [sites.google.com/site/xxcantorxdustxx/home](http://sites.google.com/site/xxcantorxdustxx/home)  
✉ [the.delta.axiom@gmail.com](mailto:the.delta.axiom@gmail.com)  
✉ [hammera@battelle.org](mailto:hammera@battelle.org)  
& We've just gotten started

Thank you

..cantor.dust.. // redux

