TENTACLE:
Environment-Sensitive Malware Palpation

Yosuke Chubachi, Kenji Aiko

FFRI, Inc.
http://www.ffri.jp

PacSec Tokyo November 2014
About us

Yosuke Chubachi is a security engineer at FFRI, Inc. since this spring. He studied at the graduate school of information system engineering, University of Tsukuba. He is a Security Camp lecturer and a member of executive committee of SECCON since 2012.

Kenji Aiko is a programmer at FFRI, Inc., and is a one of the developers of "FFR yarai" which is a targeted attack protection software. He is a Security Camp lecturer and a member of executive committee of SECCON since 2012.
Contents

• Background/Motivation
• Taxonomy of anti-sandbox techniques
• Tentacle Design
• Implementation
• Experiments
• Conclusion
Background

- Malware explosion
  - 120,000,000/recent year

- Antivirus is dead…?

AV Test: Statistics –New Malware- (Nov. 05 2014 viewed)
We need dynamic and automated malware analysis

• “Scalability” is most important factor in information explosion era
  – Cloud
  – Bigdata
  – IoT

• Malware analysis also needs “scalable” methodology
  – Automation is justice
Obi wan said, “Use the sandbox, Luke”

- Security engineer and researcher use sandbox environment for malware analyzing

- Automated dynamic analysis technology also based on VM/application sandbox
  - Anubis(online sandbox)
  - Cuckoo Sandbox(VirtualBox base)
  - Some commercial UTM Appliance
  - Some commercial endpoint security solution
Sophisticated malware strike back

• Sophisticated malware arms many anti-analyze techniques
  • Naturally using targeted attacks, cyber espionage, banking malware

• We called those malware “evasive malware”
Related work

• BareCloud [Dhilung K et al., USENIX SEC’14]
  – “5,835 evasive malware out of 110,005 recent samples”

• Prevalent Characteristics in Modern Malware [Gabriel et al., BH USA ‘14]
  – “80% malware detect vmware using backdoor port”

What do you think?
Case study: Citadel

- Some citadel detects the execution environment and do not engage in malicious behavior when the current host differs from the infected host
  - To avoid dynamic malware detection (like sandbox analysis)

- Showing 2 examples
  - Host-fingerprinting
  - VM/Sandbox detection
Host-fingerprinting

- Embedding infected host’s unique value into execution binary

Different signature pattern
Host-fingerprinting (cont’d)

- Getting GUID on system drive using the GetVolumeNameForVolumeMountPoint()
- Comparing running host’s GUID value and embedded infected host’s value
- Process executes malicious code if GUID values are consistent

**Infected host’s GUID (packed)**

**Unpacked GUID Format:**

\{XXXXXXXX-XXXX-XXXX-XXXXXXXX\}

**Environment-sensitive Citadel**

**Unpacker**

**Malicious code with host unique value**
VM/Sandbox detection

- Checking process’s product name
  - like "*vmware*", "*virtualbox"

- Scanning specific files and devices
  - C:\popupkiller.exe
  - C:\stimulator.exe
  - C:\TOOLS\execute.exe
  - \.\NPF_NdisWanIp
  - \.\HGFS
  - \.\vmci
  - \.\VBoxGuest
Citadel behavior of host/environment inconsistency

- For example:
  - Process termination
  - Running fake(or harmless) code
Citadel runtime activities

- Initialization
  - unpack
- Scouting
  - Environment-awareness
  - Host fingerprinting
- Malicious Behavior
  - Malicious code execution
    - Code injection
    - Unauthorized Access
    - Connecting C&C server
Aim of evasive malware

• Signature-based AV detection avoiding

• Anti-analyzing
  – Anti-security engineer
  – Sandbox evasion
Motivation

• An automatic investigation into a condition used by sandbox evasion
  – Faking bare-metal environment on sandboxes

Transparent Sandbox
Using Investigated Conditions
Know Your Enemy

TAXONOMY OF ANTI-SANDBOX
Taxonomy of anti-sandbox techniques

- Anti-sandbox maneuver
  - Environment awareness
    - Using result of sandbox detection
    - (Stalling code)
    - (User interaction checks)

- Sandbox (debug/sandbox/vm) detection
  - Artifact fingerprinting
  - Execution environment fingerprinting
  - Execution timing detection
Environment awareness

• Checking host environments
• If malware runs **decoy routine** then it detects analyzer’s sign
  – Malicious behavior never executed

![Diagram of Environment awareness process]
Sandbox (debug/sandbox/vm) detection

Environment aware Malware Artifact Fingerprinting

VM related Artifacts
Sandbox specific Artifacts

Execution Timing Detection
Execution Environment Fingerprinting

VMM?
Hardware
Artifact fingerprinting

- Sandbox/VM related processes
  - Like vmware, virtualbox etc.
- Sandbox/VM environment specific files
- Sandbox/VM environment specific registry keys
- Sandbox/VM environment specific devices and its attributes
  - ex). QEMU HDD vendor name
- Sandbox/VM Specific I/O port
  - VMWare backdoor port is most famous artifact in malware
Execution environment fingerprinting

• Using virtual machine implementation specific platform value and reaction
  – CPUID instruction result
  – Redpill
    • Using LDT/GDT and IDT incongruousness
  – Interesting research here: Cardinal Pill Testing
## Execution timing detection

- For example: using clock count differential
  - Traditional anti-debug technique

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>400022A2</td>
<td>00</td>
<td>PUSHAD</td>
</tr>
<tr>
<td>400022A3</td>
<td>0F31</td>
<td>RDTSC</td>
</tr>
<tr>
<td>400022A5</td>
<td>31C9</td>
<td>XOR ECX,ECX</td>
</tr>
<tr>
<td>400022A7</td>
<td>01C1</td>
<td>ADD ECX,EAX</td>
</tr>
<tr>
<td>400022A9</td>
<td>0F31</td>
<td>RDTSC</td>
</tr>
<tr>
<td>400022AB</td>
<td>29C8</td>
<td>SUB EAX,ECX</td>
</tr>
<tr>
<td>400022AD</td>
<td>3D FF0F0000</td>
<td>CMP EAX,0FF</td>
</tr>
<tr>
<td>400022B2</td>
<td>61</td>
<td>POPAD</td>
</tr>
<tr>
<td>400022B3</td>
<td>0F83 11010000</td>
<td>JNB 400023CA</td>
</tr>
</tbody>
</table>

Comparing two TSC differentials
Targets of today’s presentation

• Anti-sandbox maneuver
  ✓ Environment awareness
    • Using result of sandbox detection
  □ Stalling code
  □ User interaction checks

• Sandbox (debug/sandbox/vm) detection
  ✓ Artifact fingerprinting
  ✓ Execution environment fingerprinting
  □ Execution timing detection
Automatically disarmament system for armed malware with anti-sandboxing

**TENTACLE: DESIGN**
Concept: malware palpation

1. “Thousand form” sandbox runs malware again and again
   – Changing “virtual” artifacts exposure each execution for execution branch detection

2. Retroactive condition analysis
   – Specifying “branch condition” on unnatural process termination
Malware palpation

- Sandbox runs same malware again and again
- Sandbox fakes different sandbox-related artifacts each malware execution
  - Detecting execution difference using code execution integrity (CEI)
Code Execution Integrity (CEI)

- CEI shows uniqueness of instruction execution history
  - Inspired by TPM trust chaining
- “measurement” per instruction

Digest[i] = SHA1( fetched CPU instruction + Digest[i-1] )

```
mov $0x616b6157, %eax 0xb857616b61  
push %ebx               0x53 
push %eax               0x50 
mov $4, %edx            0xba04000000  
mov $1, %ebx            0xbb01000000  
```

\[ d[0] = SHA1(0xb857616b61) \]
\[ d[1] = SHA1(d[0] + 0x53) \]
\[ d[2] = SHA1(d[1] + 0x50) \]
\[ d[3] = SHA1(d[2] +0xba04000000) \]

...
Execution branch detection

- Using execution step count and code execution integrity (CEI) value

```
in  eax, dx;
cmp  ebx,
jne   notVmware
jmp   isVmware

notVmware: mov rc, 0
           jmp done

isVmware:  mov rc, eax
           jmp done
```
Retroactive condition analysis

• Sandbox retroactive from termination to terminated reason API and arguments when suspicious termination
  - Only a few steps executions
  - To terminate before network activities

```assembly
sub esp, 1024
mov ebx, esp
push 400h
push ebx
push 0h
call GetModuleFileNameA
lea eax, MyPath
push eax
push ebx
call lstrcmpA
test eax, eax
push 0h
lea eax, MsgCaption
push eax
jz _ok
lea eax, NGMsgText
push eax
push 0h
call MessageBoxA
invoke ExitProcess, NULL
_ok:
lea eax, OKMsgText
```
know thyself

TENTACLE: IMPLEMENTATION
Tentacle implementation overview

- Tentacle based on IA-32 CPU emulator
IA-32 CPU emulator-based sandbox

- We have already CPU Emulator-based sandbox for win32 execution (in-house use)
  - Like IDA Bochs PE operation mode
IA-32 CPU Emulator: Virtual contexts

CPU Emulator Process

- Virtual CPU Contexts
- Virtual Process Contexts
- Virtual Module (DLL) Contexts
- Virtual Resources (handle) Contexts
- Virtual Heap Contexts
  - Target Process's VA Space contexts

Target Process

- DLLs
- Data
- Code
- Headers
IA-32 CPU Emulator: API emulation

CPU Emulator Process

- Virtual kernel32 Functions
- Virtual user32.dll Functions
- Virtual advapi32.dll Functions
- Virtual msvcrtd.dll Functions
- API Emulation Components
- Virtualized Resource Manager

Target Process

- kernel32.dll
- user32.dll
- advapi32.dll
- msvcrtd.dll
- ...
Limitations

• The original CPU emulator supports a limited API

• The original CPU emulator supports a limited CPU instruction
Tentacle implementation details

• Nyarlathotep “Thousand Form” Sandbox
  – Camouflaging sandbox/vm related artifact existence
  – Execution branch detection using CEI

• Retroactive condition analysis

• *Necronomicon*
  – Execution logging framework
Necronomicon

• Necronomicon logs instruction per execution
  – Tracing specific API call and its arguments for
    Retroactive condition analysis
    • _lstrcmp, GetModuleFileName,
      GetVolumeNameForVolumeMountPoint…

• Code execution integrity calculation per execution
  – For execution branch detection
Nyarlathotep “Thousand Form” Sandbox

• For example, VMWare related artifacts
  – backdoor port
  – vm-related files (2 files)
  – registry entries (1 entry)

• Sandbox fake “in” instruction result for backdoor port check

• Files and registry entry virtualization used by sandbox’s own virtual resource manager
Experiments

- Disarmament #01: Artifact fingerprinting sample
- Disarmament #02: Comparing Disk GUID value sample
- Disarmament #03: Comparing executable file path sample
Disarmament #01: Artifact fingerprinting sample

```c
int _tmain(int argc, _TCHAR* argv[]) {
    int count = 0;
    delay();
    if ( sbdetect_vmware_backdoor_port() > 0){
        printf("VMWare backdoor port detected. I am on the virtual.");
    }
    if ( sbdetect_vmware_sysf01() == 0 ){
        printf("vmmouse.sys detected. I am on the virtual.\n");
        exit(1);
    }
    if ( sbdetect_vmware_sysf02() == 0 ){
        printf("vmmouse.sys detected. I am on the virtual.\n");
        exit(1);
    }
    if ( sbdetect_vmware_reg01() == 0 ){  
        printf("RegKey: \"SOFTWARE\VMware, Inc.\VMware Tools\"
        detected. I am on the virtual.\n");
        exit(1);
    }
    //Malicious behavior
    printf("malicious behavior\n");
    exit(1);
    return 0;
}
```
sbdetect_vmware_backdoor_port()

int sbdetect_vmware_backdoor_port(void)
{
    int rc = 0;
__try
{
    __asm
    {
        mov   eax, 'VMXh'
        mov   ebx, 0;
        mov   ecx, 0xA
        mov   edx, 'VX'  // port
        in    eax, dx;  // read port
        cmp   ebx, 'VMXh' // Vmware echo ‘VMXh’
        jne   notVmware
        jmp   isVmware
    notVmware:
        mov   rc, 0
        jmp   done
    isVmware:
        mov   rc, eax
        done:
    }
}
__except(EXCEPTION_EXECUTE_HANDLER)
{
    rc = 0;
}
}

return rc;
sbdetect_vmware_sysf01()

//
// Detect proven: Windows 7 32bit
// Not detected: Windows 7 64bit
//
int sbdetect_vmware_sysf01() {
    DWORD ret;
    TCHAR target[255] = "%WINDIR%\system32\drivers\vmmouse.sys";
    ret = GetFileAttributes(target);
    if( ret != INVALID_FILE_ATTRIBUTES ){
        return 0;
    }
    else{
        return 1;
    }
}
sb detect_vmware_sysf02()

//
// Detect proven: Windows 7 32bit
// Not detected: Windows 7 64bit
//
int sb detect_vmware_sysf02() {
    DWORD ret;
    TCHAR target[255] = "%WINDIR%\system32\drivers\vmhgfs.sys";
    ret = GetFileAttributes(target);
    if( ret != INVALID_FILE_ATTRIBUTES ){
        return 0;
    }
    else{
        return 1;
    }
}
Disarmament #01

```
[temtacle] TEST targets

[temtacle] Target file: ../Release/SampleAntiSandbox01.exe
[temtacle] Running vanilla environment

[temtacle] palpatuion#00 All artifact exposure
[temtacle] Anti-sandbox detected.

[temtacle] palpatuion#01
[temtacle] Detect reason: VMMare backdoor port

[temtacle] palpatuion#02
[temtacle] Detect reason: WINDIR\system32\drivers\vmmouse.sys exist.
nn_pages = 213

[temtacle] palpatuion#03
[temtacle] Detect reason: WINDIR\system32\drivers\vmhgf.sys exist.
```

継行するには何かキーを押してください．．．
Disarmament #02: Comparing Disk GUID value sample

.start:
    sub esp, 1024
    mov ebx, esp
    sub esp, 4
    mov eax, esp
    push eax
    push ebx
    lea eax, DriveC
    push eax
    call GetVolumeNameForVolumeMountPointA
    lea eax, VolumeC
    push eax
    push ebx
    call lstrcmpA
    test eax, eax
    push 0h
    lea eax, MsgCaption
    push eax
    jz _ok
    lea eax, NGMsgText
    push eax
    push 0h
    call MessageBoxA
    invoke ExitProcess, NULL

_ok:
    lea eax, OKMsgText
    push eax
    push 0h
    call MessageBoxA
    invoke ExitProcess, NULL

.end start
Disarmament #02

[tentacle] TEST targets

[tentacle] Target file: ../Release/SampleAntiSandbox03.exe
[tentacle] Running vanilla environment

[tentacle] palpatuion#00 All artifact exposure
[tentacle] Retroactive condition analysis
FOUND: 00401028 TEST EAX, EAX
FOUND: 0040101b
API: GetVolumeNameForVolumeMountPointA
ARGS: ¥¥??¥Volume[b753a495-0bc0-11e4-bf05-806e6f6e6963]¥
API: lstrcmp
ARGS: ¥¥??¥Volume[b753a495-0bc0-11e4-bf05-806e6f6e6963]¥
API: lstrcmp
ARGS: ¥¥??¥Volume[8e7e8884-600d-11e4-ae07-806e6f6e6963]¥
[tentacle] Sandbox evasion maneuver detected.
続行するには何かキーを押してください...
Disarmament #03 Comparing executable file path sample

```asm
.386
.model flat, stdcall
option casemap: none
include \masm32\include\windows.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\kernel32.lib
include \masm32\include\user32.inc
includelib \masm32\lib\user32.lib
.data
    MyPath db "C:\x\sample2.exe", 0
    MsgCaption db "MESSAGE", 0
    OKMsgText db "Normal Message", 0
    NGMsgText db "Detect Message", 0
    .code
    start
        sub esp, 1024
        mov ebx, esp
        push 400h
        push ebx
        push 0h
        call GetModuleFileNameA
        lea eax, MyPath
        push eax
        push ebx
        call lstrcmpA
        test eax, eax
        push 0h
        lea eax, MsgCaption
        push eax
        jz _ok
        lea eax, NGMsgText
        push eax
        push 0h
        call MessageBoxA
        invoke ExitProcess, NULL
        _ok:
        lea eax, OKMsgText
        push eax
        push 0h
        call MessageBoxA
        invoke ExitProcess, NULL
    end start
```
Disarmament #03

TENTACLE

[temtacle] TEST targets

-------------------------------
[temtacle] Target file: ../Release/SampleAntiSandbox04.exe
[temtacle] Running vanilla environment

[temtacle] palpatuion#00 All artifact exposure
[temtacle] Retroactive condition analysis
FOUND: 00401022 TEST EAX, EAX
FOUND: 00401015
API: GetProcAddress
ARGS: c:\Users\chubachi-devel\Documents\tentacle\Release\SampleAntiSandbox04.exe
exe
API: lstrcmp
ARGS: c:\Users\chubachi-devel\Documents\tentacle\Release\SampleAntiSanC:\yx\samnple2.exe
API: lstrcmp
ARGS: C:\x\ysample2.exe
[temtacle] Sandbox evasion maneuver detected.

続行するには何かキーを押してください...
Future work

• Improving anti-sandbox detection
  – Stalling code detection and evasion

• Improving sandbox quality
Conclusions

• This is proof of concept of automatically disarmament system for armed malware with anti-sandboxing with CPU emulator-based sandbox

• We introduced anti-sandbox taxonomy

"Know thy self, know thy enemy. A thousand battles, a thousand victories." - Sun Tzu
Bibliography

- Analyzing Environment-Aware Malware, Lastline, 2014.05.25(viewed)
- Aurélien Wailly. Malware vs Virtualization The endless cat and mouse play, 2014.05.25(viewed)
- IDA Boch PE operation mode
  https://www.hex-rays.com/products/ida/support/idadoc/1332.shtml
Thank you!