Inside the Matrix,How to Build Transparent Sandbox for Malware Analysis

C.K Chen (Bletchley)

Who am I

- C.K Chen (陳仲寬)
  - P.H.D Student in DSNS Lab, NCTU
  - Research in
    - Reverse Engineering
    - Malware Analysis
    - Virtual Machine

About DSNS

- 謝續平教授
- 實驗室研究方向
  - 惡意程式分析
  - 虛擬機器
  - 數位鑑識
  - 網路安全

Outline

- VM for Malware Analysis
- Detect Security Utilities
- Out-of-Box Monitor
  - Emulation
  - Virtualization
- Malware Behavior Analysis
- Dynamic Taint Tracking
- Cloudebug
- Anti-VM
- Behavior Comparison to Detect Anti-VM

VM for Malware Analysis

- VM play an important role for nowadays for malware analysis
  - Isolated Environment
  - Fast Recovery

Reverse with VM

- What we are doing everyday
  - Automatic analysis malware:
    - Put monitor program into VM to keep track of malware
  - Reversing Malware
    - Put reversing tools(debugger, disassembler) into VM and reversing

Detect Security Utilities

- While your security utilities are placed in the same environment, it is possible for malware to detect it's existence
  - KillAV
  - Anti-Debugger

Kill AV

- Malware can check the existence of anti-virus, and then stop or bypass anti-virus
  - Process Name
  - If important function being hooked
  - Read Process Memory

- Any software in the same environment with malware can be detected

```
void block(){
char *apps[]={"avp.com","avp.exe","egui.exe","ekrn.exe",
    "mseinstall.exe","msseces.exe","MsMpEng.exe","DoScan.exe",
"defwatch.exe","360Safebox.exe","360tray.exe","McInst.exe",
    "ccapp.exe","CCenter.exe","ccEvtMgr.exe","ccSetMgr.exe"};
while(1){
BlockApp(apps[0]);
BlockApp(apps[1]);
BlockApp(apps[2]);
BlockApp(apps[3]);
BlockApp(apps[4]);
BlockApp(apps[5]);
BlockApp(apps[6]);
BlockApp(apps[7]);
BlockApp(apps[8]);
```

Anti-Debug

- To confuse analyst, malware employ anti-debug to detect or stop debug software

- Everything you put into VM expose the threat
  - File
  - Process
  - Registry

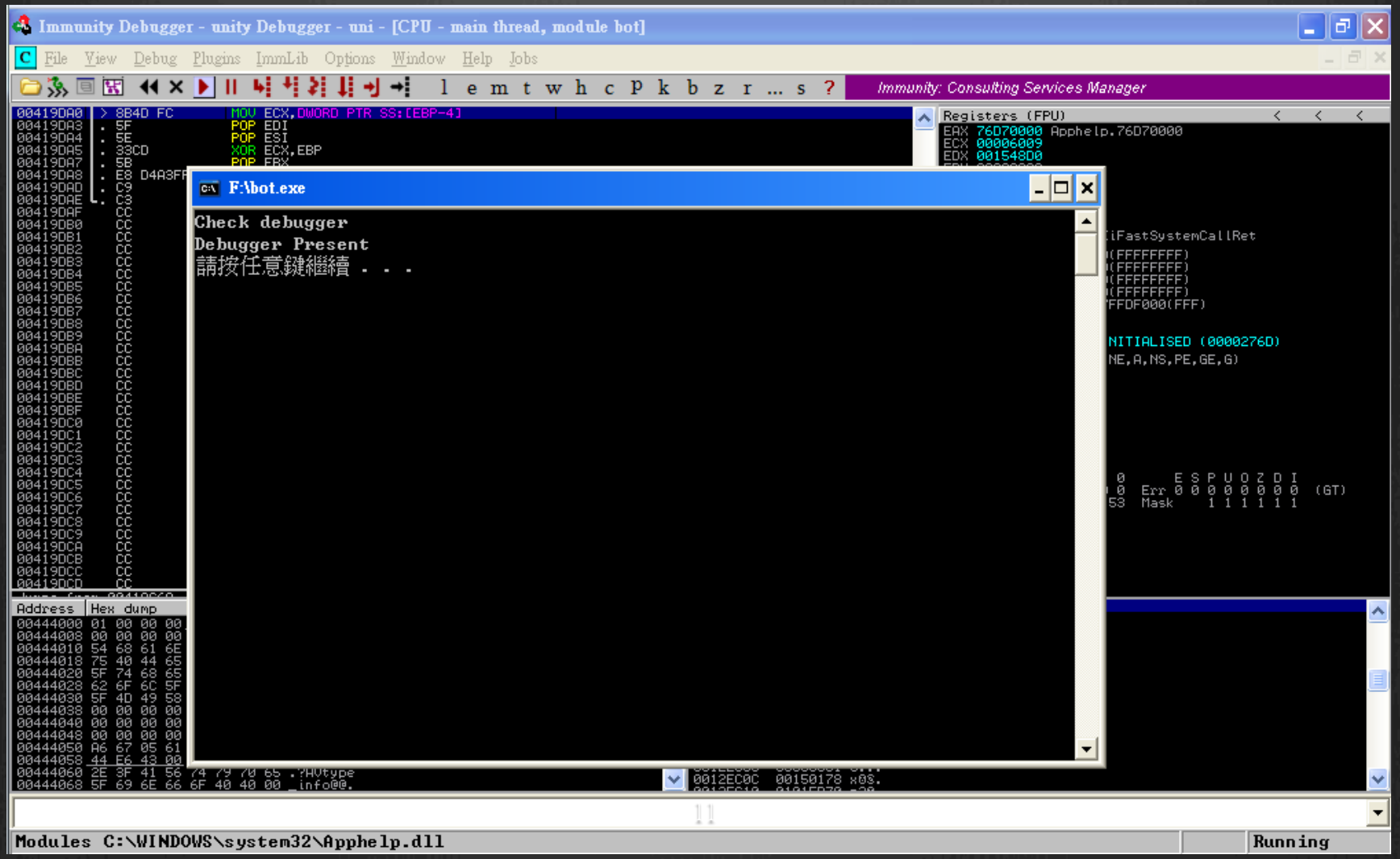
Anti-Debug Example

```
push offset exception_handler; set exception handler
push dword ptr fs:[0h]
mov dword ptr fs:[0h],esp
xor eax,eax;reset EAX invoke int3
int 3h
pop dword ptr fs:[0h];restore exception handler
add esp,4

test eax,eax; check the flag
je rt_choke
jmp rf_choke
```

```
exception_handler:
mov eax,dword ptr [esp+0xc];EAX =
 ContextRecord
mov dword ptr [eax+0xb0],0xffffffff;set flag
 (ContextRecord.EAX)
inc dword ptr [eax+0xb8];set ContextRecord.EIP
xor eax,eax
retn
```

Anti-Debug Result



The "Ultimate Anti-Debugging" Reference

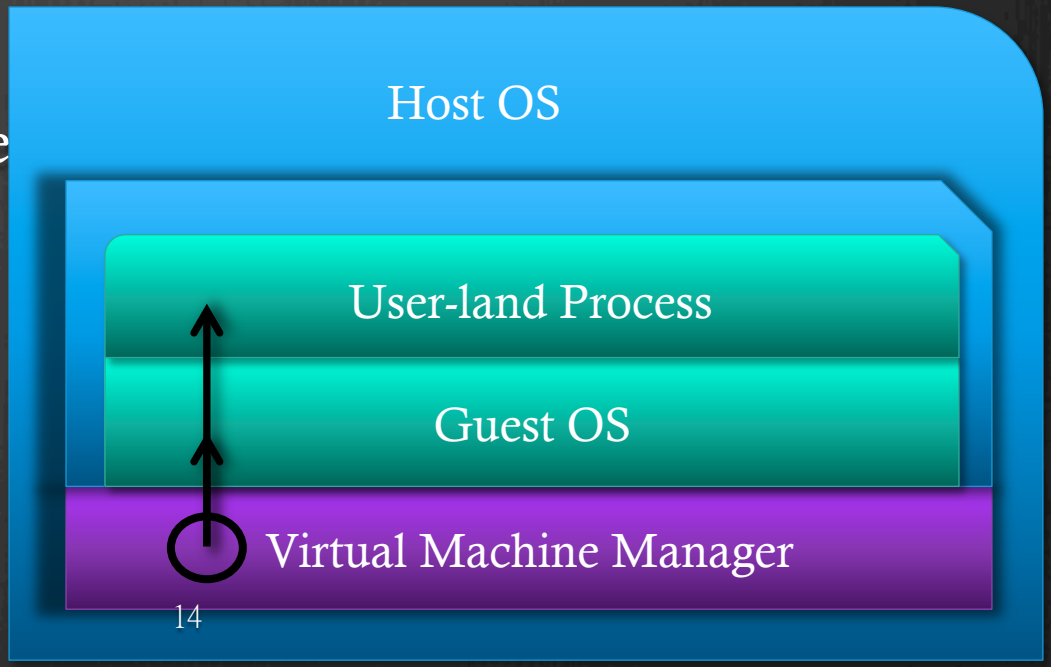
- http://pferrie.host22.com/papers/antidebug.pdf

How can we do?

- Can we move analysis tools outside the vm?

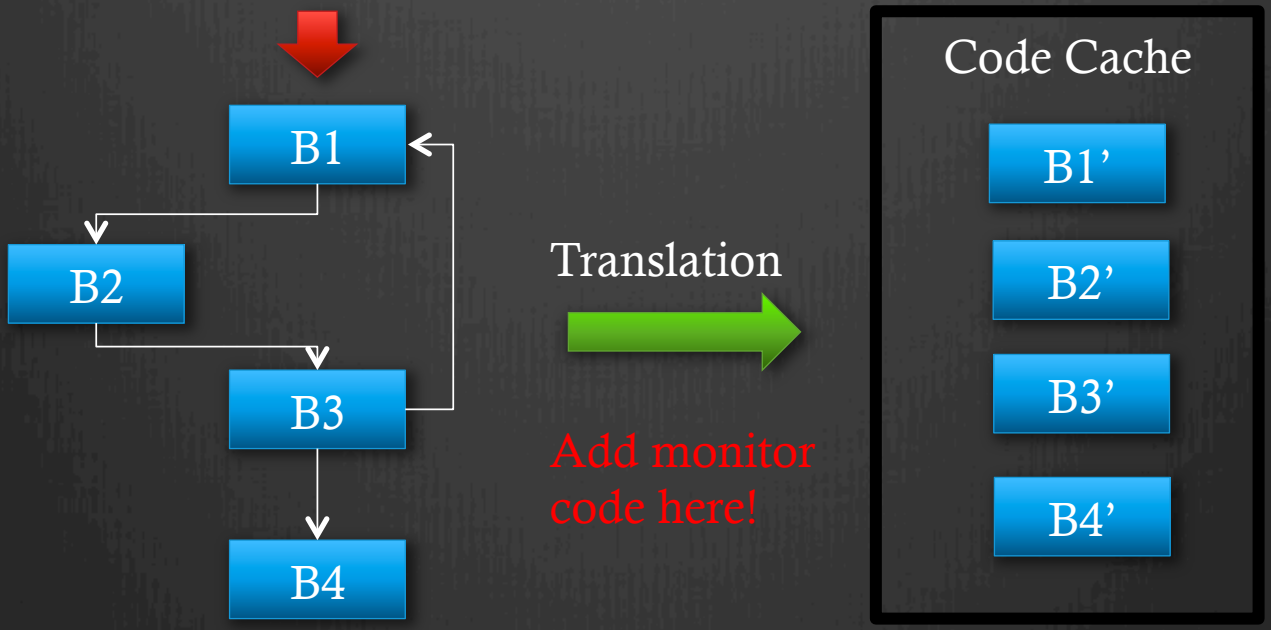
Out-of-Box Monitor

- Is it possible to monitor program behavior outside the VM
  - Out-of-Box Hooking
  - Virtual Machine Introspection
- How can we monitor the program's behavior outside the VM
- Virtual Machine Type
  - Emulation
  - Virtualization



Emulation-based VM

- Emulation-based VM
  - QEMU, Hydra, Bochs
  - Interpreter, Dynamic Translation



Monitor Based on Emulation

- Temu
- TTAnalyzer
  - Now, it become Anubis
- MBA
  - Develop by us !

Identify Process

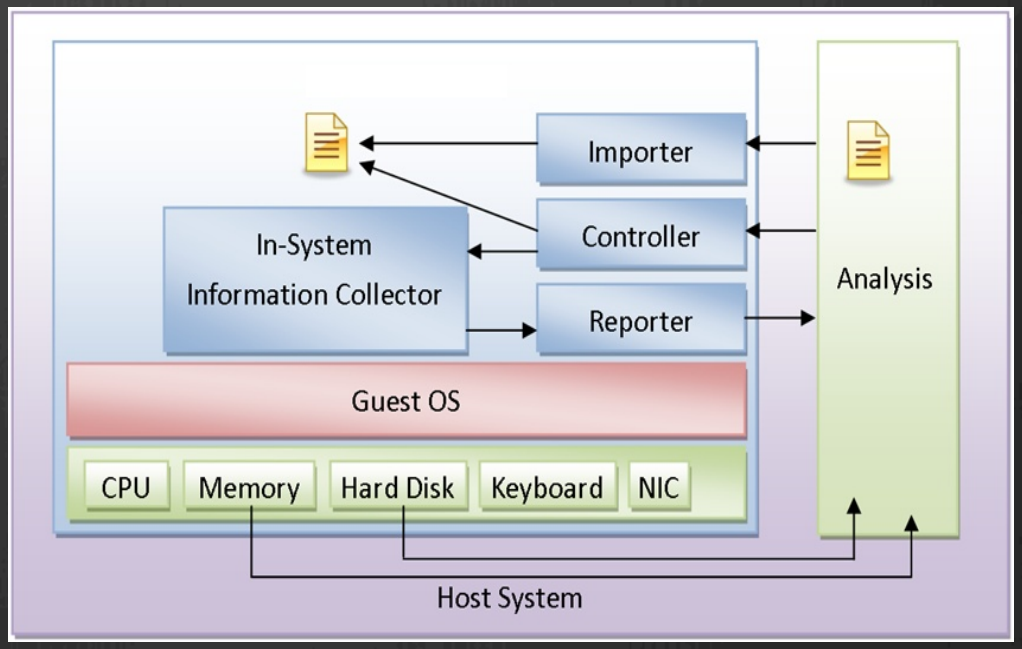
- The first step of Out-of-box monitor is to identify process we want to check

Monitor Execution Trace

- Then we would like to monitor execution of process
- Helper function
- Monitor Execution Trace
  - Add helper function when each instruction translate

Malware Behavior Analyzer

- MBA(Malware Behavior Analyzer)
  - MBA run sample in the qemu and extract it's behavior
  - Produce readable report for analysts
  - Monitor binary，前後比較，內外比對
- What MBA trace
  - File
  - Registry
  - Network
  - MBR
  - SSDT
  - …



Report of MBA(1)

- Analysis file : cad9d083ab6de63b9ddbb08fb0fc64ad
  - It's classify to TR/Inject.126976.5 by AntiVir

Report of MBA(1)

- Analysis file : cad9d083ab6de63b9ddbb08fb0fc64ad
- Modified Files

```
===== Files tainted =====
/Documents and Settings/dsns/NTUSER.DAT
/Documents and Settings/dsns/NTUSER.DAT.LOG*
/Documents and Settings/dsns/桌面/
cad9d083ab6de63b9ddbb08fb0fc64ad.exe
/WINDOWS/system32/config/software
/WINDOWS/system32/config/software.LOG
/WINDOWS/system32/inetsrv/inetsr.exe
```

Report of MBA(2)

- Analysis file : cad9d083ab6de63b9ddbb08fb0fc64ad
- Network Packets

```
===== Packet tainted =====
-> 168.95.1.1 , UDP 1026 -> 53 , (v 0x01 0x00 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00
0x05 gsmof 0x06 seed01 0x03 com 0x02 tw 0x00 0x00 0x01 0x00 0x01
-> 50.115.42.145 , TCP 1027 -> 443 , 0x02 0x04 0x05 0xb4 0x01 0x01 0x04 0x02
-> 50.115.42.145 , TCP 1027 -> 443 , 0x02 0x04 0x05 0xb4 0x01 0x01 0x04 0x02
-> 50.115.42.145 , TCP 1027 -> 443 , 0x02 0x04 0x05 0xb4 0x01 0x01 0x04 0x02
-> 50.115.42.145 , TCP 1027 -> 443 , 0x02 0x04 0x05 0xb4 0x01 0x01 0x04 0x02
-> 50.115.42.145 , TCP 1027 -> 443 , 0x02 0x04 0x05 0xb4 0x01 0x01 0x04 0x02
-> 50.115.42.145 , TCP 1027 -> 443 , 0x02 0x04 0x05 0xb4 0x01 0x01 0x04 0x02
-> 50.115.42.145 , TCP 1027 -> 443 , 0x02 0x04 0x05 0xb4 0x01 0x01 0x04 0x02
```

Report of MBA(3)

- Analysis file : cad9d083ab6de63b9ddbb08fb0fc64ad
- Modified Registries

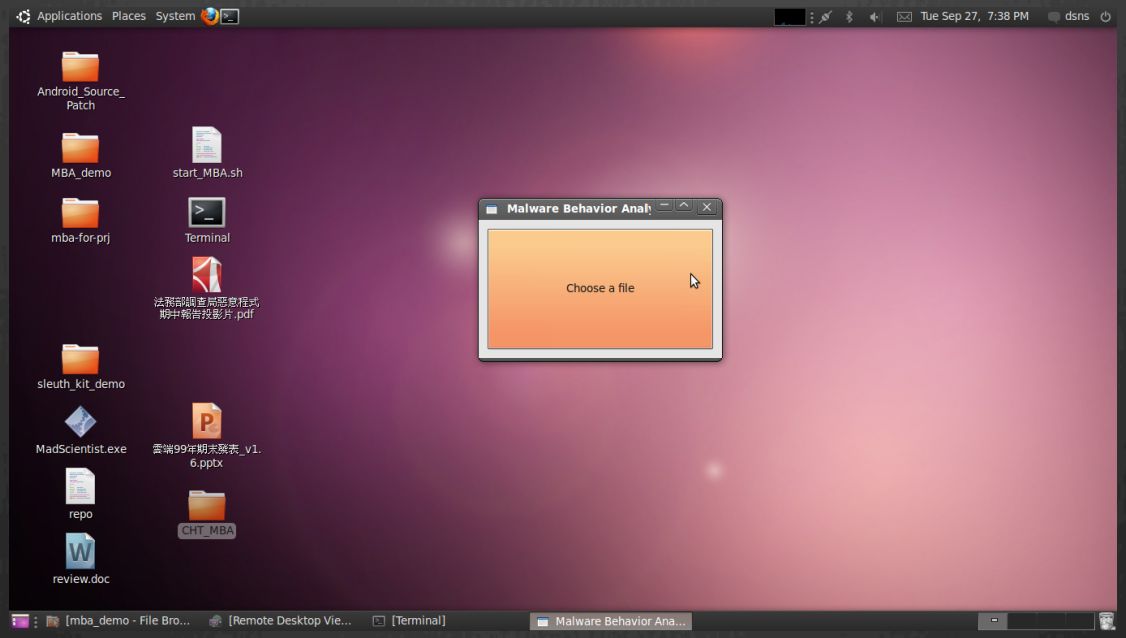
```
===== Registry tainted =====
/WINDOWS/system32/config/SOFTWARE/Microsoft/Active Setup/Installed Components/
{181E2749-8F28-E14F-ECEF-F89FC5739401} StubPath REG_SZ c:\windows\system32\inetsrv
\inetsr.exe
/WINDOWS/system32/config/SOFTWARE/Microsoft/Cryptography/RNG Seed REG_BINARY
/Documents and Settings/dsns/ntuser.dat/Software/Microsoft/Windows/ShellNoRoam/MUICache C:
\DOCUME~1\dsns\LOCALS~1\Temp\anyexe.bat REG_SZ anyexe
```

- Created Process

```
===== Process tainted =====
cad9d083ab6de63, 904
svchost.exe, 876
```

Demo

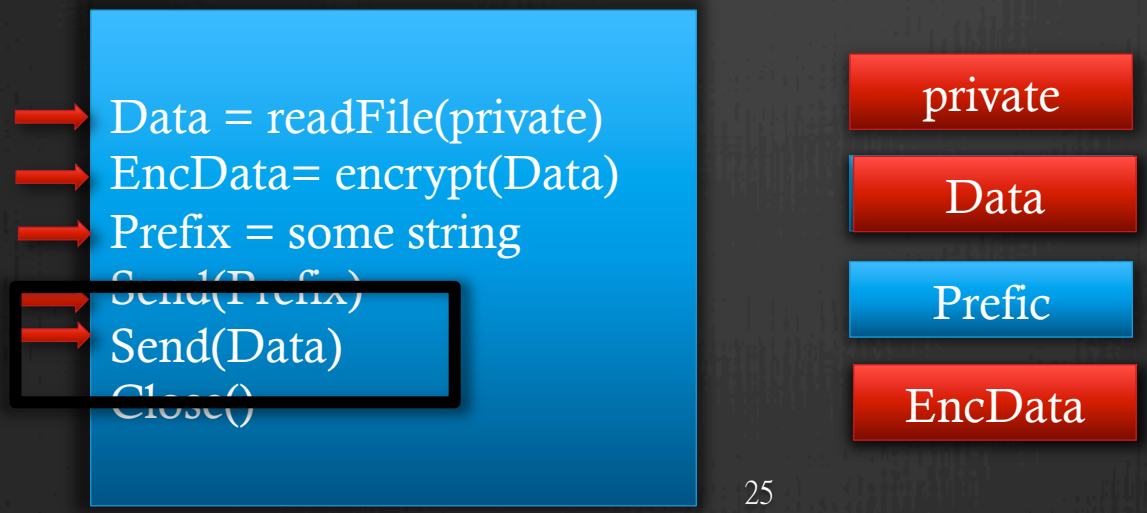
- As my experience, this demo will make my pc halt for a while, so we leave it to end of presentation.



Dynamic Taint Tracking

- Dynamic taint tracking is useful tool for binary analysis
  - Precise track influence data of certain event
  - Eliminate un-related event/data

- Concept of Infection



Taint Source

- What we want to track
  - File
  - Network
  - Executables
  - ….
- Private in previous example

Taint Propagate

- How to propagate taint tag
  - These rules describe how data flow corresponding to each behavior

Taint Sink

- Where we want to check taint status
  - Send() in our example
- Example
  - Taint Source : sensitiveFile
  - Taint Sink : sendPkt()

```
Content = readFile("sensitiveFile");
Encode = ""
For i in content :
      encode<= encode + i ^ ff
sendPkt(encode)
```

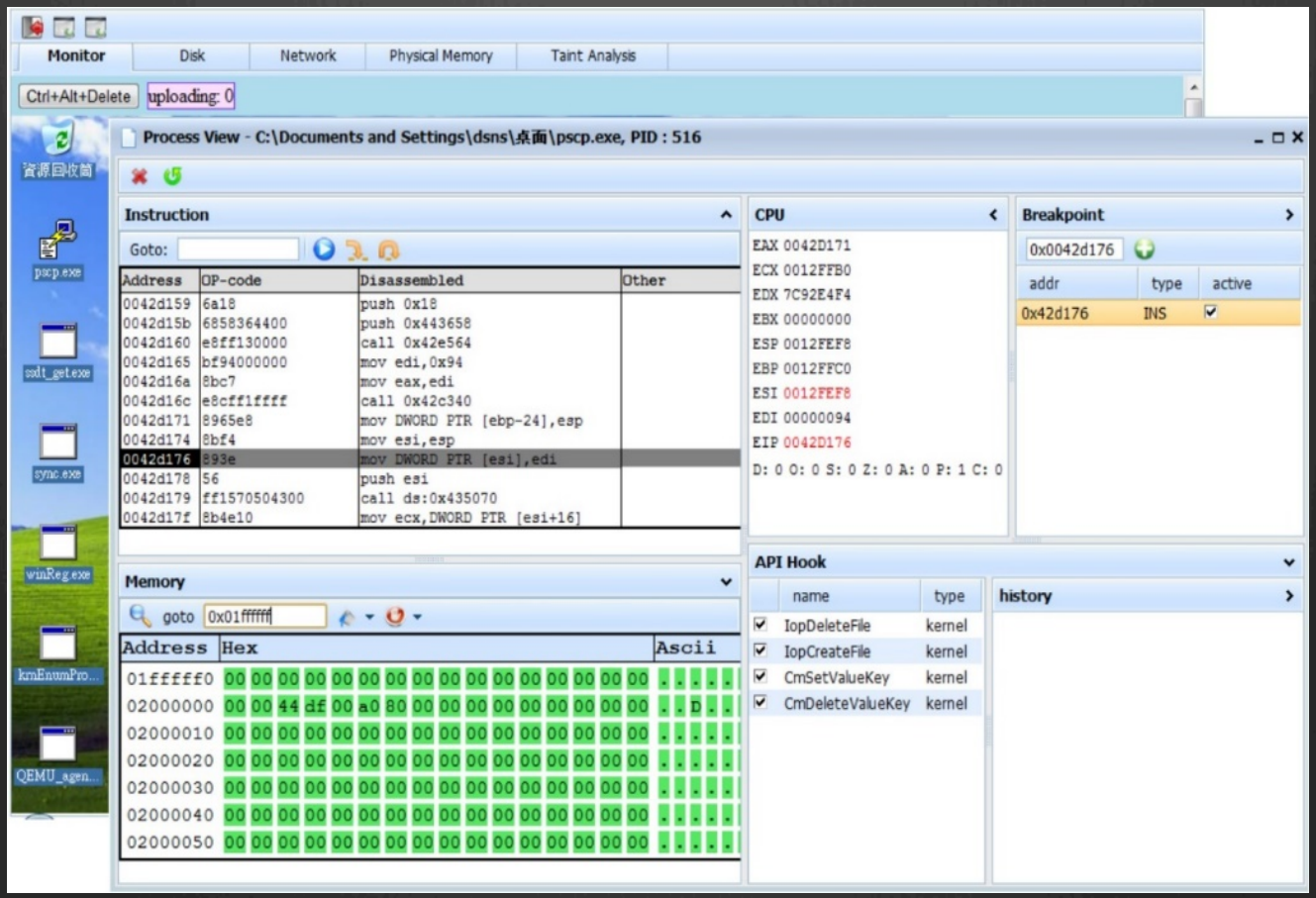
Other Application of Taint

- Detect software vulnerabilities and identify possible exploit
  - If EIP tainted while program running
  - Crax use Taint/Concolic Execution to produce exploit for software testing
    - There are the talk in HITCON PLG by SQLab student
- Detect sensitive data leak
- Detect key logger

Cloudebug - A User-interactive Malware Analysis Platform

- Deploy as the web service
  - Analysis malware without environment setting
- Transparent System
  - Out-of-box Monitor
  - Out-of-box Debugging
- Advanced Analysis Capability
  - Taint
- User Friendly
  - Javascript API

Demo


Monitor
Disk
Network
Physical Memory
Taint Analysis
Ctrl+Alt+Delete
uploading: 0
Process View - C:\Documents and Settings\dsns\桌面\pscp.exe, PID : 516
Instruction
Goto:
Address OP-code Disassembled Other
0042d159 6a18 push 0x18
0042d15b 6858364400 push 0x443658
0042d160 e8ff130000 call 0x42e564
0042d165 bf94000000 mov edi,0x94
0042d16a 8bc7 mov eax,edi
0042d16c e8cff1ffff call 0x42c340
0042d171 8965e8 mov DWORD PTR [ebp-24],esp
0042d174 8bf4 mov esi,esp
0042d176 893e mov DWORD PTR [esi],edi
0042d178 56 push esi
0042d179 ff1570504300 call ds:0x435070
0042d17f 8b4e10 mov ecx,DWORD PTR [esi+16]
CPU
EAX 0042D171
ECX 0012FFB0
EDX 7C92E4F4
EBX 00000000
ESP 0012FEF8
EBP 0012FFC0
ESI 0012FEF8
EDI 00000094
EIP 0042D176
D: 0 O: 0 S: 0 Z: 0 A: 0 P: 1 C: 0
Breakpoint
0x0042d176
addr type active
0x42d176 INS
Memory
goto 0x01ffffff
Address Hex Ascii
01fffff0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... .
02000000 00 00 44 df 00 a0 80 00 00 00 00 00 00 00 00 00 ..D. .
02000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... .
02000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... .
02000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... .
02000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... .
02000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..... .
API Hook
name type
IopDeleteFile kernel
IopCreateFile kernel
CmSetValueKey kernel
CmDeleteValueKey kernel
history
資源回收筒
pscp.exe
sdt_get.exe
sync.exe
winReg.exe
kmEnumPro...
QEMU_agen...

Detect Virtual Machine Environment

- Types and samples of anti-vm technique
  - Hardware Characteristic Checks
  - Timing Checks
  - Emulation Bug Checks

Environment Characteristic Checks

- Hardware specification used to detect virtualization platform
  - Files
  - Registry
  - Process
  - Device Name

```
xor eax, eax
cpuid
cmp ecx, 444d4163h
jne exit
mov eax, 80000000h
cpuid
cmp eax, 2
jb exit
mov eax, 80000002h
cpuid
cmp eax, 554d4551h
je $ ;detected
```

M-Check

Divergence Point

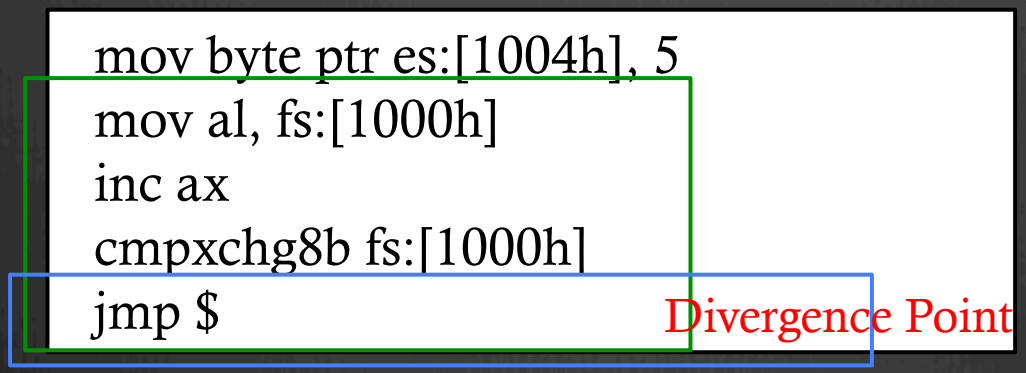
Timing Checks

- Timing difference between physical machine and virtual machine can be used to detect VM

```
…
0x4012ce:    rdtsc
0x4012d0:    mov [0x404060], %eax
0x4012d5:    rdtsc
0x4012d7:    mov [0x404070], %eax
0x4012dc:    mov %edx, [0x404060]
0x4012e2:    mov %eax, [0x404070]
0x4012e7:    sub %eax, %edx
0x4012e9:    cmp %eax, 0xff
0x4012ee:    jle 0x4012fe
…
```

Emulation Bug Checks

- Instruction emulated by software may be inconsistence to physical machine



What is Transparent VM

- Guideline from Ether
  - Higher Privilege
  - No Non-privileged Side Effects
    - Any privilege instruction are back to vmm and emulated by software
  - Identical Basic Instruction Execution Semantics
    - 16 rep prefix instruction will make qemu crash
  - Transparent Exception Handling
  - Identical Measurement of Time

Is it possible to build Transparent VM

- Construct transparent analysis VM platform
  - It is extremely hard to implement a transparent system
  - Difficult to verify the completeness
  - Large amount of analysis tool is not based on transparent platform
- How can we do if we don't have such transparent VM

Behavior Comparison to Detect Virtual Machine Awareness

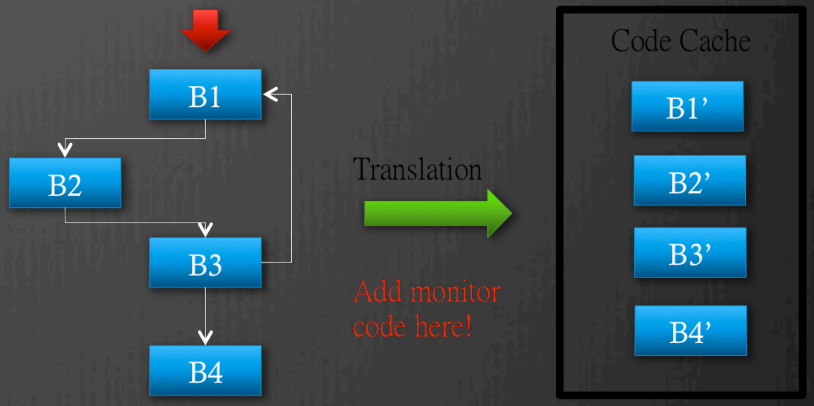
- Hybrid Emulation & Virtualization to detect Anti-VM malware
- Anti-vm technique is hard to detect all the vm platform in one instruction
  - The code coverage diverge in different VM system
- How to hunt anti-vm malware
  - Execute program in multiple VM system(or physical one if possible)
  - Construct code coverage
  - Compare if there are something different

Virtualization-based VM

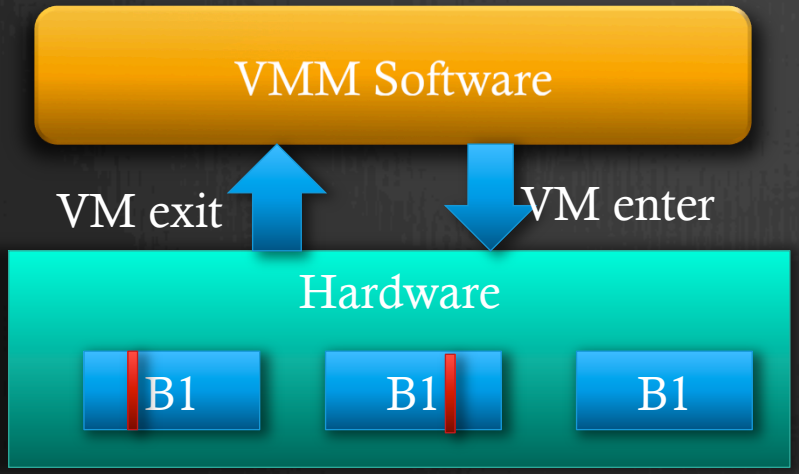
- Virtualization-based VM
  - KVM, XEN, ….
  - Use hardware-assistant virtualization to improve the transparent and performance
- Programming Logic
  - Compare to emulation system which like sequential logic
  - Hardware-Assistant Virtualization more like event-driven model

Virtualization-based VM

- Remind how emulation works



- How Virtualization Work



Monitoring Scope

VM

- Privilege instruction

- Privilege instruction

Emulator

- Inst1
- Inst2
- Inst3
- Privilege instruction
- Inst 4
- Inst 5
- Privilege instruction

Monitor Based on Virtualization

- Ether(XEN)
- XENAccess(XEN)
- VMITools(XEN, KVM)
- Nitro(KVM)
- ….

Monitor System Call

- System call monitor can be implement with similar concept
  - Make the exception every time system call happened
- How system work
  - When system call happened by SYSENTER instruction
  - OS jump to privilege location defined by SYSENTER_EIP_MSR to handle system call
- Monitor system call
  - Replace value in SYSENTER_EIP_MSR to some invalid address
  - Implement exception handler to profile behavior and put correct value back

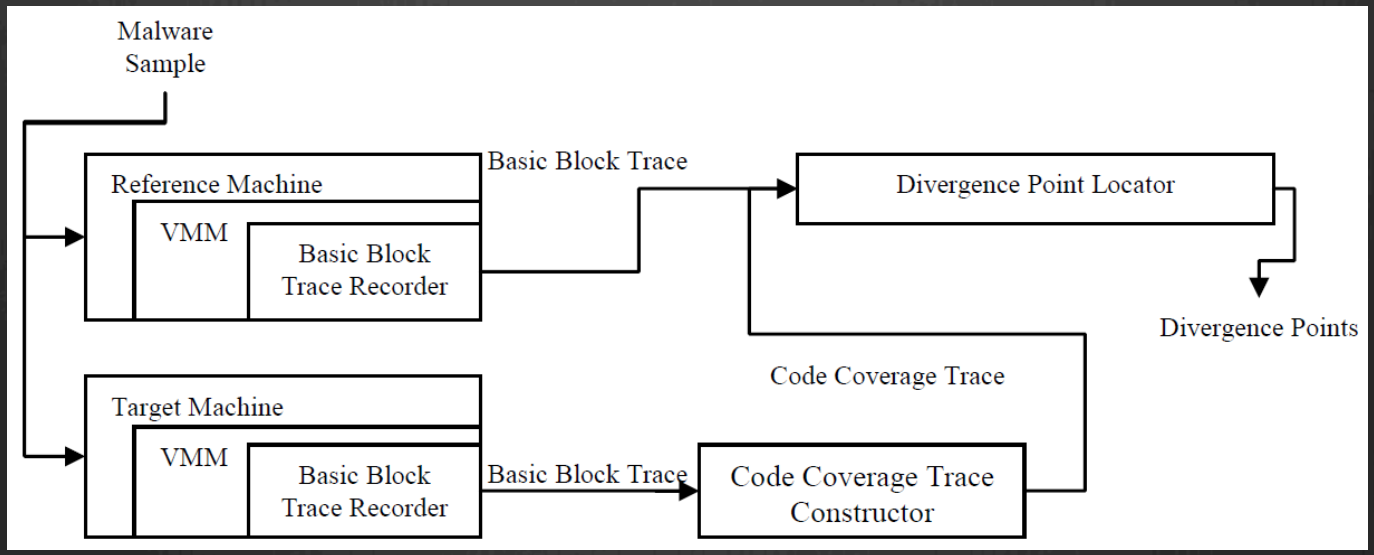
Monitor Instruction Trace

- While put the program to execute in virtualization system, our software cannot direct monitor instruction trace
  - The instruction is directly run by CPU
  - Not go through VMM, which means not manipulate by software

- Enforce debug exception triggered every instructions
  - Setting Trap flag to enable debug trap in every instruction
  - VMExit happened, and VMM gain the control
  - Therefore we can use software to handle/profile the behavior

Divergence Point Locator

- In our system, we use two VM system
  - Qemu
  - XEN
  - kvm



Partial Assembly Code of *rdtsc* Timing Check

- ~~Assembly of rdtsc sample~~

```
…
0x4012ce: rdtsc
0x4012d0:      mov [0x404060], %eax
0x4012d5:      rdtsc
0x4012d7:      mov [0x404070], %eax
0x4012dc:      mov %edx, [0x404060]
0x4012e2:      mov %eax, [0x404070]
0x4012e7:      sub %eax, %edx
0x4012e9:      cmp %eax, 0xff
0x4012ee:      jle 0x4012fe
…
```

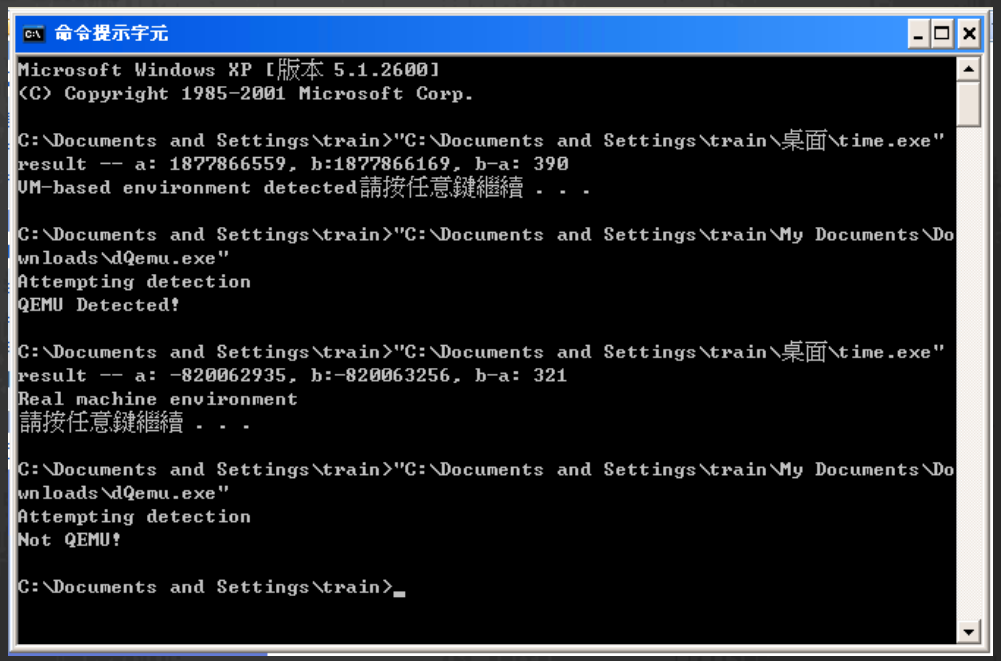
Result of *rdtsc* Timing Check

- Code block coverage of rdtsc sample

Executed Basic Blocks on KVM	Executed Basic Blocks on QEMU
…	…
0x401260-0x40126a	0x401260-0x40126a
0x401446-0x401449	0x401446-0x401449
0x4012ba-0x4012ee	0x4012ba-0x4012f7
0x4012fe-0x401305(not executed on QEMU)	0x401850-0x401850
0x401850-0x401850	0x40130a-0x401311
0x40130a-0x401311	…
…	

Bypass Anti-VM in the Fly

- Once we know the location of Anti-VM, we can make the signature
  - For runtime patch the executed process
  - Make Anti-VM fails

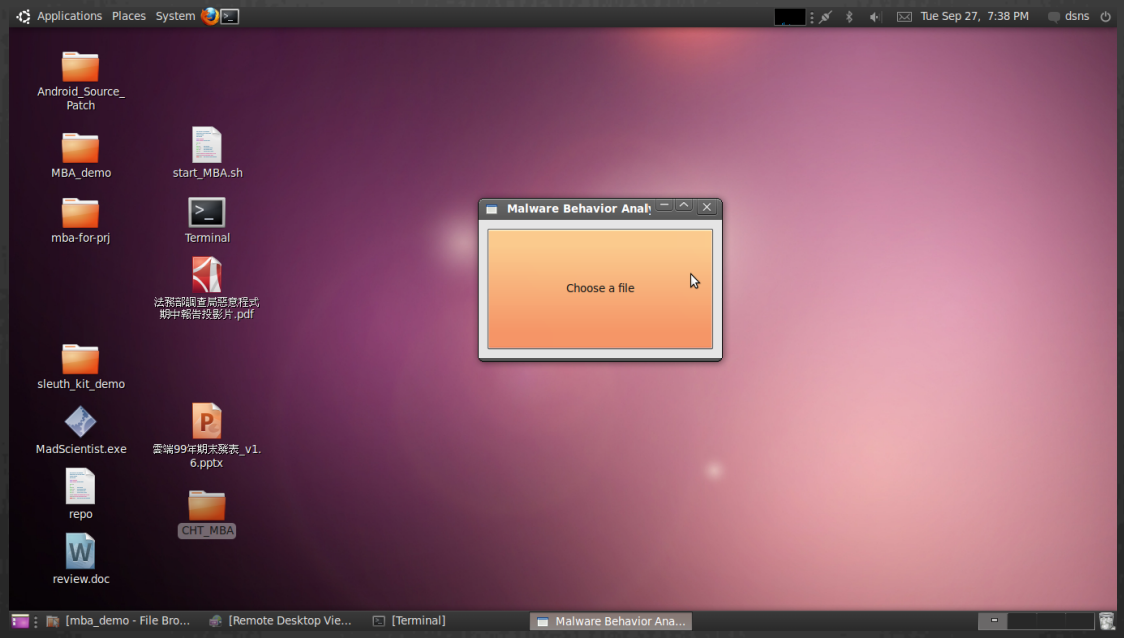


Summary

- Out-of-box monitor to defense anti-debug
- Malware behavior analyzer
- Taint tracking
- Cloudebug
- Anti-vm
- Trace comparison to find out anti-vm

Demo

- Remember that we need to demo ☺



Q & A