COUNTERING INNOVATIVE SANDBOX EVASION TECHNIQUES USED BY MALWARE

Frederic Besler, Carsten Willems, and Ralf Hund
• State of the Art malware analysis systems use hybrids of static and dynamic analysis
• Dynamic analysis usually in form of behavior based detection aka. sandbox
• Malware tries to detect and evade sandbox
• We need to talk about sandbox evasion and anti-evasion
1. Detect the Sandbox
2. Defeat the Monitor
3. Context Awareness

* Categories are fuzzy and overlap
1. Detect the Sandbox
1. Detect the Sandbox

• Actively detect the analysis environment
  - If in analysis environment: Quit or act benign
  - Else: Execute the malicious code

• Most commonly used methods:
  - Detect underlying technology, e.g. hypervisor or emulator
  - Detect specific sandbox product, e.g. check for files or processes
  - Detect artificial environments, e.g. check for “clean” system
Detect Virtualization via Artifacts

- Examples:
  - Registry, Files (Software, Drivers), Processes
  - Device Names, Device IDs, MACs
  - SMBIOS, ACPI tables
  - IO Ports (VMware, KVM/Qemu)

Malware

✓ Large attack surface
✓ Essential virtual devices are hard to hide

Clean registry, remove files, ...
Don’t use paravirt. Devices
Use custom hypervisor

REG_BLACKLIST = ["vbox", "vmware", "bochs", "qemu", "red.?hat", "virt.?io", "kvmnet", "netkvm", ...]
PROC_BLACKLIST = ["vbox.*\.*.exe", "vmware.*\.*.exe", "prl_.*\.*.exe", ...]
VENDOR_IDS = {
  0x1a71: "XenSource, Inc."
  0x5853: "XenSource, Inc."
  0xfffd: "XenSource, Inc."
  0x15ad: "VMware"
  0xffff: "VMWare Inc (temporary ID)"
  # VBox
  0x80ee: "InnoTek Systemberatung GmbH"
  # QEMU/KVM
  0x1af4: "Red Hat, Inc."
  0x1b36: "Red Hat, Inc."
  0x1ab8: "Parallels, Inc."
}
1. Detect the Sandbox – Virtualization

CPUID
- CPU tells guest that it is virtualized
- Examples:
  - CPUID Hypervisor Bit
  - CPUID Hypervisor Brand String
  - Other CPUID Artifacts (e.g. Function 0x80000009)

```c
struct {
  int EAX;
  int EBX;
  int ECX;
  int EDX;
} out;

__cpuidex((int*)&out, 1, 0);
if (!(out.ECX >> 31))
do_evil();
```

Real System

| HV Bit: 0 |
| HV Brand: '\@\@' |

Virtualized System

| HV Bit: 1 |
| HV Brand: 'VBoxVBoxVBox' |

- No false positives
- Can be disabled (requires support of HV)
- Disable where possible
1. Detect the Sandbox – Virtualization

Side Channels
- HV consumes CPU cycles
- HV and VM(s) share resources
- Examples:
  - Time instructions
  - Time caching side effects

Malware
- Very hard to prevent
- Specific to hardware
- Can be noisy

Countermeasures
- Heuristics to detect specific attacks
- Spoof timer values

Real System
- CPID avg. time: 145

Virtualized System
- CPID avg. time: 4581

Code snippet:
```c
long long s, acc = 0;
int out[4];
for (int i = 0; i < 100; ++i) {
    s = __rdtsc();
    __cpuidex(out, 0, 0);
    acc += __rdtsc() - s;
}
if (acc / 100 < 200) do_evil();
```
Unusual Hardware Characteristics
- Examples:
  - CPU type, number cores
  - HD space, Ram
  - Printers
  - USB devices/sticks
  - Display resolution

1. Detect the Sandbox – Artificial Environment

wmic cpu get NumberOfCores
wmic memorychip get capacity
wmic diskdrive get size
wmic printer get name
wmic desktopmonitor get screenheight, screenwidth
wmic path win32_VideoController get name

✓ Huge attack surface
✓ Expensive to fake

Countermeasures:
- Give VM realistic resources
- Fake values

Malware
1. Detect the Sandbox – Artificial Environment

User Artifacts
- Examples:
  - Installed software
  - Cookies, entered URLs
  - Recently used files
  - Entered commands

```
REG_KEYS = [
  # Office MRU lists
  "SOFTWARE\Microsoft\Office\VERSION\PRODUCT\File MRU",
  "SOFTWARE\Microsoft\Office\VERSION\PRODUCT\Place MRU",
  "SOFTWARE\Microsoft\Office\VERSION\PRODUCT\User MRU",
  # Explorer Artifacts
  "SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\TypedPaths",
  "SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\WordWheelQuery",
  "SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\RunMRU",
  "SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\ComDlg32\*",
  # IE
  "SOFTWARE\Microsoft\Internet Explorer\TypedURLs",
  "SOFTWARE\Microsoft\Internet Explorer\TypedURLsTime",
  ...
]
```

✓ Huge attack surface
✗ Can be faked (laborious but not complex)

Current.
- Add random data
2. Defeat the Monitor
2. Defeat the Monitor

- Exploit weaknesses in the monitor
- Most attacks only work against a single product or underlying technology
- Some methods only effective vs. in-guest monitoring (i.e. Hooking)
- Others more generic and work in all sandboxes
2. Defeat the Monitor – Hooking

Remove Hooks

- **Examples:**
  - Restore instructions from disk (check signature 😊)
  - Restore IAT, EAT

Malware

- Noticeable
- Can cause instability

Counterm.

- Check hook integrity

Hooked Function

<table>
<thead>
<tr>
<th>Regular</th>
<th>Hooked</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00: call API</td>
<td>0x00: jmp Hook</td>
</tr>
<tr>
<td>0x00: mov edi, edi</td>
<td>0x05: push ecx</td>
</tr>
<tr>
<td>0x02: push ebp</td>
<td>0x07: ... more ...</td>
</tr>
<tr>
<td>0x03: mov ebp, esp</td>
<td>0xff: ret</td>
</tr>
<tr>
<td>0x05: push ecx</td>
<td></td>
</tr>
<tr>
<td>0x07: ... more ...</td>
<td></td>
</tr>
<tr>
<td>0xff: ret</td>
<td></td>
</tr>
</tbody>
</table>

Reconstruct Instructions
2. Defeat the Monitor – Hooking

Circumvent Hooks

- **Examples:**
  - Use System calls
  - Use undocumented APIs
  - Unaligned function calls

- **Malware:**
  - Hard to counter
  - Can cause instability

- **Countermeasures:**
  - Move hooks deeper into the system

```
0x00: call API
0x02: push ebp
0x03: mov ebp, esp
0x05: push ecx
0x07: ... more ...
0xff: ret
```

```
0x00: jmp Hook
0x05: push ecx
0x07: ... more ...
0xff: ret
```

```
0x00: push 0x0c
0x02: mov edi, edi
0x04: push ebp
0x05: mov ebp, esp
0x07: jmp API+0x5
```

“Unaligned Call”
2. Defeat the Monitor – Generic

Delay Execution

• Execute malicious code after timeout
• Implementations range from simple to complex
• Problems:
  - Multitude of different time sources
    ▪ GetTickCount, RDTSC, SharedUserData, internet, ...
  - Multitude of timer functions
    ▪ Sleep, WaitFor*Object, SetTimer, timeSetEvent, ...

✓ Hard to counter (if done right)
✓ Easy to implement

```c
Sleep(10 * 60 * 1000);
do_evil();
```
2. Defeat the Monitor – Generic

Delay Execution

Fast Forward Sandbox Execution
- Patch particular calls
  - Problems: Risk of inconsistent state, e.g., all time sources need to be synced
- Manipulate timer behavior
  - Problems: Could cause system instabilities, not trivial to implement
- Manipulate whole system time
  - Problems: Could cause system instabilities, increases system load

```c
DWORD evil_thread(void *p) {
    Sleep(10 * 60 * 1000);
    do_evil();
}

CreateThread(..., &evil_thread, ...);
Sleep(10 * 3600 * 1000);
TerminateProcess(-1, 0);
```
Exploit Monitoring Costs

- Monitoring consumes CPU time ➞ malware execution takes longer
- Use code that is computationally intensive for the monitor
- Postpone malicious behavior until after sandbox timeout
- Examples:
  - API hammering

2. Defeat the Monitor – Generic

Easy to implement
- Generally very noisy

Adaptively (de) activate the monitor
Probabilistic JS decoding
- Decoder is generated in brute-force fashion
- Without monitoring, payload is generated in one minute
- With monitoring:
  ▪ Monitoring the interpreter costs time
  ▪ `eval()` calls are expensive for analysis environments
  ▪ Result: Execution time increases by an order of one magnitude

```javascript
function e37b0(){
    return new Array('a7493','ret','ec468')[Math.floor(Math.random()*3)];
}
function b32eb(){
    return new Array('a7493','arCode(parseIn','ec468')[Math.floor(Math.random()*3)];
}
function e46ef(){
    return new Array('a7493','.substr(2,2),1','ec468','ec468','ec468')[Math.floor(Math.random()*5)];
}
function eb6e0(aad69){
    return (new Function('ec071','e37b0()''+urn''+ String'+'6').fromCh+b32eb()'+ec071'+e46ef()+'6')(aad69));
}
```
3. Context Awareness
3. Context Awareness

- Neither detect nor defeat sandbox
- Instead execute payload only in certain context
- Different variants:
  - Wait to trigger condition, e.g., user interaction
  - Check for specific environment, e.g., company domain
3. Context Awareness – User Interaction

• Specific User interaction
  - Automated Sandboxes can not interact meaningfully
  - Examples:
    ▪ Fake installers
    ▪ Documents requiring interaction
    ▪ Only interact with an opened browser

✓ Hard to fake ‘meaningful’ interaction
✗ Makes the malware visible
✗ Requires user ‘cooperation’

• Locate and click buttons
• Automatic mouse movement
3. Context Awareness – Explicit Checks

Context checks

• Examples:
  - Date
  - Time zone
  - Username

  ✓ Easy to implement

Malware

  • Use symbolic execution to find constraints
  • Problems: Very costly, hard to resolve complex scenarios

```c
SYSTEMTIME st;
GetSystemTime(&st);
if (st.wYear == 2017 && st.wMonth == 6 && st.wDay == 13)
do_evil();
```
Environmental keying

- Malicious payload is encrypted
- Decryption key derived from environment markers
- Examples:
  - Gauss: A characteristic combination of path and folder were chosen to generate EMBEDDED_HASH
  - Ebowla: Framework to build environmental keyed payloads

✓ Very hard to detect and defeat automatically
  ○ Inhibits spreading

Malware

- Implement heuristics which are looking for use of cryptography
- Detonate malware in target(like) environment

Countermeasures

for path in PATH:
    for folder in PROGRAMFILES:
        if hash(path + folder) == EMBEDDED_HASH:
            decrypt_and_do_evil(path, folder)
Summary
Summary

- There is no silver bullet ➔ each technique requires specific handling
- Many evasion attempts are noisy, therefore detectable (at least from ring -1)

We should

- Use realistic environments, e.g. cookies, MRU, ...
- Use non fingerprint-able environments, e.g. randomize files, usernames, ...
- Detonate in expected target environment, e.g., golden image
- Other:
  - Analyze within different environments, e.g., OS patch level, network config
  - Constantly adapt new anti evasion techniques
Thank you for your attention!

fbesler@vmray.com
Tools

- https://github.com/LordNoteworthy/al-khaser
- https://github.com/AlicanAkyol/sems
- https://github.com/Th4nat0s/No_Sandboxes
- https://github.com/Genetic-Malware/Ebowla
- https://github.com/a0rtega/pafish
- https://github.com/CheckPointSW/InviZzzible
- https://github.com/hfiref0x/VMDE
References

- https://www.vmray.com/blog/sandbox-evasion-techniques-part-1/
- https://www.joesecurity.org/blog/3660886847485093803