COUNTERING INNOVATIVE SANDBOX EVASION TECHNIQUES USED BY MALWARE

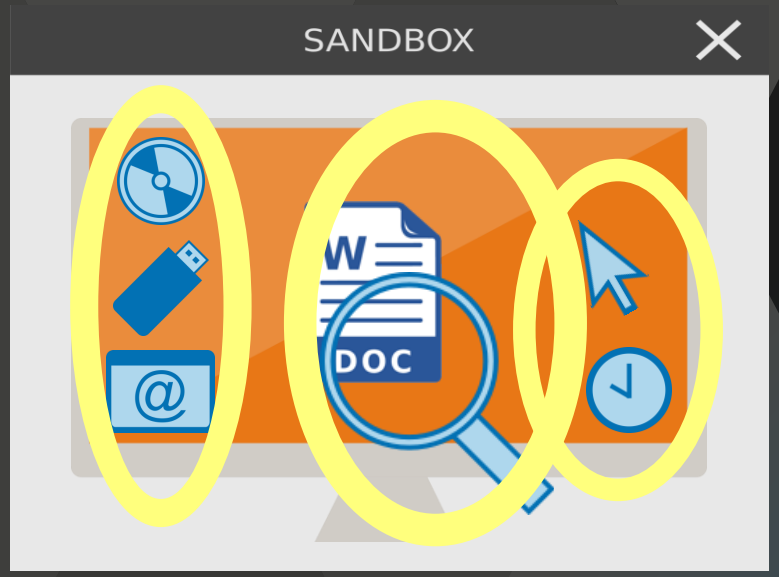
Frederic Besler, Carsten Willems, and Ralf Hund

Introduction

VMRAY

- 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

Evasion Techniques - Categories



1. Detect the Sandbox

2. Defeat the Monitor

3. Context Awareness

* Categories are fuzzy and overlap

1. Detect the Sandbox

1. Detect the Sandbox

VMRAY

- 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

1. Detect the Sandbox – Virtualization Artifacts

VMRAY

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

Counterm.

- 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"
    0xfffe:  "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

VMRAY

CPUID

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

```
struct { int EAX; int EBX; int ECX; int EDX; } out;
__cpuidex((int*)&out, 1, 0);
if (!(out.ECX >> 31))
    do_evil();
```

```
char out[4 * 4 + 1] = { 0 };
__cpuidex((int*)out, 0x40000000, 0);
if (strlen(out + 4) >= 4)
    do_evil();
```

Real System	Virtualized System
HV Bit: 0 HV Brand: '@?'	HV Bit: 1 HV Brand: 'VBoxVBoxVBox'

Malware

✓ No false positives

✗ Can be disabled (requires support of HV)

Counterm.

- 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

```
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();
```

Real System	Virtualized System
CPUID avg. time: 145	CPUID avg. time: 4581

Malware

- ✓ Very hard to prevent
- ✗ Specific to hardware
- ✗ Can be noisy

Counterm.

- Heuristics to detect specific attacks
- Spoof timer values

1. Detect the Sandbox – Artificial Environment

VMRAY

Unusual Hardware Characteristics

- Examples:
  - CPU type, number cores
  - HD space, Ram
  - Printers
  - USB devices/sticks
  - Display resolution

```
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
```

Malware

- ✓ Huge attack surface
- ✓ Expensive to fake

Counterm.

- Give VM realistic resources
- Fake values

1. Detect the Sandbox – Artificial Environment

VMRAY

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",
    ...
]
```

Malware

✓ Huge attack surface

✗ Can be faked (laborious but not complex)

Counterm.

- 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

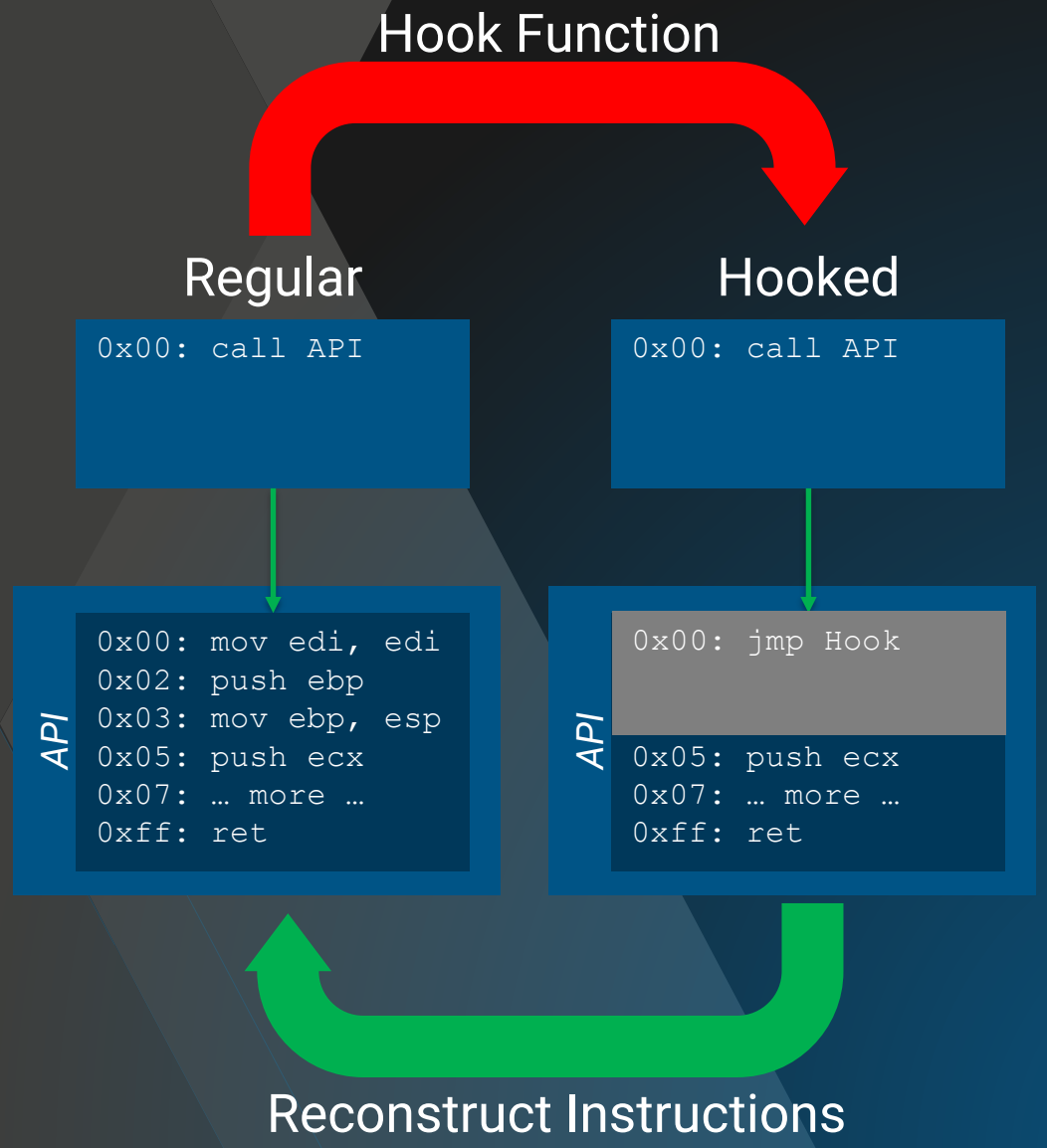
- Examples:
  - Restore instructions from disk (check signature ☺)
  - Restore IAT, EAT

Malware

× Noticeable
× Can cause instability

Counterm.

- Check hook integrity



2. Defeat the Monitor – Hooking

Circumvent Hooks

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

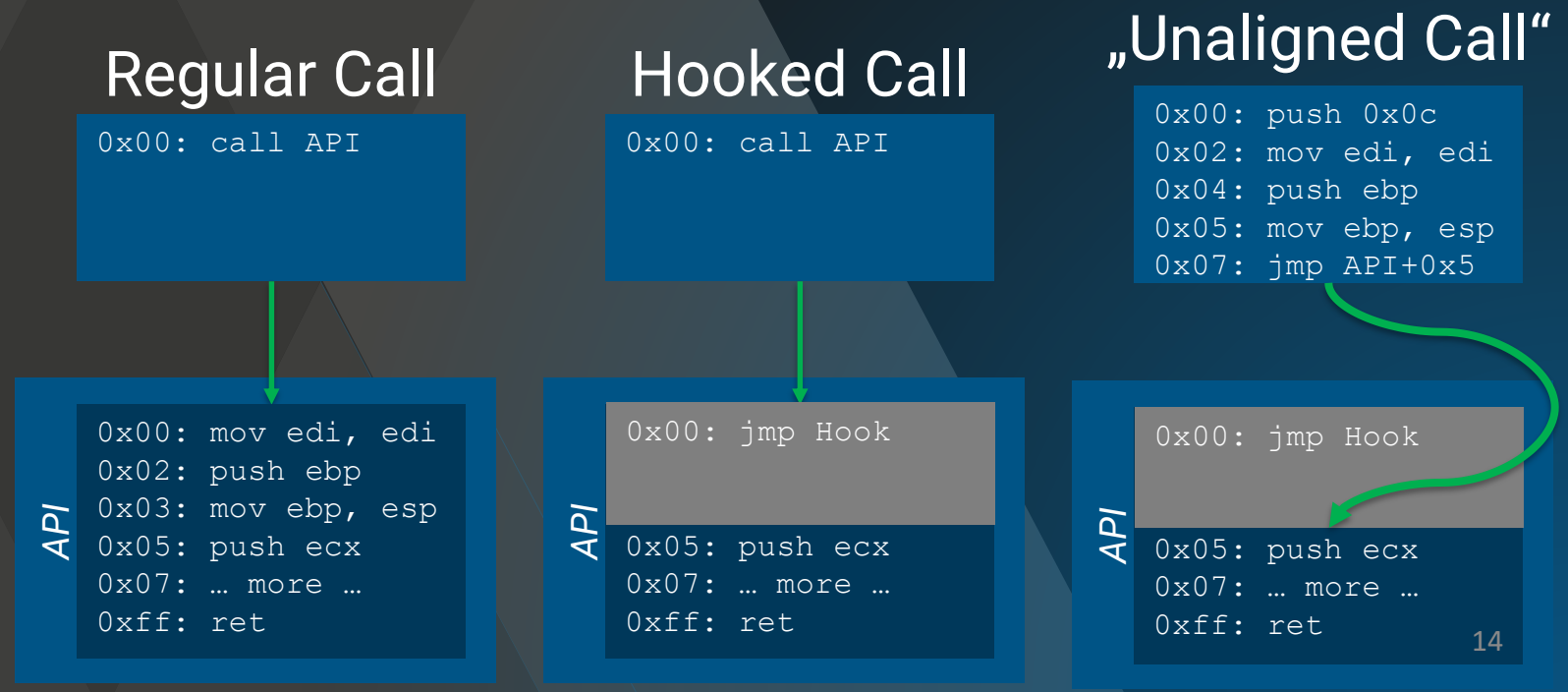
Malware

✓ Hard to counter

✗ Can cause instability

Counterm.

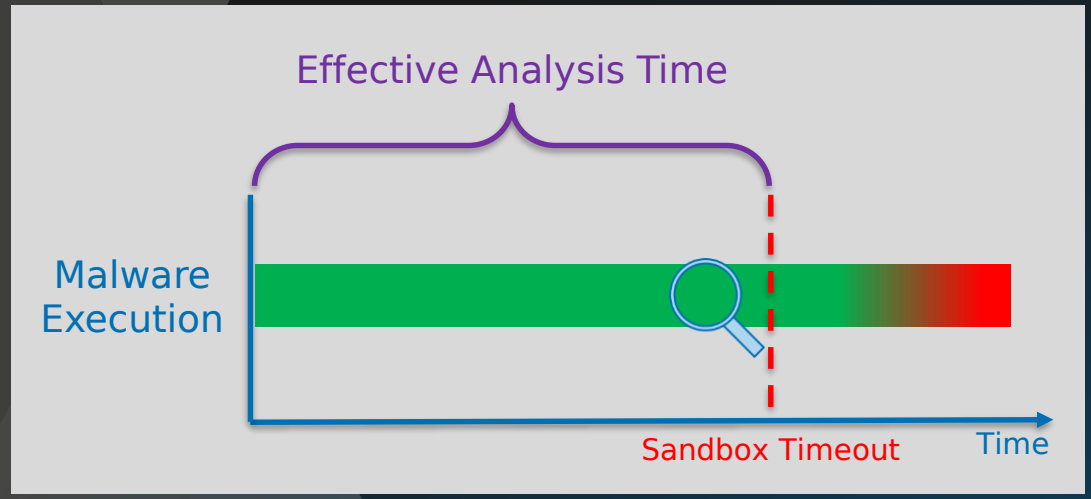
- Move hooks deeper into the system



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, …




```
Sleep(10 * 60 * 1000);
do_evil();
```

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

2. Defeat the Monitor – Generic

Delay Execution

Counterm.

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

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

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

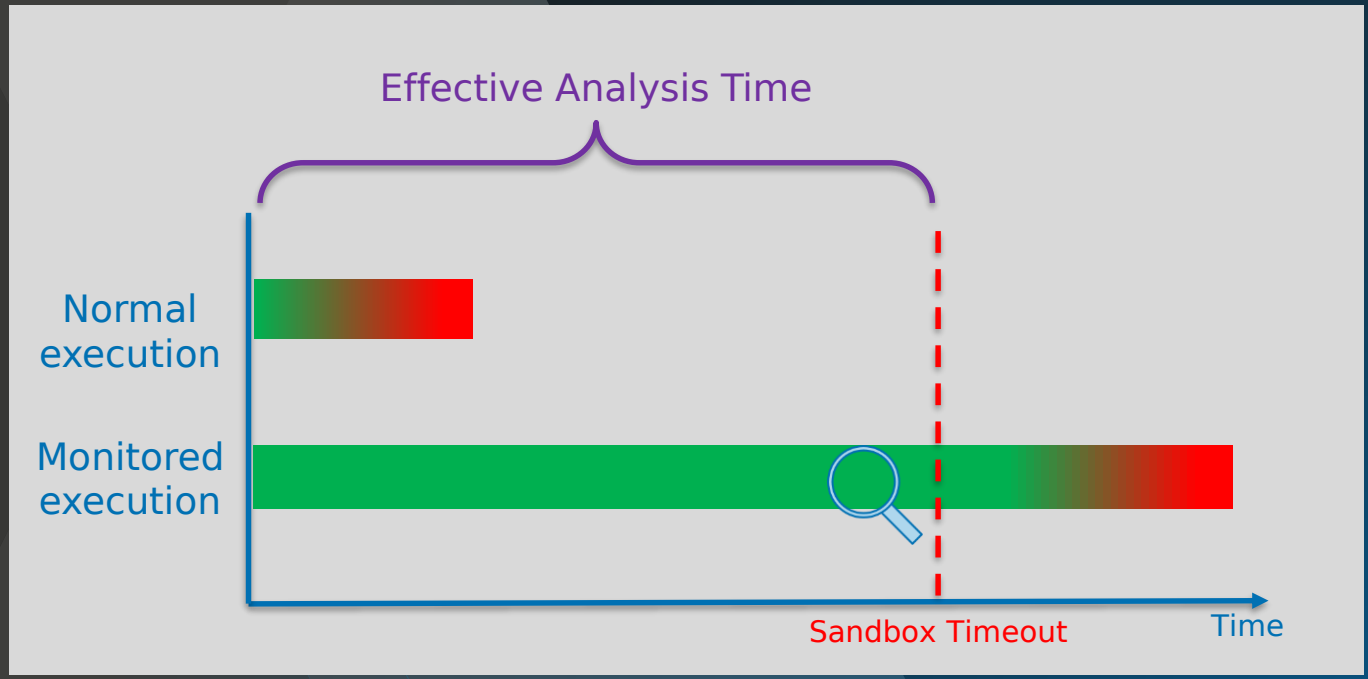
2. Defeat the Monitor – Generic

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

Malware
✓ Easy to implement
✗ Generally very noisy

Counterm.
- Adaptively (de) activate the monitor



2. Defeat the Monitor – Case Study

VMRAY

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

```
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'+''+'.fromCh'+b32eb()+'t(ec071'+e46ef()+'6)^6)')(aad69));
}
```

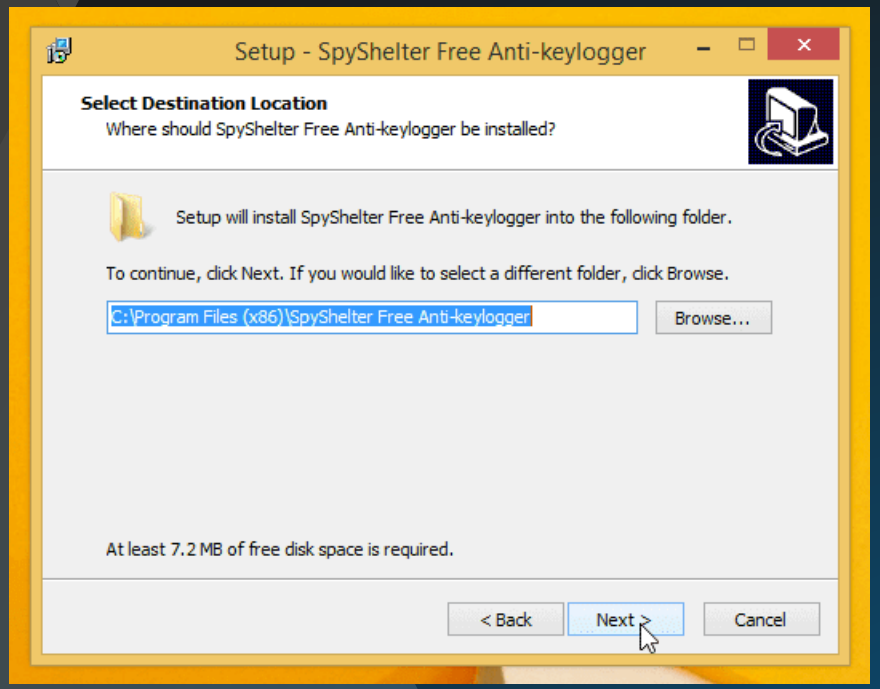
3. Context Awareness

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



Malware

- ✓ Hard to fake 'meaningful' interaction
- ✗ Makes the malware visible
- ✗ Requires user 'cooperation'

Counterm.

- Locate and click buttons
- Automatic mouse movement

3. Context Awareness – Explicit Checks

Context checks

- Examples:
  - Date
  - Time zone
  - Username

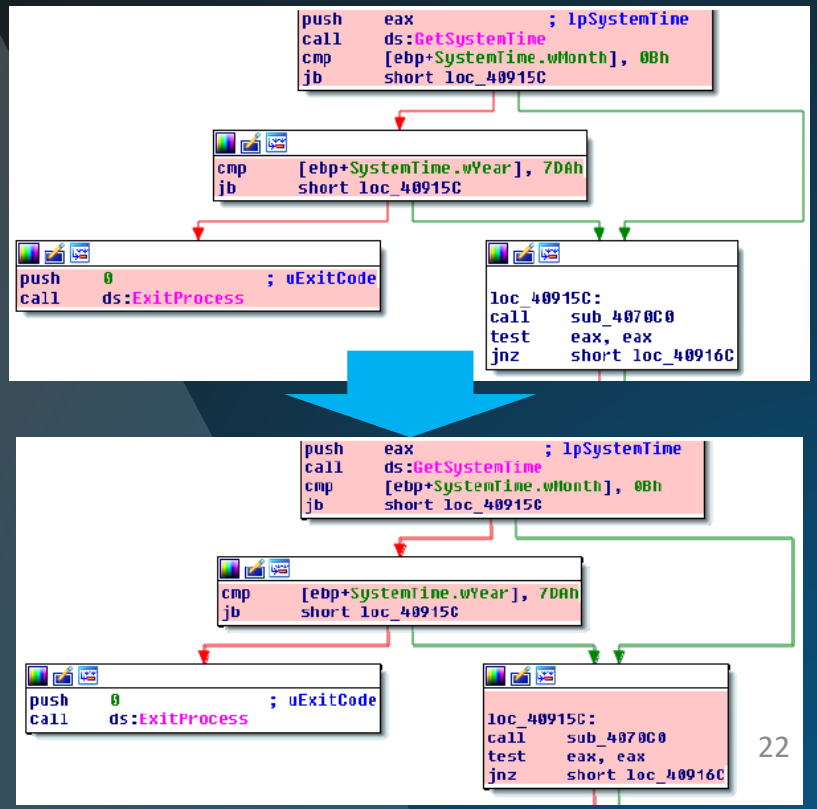
Malware

✓ Easy to implement

Counterm.

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

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



3. Context Awareness – Implicit Checks

VMRAY

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

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

Malware

✓ Very hard to detect and defeat automatically

○ Inhibits spreading

Counterm.

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

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.vmray.com/blog/sandbox-evasion-techniques-part-2/
- https://www.vmray.com/blog/sandbox-evasion-techniques-part-3/
- https://www.vmray.com/blog/sandbox-evasion-techniques-part-4/
- https://www.joesecurity.org/blog/3660886847485093803
- https://www.botconf.eu/2015/sandbox-detection-for-the-masses-leak-abuse-test/
- https://www.blackhat.com/docs/asia-15/materials/asia-15-Chubachi-Slime-Automated-Anti-Sandboxing-Disarmament-System.pdf
- https://www.usenix.org/sites/default/files/conference/protected-files/sec14_slides_shi-hao.pdf
- https://aurelien.wail.ly/publications/hip-2013-slides.html
- https://www.botconf.eu/wp-content/uploads/2014/12/2014-2.7-Bypassing-Sandboxes-for-Fun.pdf
- https://www.virusbulletin.com/blog/2016/december/vb2016-paper-defeating-sandbox-evasion-how-increase-successful-emulation-rate-your-virtualized-environment/
- https://www.securitee.org/files/wearntear-oakland2017.pdf
- https://securelist.com/33561/the-mystery-of-the-encrypted-gauss-payload-5/