Freeze Drying for Capturing Environment-Sensitive Malware Alive

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

Yosuke Chubachi
WHO AM I?

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. His research interests are in operating system and virtual machine monitoring. Particular interests include access control and intrusion prevention systems.

He is a Security Camp lecturer (national information security human resource development program ) since 2011 and a member of executive committee of SECCON (SECurity CONtest, the largest CTF organizer in Japan) since 2012.

忠鉢 洋輔

CHUBACHI Yosuke
Contents

• Background
• Idea
  – Malware migration system for capturing malware alive

• Challenges
  – Process migration
  – Anti-anti-sandbox arming

• Implementation
  – Overview
  – IA32 CPU Emulator
  – Process migration using process-level sandbox
  – API Proxies for faking an artifact

• Conclusions
Background

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

• First, we need protection
• Second, we are curious about true intention
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[1]
  – To avoid behavior-based malware detection (like sandbox analysis)

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

- Embedding infected host’s unique value into execution binary
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)**

Format: `{XXXXXXXX-XXXX-XXXX-XXXXXXXX}`

**Unpacked GUID**

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
• I assume that scouting code carry out before main malicious routine

**Diagram:**

- Initialization (unpack)
- Anti-analyzing
- Malicious routine

**Text:**

- Not really matter
- A serious matter
Idea

- Security analyst or incident handler concentrate malicious activity observation if he migrate malware process from infected host to analyzing environment (or honeypot) when anti-analyzing behavior

```
Malware

End-user's Host

migration

Analyzing Environment
```
Use Case I: Malware live capturing

- End-users execute suspicious executable files anyway
- Capturing system will suspend program if to detect anti-analyzing behavior
- Malware analysts may observe to concentrate malicious activities
Use Case II: Honeypot

- Faking an artifact of the target host
  - To deceive cyber espionage malware
Challenges I

PROCESS MIGRATION
Challenges

1. Process migration is very difficult (well-known)
   - Needs to migrate execution contexts, memory contexts, persistent contexts and related kernel objects
   - Environment sensitivity
Off-topic: Virtual Machine migration

- VM migration is a practical way of process migration between hosts
VM migration is too much larger

• Too many resources are migrated for malware analyzing

• VM solution forces additional system to end-users and employer
  – Increasing complexity, Maintainability and cost
Our solution: Using process-level sandbox

- CPU emulator-based sandbox is convenient for process migration
  - Grubbed all contexts
  - User-mode emulator virtualize process related kernel objects
CPU emulator-based sandbox
Process migration using CPU emulator-based sandbox
Malware freeze-drying

- Sandbox suspends target program when a trigger event occurred
- A suspended trigger is anti-analyzing behavior [2]

Diagram:
- Sandbox
  - Execution Context
  - Target Process
  - Memory Context
  - Runtime & Libraries (Virtualized)
  - FILE
  - HEAP
  - Kernel Objects & Entities (virtualized)
- Serialize
  - Packed Living Malware
Live malware defrosting

- Sandbox resumed packed living malware
- Reconstructing address gaps
But...

- Migrated malware will probably executes anti-analyzing(anti-sandbox) continuously

- The system needs anti-anti-sandbox arming
Challenges (updated)

1. Process migration is very difficult
   → Using CPU emulator-based sandbox

2. Arming against anti-sandbox
Challenges Ⅱ

ANTI-ANTI-SANDBOX ARMING
Taxonomy of anti-sandbox techniques

• Anti-sandbox maneuver
  – Stalling code [3]
  – Environment awareness [4][5]
    • Using result of sandbox detection
  – (User interaction checks)

• Sandbox (debug/sandbox/vm) detection
  – Artifact fingerprinting[5][6]
  – Execution incongruousness[7][8]
  – Platform stimulation[9][10]
Stalling code

- Evasive malware\[^2\] often uses
  - A sandbox limits malware execution time

- Stalling code detection and avoiding techniques already proposed\[^3\]

```c
unsigned count, t;
void helper() {
  t = GetTickCount();
  t++;
  t++;
  t = GetTickCount();
}
void delay() {
  count=0x1;
  do {
    helper(); // equal nop
    count++;
  } while (count!=0xe4e1c1);
}
```

Stalling code in W32.DelfInj\[^3\]
Environment awareness

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

![Diagram showing the process of initializing, sandboxing, detection, and decoy routine](image)
Sandbox (debug/sandbox/vm) detection

- Environment aware Malware
- Artifact Fingerprinting

- VM related Artifacts
- Sandbox specific Artifacts

- Execution Incongruousness
- Platform Stimulation
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 Incongruousness

- Using clock count differential
  - Traditional anti-debug technique
- Redpill\[8\]
  - Using LDT/GDT and IDT incongruousness

```
<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>400022A2</td>
<td>60 0F31</td>
<td>PUSHAD 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>
```
Platform stimulation

- Using virtual machine implementation differentials
  - Like CPUID instruction result
  - Interesting research here: Cardinal Pill Testing[9]
Our solution: Anti-anti-sandbox arming

- **Hiding an artifact using API proxies**
- Stalling code detection and evasion (future work)
  - Following prior works
- Faithful CPU emulation (future work)
  - Following prior works and showing GUTS
IMPLEMENTATION
Sweetspot Overview

- Sweetspot have two sandbox, **Freeze-drying Sandbox** and **Defroster Sandbox**
- Sandboxes are 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[11]
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

Virtualized Resource Manager

API Emulation Components

- Virtual kernel32 Functions
- Virtual user32.dll Functions
- Virtual advapi32.dll Functions
- Virtual msvcrtd.dll Functions

TARGET Process

Kernel32, dll
User32.dll
Advapi32.dll
Msvcrtd.dll

...
IA-32 CPU Emulator: Virtual resource handling

- File system is almost virtualized
- Registry hive is almost virtualized
- GUI components and user interaction function is virtualized partially
- media components is not virtualized (squashing request)
**Sweetspot: Malware Live Capturing System**

- **Freeze-dryer**
  - Serializing process contexts and execution file if detected suspend trigger
  - All malware activity sealed in the sandbox

- **Defroster**
  - Restoring execution context
  - Address reconstruction
  - API-proxies for faking an artifacts
Freeze-dryer

- End-user can use like an anti-virus’s file scanner

- Freeze dryer serialize process context if detects anti-sandbox behavior occurred
  - Dumping all VA space anyway

- Using msgpack[^12] library for serialization
Defined suspend trigger (Work in progress)

• Specific API-call
  – GetVolumeNameForVolumeMountPoint()
  – GetVolumeInformation()

• Specific API-call and its arguments
  – Searching vm-related artifacts
    • Virtual file system and virtual registry hive except finding sandbox artifacts

• Detecting stalling code(WiP)
Defroster – Execution replaying

1. Unpacking process contexts (incl. execution file)
   - Allocating sandbox’s heap

2. Loading execution file before entry point

3. Restoring current process context from unpacked contexts
   - Remapping address in unpacked process contexts
     - Covering all virtual address space
Demo: Process migrated!
API Proxies

- Malware can access specified directories on Defroster
  - Like `%APPDATA%`

- API Proxies enable to provide arbitrarily resources for malware
Anti anti sandbox arming using API Proxies

• Defroster performs play innocent with sandbox/vm related artifacts
  – No vm-related artifact exist in sandbox’s virtual file system and virtual registry hive

• For faking an artifacts
  – Fake artifacts mounting virtual file system before malware resuming
Limitations

- The original CPU emulator supports a limited API
  - ex). Cannot CreateProcess and CreateThread

- The original CPU emulator supports a limited CPU instruction
  - ex). Cannot complete emulation with SSE instruction

- Anti-anti sandbox implementation is not enough

- API Proxies not supported Network API(winsock2) yet
Demonstrations

• Simple program (incl. heap and handle migration)
• Anti-anti-sandbox PoC
Future work

• Improving anti-sandbox detection and anti anti-sandbox
  – Stalling code detection and evasion
  – More faithful CPU/API emulation

• Improving API proxies utility

• Defroster-based stealth debugger
Conclusions

• This is proof of concept of live malware capturing using process migration with CPU emulator-based sandbox

• We introduced anti-sandbox taxonomy and proposed API-proxy based countering approach
References

• [1]: Analyzing Environment-Aware Malware, Lastline, 2014.05.25(viewed)  


• [7]: Aurélien Wailly. Malware vs Virtualization The endless cat and mouse play, 2014.05.25(viewed)  


• [11]: IDA Boch PE operation mode  
  https://www.hex-rays.com/products/ida/support/idadoc/1332.shtml

• [12]: MessagePack, 2014/09/28(viewed)  
  http://msgpack.org/
Thank you!