Detecting & Evading Automated Malware Analysis

Alexei Bulazzel
About Me

• Researcher for Crowdstrike SRI team
• Consultant at River Loop Security

• 2015 graduate from Rensselaer Polytechnic Institute (RPI), worked on malware evasion with Jeremy Blackthorne and Dr. Bülent Yener
Outline

1. Introduction to Malware Evasion
   1. Intro
   2. Types of fingerprints
   3. Fingerprint discovery
   4. Antivirus emulators

2. AVLeak

3. Evasion Mitigation & Detection Research

4. Conclusion
Automated Dynamic Malware Analysis

• Static signature-based malware detection is difficult given 1M+ new binaries daily

• Automated dynamic analysis (aka “sandbox analysis”) runs malware and observes runtime behaviors
# Traditional Malware Sandbox

<table>
<thead>
<tr>
<th>Analysis Process</th>
<th>Malware</th>
<th>User Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis Driver</td>
<td>Virtualized OS</td>
<td>User Process</td>
</tr>
<tr>
<td>QEMU</td>
<td>Virtualized Hardware</td>
<td>User Process</td>
</tr>
<tr>
<td>cuckoo</td>
<td>Malware Sandbox</td>
<td>User Process</td>
</tr>
<tr>
<td></td>
<td>Operating System or Hypervisor</td>
<td>User Process</td>
</tr>
<tr>
<td></td>
<td>Hardware</td>
<td>User Process</td>
</tr>
</tbody>
</table>
Problem

• Malware can detect analysis and behave benignly to evade it

• Over 80% of malware exhibited evasive behavior in 2nd half of 2015

```python
if(detect_analysis()){
    exit()
}
else{
    be_malicious()
}
```
Outline

1. Introduction to Malware Evasion
   1. Intro
   2. Types of fingerprints
   3. Fingerprint discovery
   4. Antivirus emulators
2. AVLeak
3. Evasion Mitigation & Detection Research
4. Conclusion
Fingerprinting Analysis Systems

- Environmental Artifacts
- Timing
- Process Introspection
- CPU “Red Pills”
- Reverse Turing Tests
- Network Artifacts
- Sensor Emulation
Environmental Artifacts

- Simplest class of fingerprints
- Explicit
  - Unique username, computer name, settings...
  - Analysis related drivers, processes, files, registry settings
  - Virtualization related artifacts (drivers, registry entries, mutexes, processes, etc...)
- Heuristic
  - Single CPU systems
  - Small amounts of RAM
  - “VMware virtual platform always emulates an i440bx chipset, leading to absurd hardware configurations: two AMD Opteron CPUs and 8 GB of RAM in an Intel motherboard from the Clinton administration, for instance.” - Garfinkel, et al.
Timing

• Virtualization introduces timing discrepancies vs real hardware
• Complex attacks are possible, but real world malware often uses simple ones
• The hardest class of fingerprint to mitigate
  – Efforts to mask time discrepancies can in fact cause more problems
• Any outside communication can be used to measure time
CPU Red Pills

• x86 has 1500+ instructions and is not fully or consistently documented
• Incorrect instruction emulation:
  – Failure to set status flags
  – Erroneous memory writes
  – Incorrect exception behavior
• Non-semantic differences:
  – i.e., SIDT/SGDT
Process Introspection

- *In-process* memory artifacts
- Injected DLLs
- Function hooks
- Page permissions
- Stack inconsistency
  - Check below $SP$
- Inability to properly handle self-modifying code
Reverse Turing Tests

- Malware (computer) tests for real human presence
- Actively prompt the user for input
- Mouse or keyboard input
- Copy-paste clipboard
- “Wear-and-tear” artifacts: login times, recently opened files, web history, photos, certificate revocation list, crash dumps
Network Artifacts

- Fixed IP addresses
- Network isolation
- Server or network stack emulation
- Extremely fast network connectivity
- Can be used for external timing detection
Outline

1. Introduction to Malware Evasion
   1. Intro
   2. Types of fingerprints
   3. Fingerprint discovery
   4. Antivirus emulators

2. AVLeak

3. Evasion Mitigation & Detection Research

4. Conclusion
Fingerprint Discovery - Static

- Look at underlying open source or freely available technologies

<table>
<thead>
<tr>
<th>Analysis Process</th>
<th>User Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malware</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Analysis Driver</th>
<th>User Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtualized OS</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EMU</th>
<th>User Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtualized Hardware</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cuckoo</th>
<th>User Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malware Sandbox</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Windows</th>
<th>Operating System or Hypervisor</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Xen</th>
<th>VT-8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware</td>
<td></td>
</tr>
</tbody>
</table>
Easy To Analyze
Fingerprint Discovery - Dynamic

**Sandbox fingerprints:**
Username: SandboxUser
Process: debug_process.exe
Driver: analysis_driver.sys

### Analysis Process

<table>
<thead>
<tr>
<th>Analysis Driver</th>
<th>User Process</th>
<th>User Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malware</td>
<td>Virtualized OS</td>
<td>User Process</td>
</tr>
<tr>
<td>EMU</td>
<td>Virtualized Hardware</td>
<td>User Process</td>
</tr>
<tr>
<td>cuckoo</td>
<td>Malware Sandbox</td>
<td>User Process</td>
</tr>
<tr>
<td>Windows</td>
<td>Operating System or Hypervisor</td>
<td>User Process</td>
</tr>
<tr>
<td>Linux</td>
<td>VT-X</td>
<td>Hardware</td>
</tr>
</tbody>
</table>
Fingerprint Discovery - Dynamic

Sandbox fingerprints:
Username: SandboxUser
Process: debug_process.exe
Driver: analysis_driver.sys

Fingerprint Discovery - Dynamic

Sandbox fingerprints:
Username: SandboxUser
Process: debug_process.exe
Driver: analysis_driver.sys

Analysis report:
Created file: SandboxUser.txt
Opened mutex: debug_process.exe
Edited registry: analysis_driver.sys

Outline

1. Introduction to Malware Evasion
   1. Intro
   2. Types of fingerprints
   3. Fingerprint discovery
   4. Antivirus emulators

2. AVLeak

3. Evasion Mitigation & Detection Research

4. Conclusion
Why Is AV Interesting?

- Actually protects *real computers*
- No good way to identify fingerprints to use for analysis
  - Reverse engineering - too hard
  - Side channel attacks - too slow
AV Emulator Limitations

• Run on home computers - quickly and without a lot of memory
• Respect copyright and software licensing
  – No QEMU/Xen/VMWare
  – Can’t use real Windows OS code
• Generally poor software engineering in the AV industry
  – Trying to keep up with 1M+ malware samples per day means its hard to maintain old code
  – Look at Tavis Ormandy’s Project Zero work for examples...
# Traditional Malware Sandbox

<table>
<thead>
<tr>
<th>Analysis Process</th>
<th>Malware</th>
<th>User Process</th>
<th>User Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis Driver</td>
<td>Virtualized OS</td>
<td>QEMU</td>
<td>Virtualized Hardware</td>
</tr>
<tr>
<td></td>
<td>Malware Sandbox</td>
<td>cuckoo</td>
<td>VT-X</td>
</tr>
<tr>
<td></td>
<td>Operating System or Hypervisor</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hardware</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## AV Emulator

<table>
<thead>
<tr>
<th>x86 Emulator</th>
<th>Environment</th>
<th>Usermode WinAPI Emulator</th>
<th>User Process</th>
<th>User Process</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Intel Logo" /></td>
<td><img src="image" alt="Computer, Files, Web" /></td>
<td><img src="image" alt="Usermode WinAPI Emulator" /></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Malware**
- **Antivirus Emulator**
- **Operating System**
- **Hardware**
AV Emulator

Malware

x86 Emulator | Environment | Usermode WinAPI Emulator | User Process | User Process

Antivirus Emulator

Operating System

Hardware
## AV Emulator

<table>
<thead>
<tr>
<th>x86 Emulator</th>
<th>Environment</th>
<th>Usermode WinAPI Emulator</th>
<th>User Process</th>
<th>User Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Malware**

**Antivirus Emulator**

**Operating System**

**Hardware**
AV Emulator

Only output is analysis report

Analysis report:
Dropped: Trojan.Infector.BAT.ABC123
Dropped: APT1337.Backdoor.2
Dropped: CryptoLocker.Downloader.K
Reversing AV Emulators

- Time consuming
- Expensive tools
- Expert knowledge
  - RE, AV, x86, Windows internals, malware behavior, anti-analysis
- Limited Lifespan - frequent updates
I've done this same exercise with anti-virus engines on a number of occasions. Generally the steps I use are:

1. Identify the CPU/Windows emulator. This is generally the hardest part. Look at filenames, and also grep the disassembly for large switch statements. Find the switches that have 200 or more cases and examine them individually. At least one of them will be related to decoding the single-byte x86 opcodes.

2. Find the dispatcher for the CALL instruction. Usually it has special processing to determine whether a fixed address is being called. If this approach yields no fruit, look at the strings in the surrounding modules to see anything that is obviously related to some Windows API.

3. Game over. AV engines differ from the real processor and a genuine copy of Windows in many easily-discriminable ways. Things to inspect: pass bogus arguments to the APIs and see if they handle erroneous conditions correctly (they never do). See if your emulator models the AF flag. Look up the exception behavior of a complex instruction and see if your emulator implements it properly. Look at the implementations of GetTickCount and GetLastError specifically as these are usually miserably broken.
Analysis report:
Dropped: Trojan.Infector.BAT.ABC123
Dropped: APT1337.Backdoor.2
Dropped: CryptoLocker.Downloader.K
Black Box Testing

Is func_x() emulated correctly?
Black Box Testing

Is `func_x()` emulated correctly?

```c
if (func_x() != CORRECT)
{
    drop_malware();
}
else
{
    exit();
}
```
Black Box Testing

AV Emulator

```c
if (func_x() != CORRECT)
{
    drop_malware();
}
else
{
    exit();
}
```

Is `func_x()` emulated correctly?
Black Box Testing

**AV Emulator**

```c
if (func_x() != CORRECT) {
    drop_malware();
} else {
    exit();
}
```

Is `func_x()` emulated correctly?

Malware Found (No)
Black Box Testing

AV Emulator

```c
if (func_x() != CORRECT)
{
    drop_malware();
}
else
{
    exit();
}
```

Is `func_x()` emulated correctly?

- Malware Found (No)
- No Malware (Yes)
Outline

1. Introduction to Malware Evasion
2. AVLeak
   1. Introduction
   2. Demo
   3. Results
   4. Discussion
3. Evasion Mitigation & Detection Research
4. Conclusion
AVLeak

• Tool for researchers to easily and quickly extract fingerprints from consumer antivirus emulators in order to evade malware detection

• “stdout for AV emulators”
AVLeak

- Tool and API for extracting fingerprints from AV emulators with advanced automated black box testing
- Use malware detections to exfiltrate specific byte values per run
- C and Python
  - Python API
- Find fingerprints in seconds not hours

<table>
<thead>
<tr>
<th>Virus Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>z</td>
</tr>
</tbody>
</table>
Multi-Byte Exfiltration

Map malware samples to byte values

AV Emulator

username="emu"
### Multi-Byte Exfiltration

Map malware samples to byte values

<table>
<thead>
<tr>
<th>Username</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Morris</td>
</tr>
<tr>
<td>B</td>
<td>Code Red</td>
</tr>
<tr>
<td>C</td>
<td>Zeus</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>a</td>
<td>Conficker</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>z</td>
<td>Brain</td>
</tr>
</tbody>
</table>

AV Emulator

```c
username="emu"
```
Multi-Byte Exfiltration

Map malware samples to byte values

GetUserName()

A  Morris
B  Code Red
C  Zeus
...

a  Conficker
...

z  Brain

Question: What is the username in the emulator?

AV Emulator

username="emu"
Multi-Byte Exfiltration

Map malware samples to byte values

Question: What is the username in the emulator?

AV Emulator

```python
username="emu"
```

for c in GetUserName():
    Drop(MalwareArray[c])

A  Morris
B  Code Red
C  Zeus
...
a  Conficker
...
z  Brain
Multi-Byte Exfiltration

Map malware samples to byte values

Question: What is the username in the emulator?

AV Emulator
username="emu"

for c in GetUserName():
    Drop(MalwareArray[c])

<table>
<thead>
<tr>
<th>A</th>
<th>Morris</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Code Red</td>
</tr>
<tr>
<td>C</td>
<td>Zeus</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>Conficker</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>Brain</td>
</tr>
</tbody>
</table>
Multi-Byte Exfiltration

Map malware samples to byte values

| A | Morris |
| B | Code Red |
| C | Zeus |
| ... |
| a | Conficker |
| ... |
| z | Brain |

Question: What is the username in the emulator?

AV Emulator
username="emu"

for c in GetUserName():
    Drop(MalwareArray[c])
Multi-Byte Exfiltration

Map malware samples to byte values

Question: What is the username in the emulator?

AV Emulator
username="emu"

for c in GetUser_name():
  Drop(MalwareArray[c])

GetUserName()

A  Morris
B  Code Red
C  Zeus
... 
a  Conficker
... 
z  Brain
Multi-Byte Exfiltration

Map malware samples to byte values

Question: What is the username in the emulator?

AV Emulator

```
username="emu"
```

for c in GetUserName():
    Drop(MalwareArray[c])

Malware Detected:
Sasser  // 'e'
Bagle   // 'm'
Blaster  // 'u'

GetUserName()

<table>
<thead>
<tr>
<th>A</th>
<th>Morris</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Code Red</td>
</tr>
<tr>
<td>C</td>
<td>Zeus</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>a</td>
<td>Conficker</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>z</td>
<td>Brain</td>
</tr>
</tbody>
</table>

April 28, 2017  Jailbreak Security Summit 33
### Multi-Byte Exfiltration

**Question:** What is the username in the emulator?

```
AV Emulator
username="emu"
```

```
for c in GetUserName():
    Drop(MalwareArray[c])
```

**Map malware samples to byte values**

<table>
<thead>
<tr>
<th>GetUserName()</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>z</td>
</tr>
</tbody>
</table>

**Malware Detected:**
- Sasser  // 'e'
- Bagle   // 'm'
- Blaster  // 'u'

**username="emu"**
AVs Tested

- Tested four commercial AVs found on VirusTotal
  - Varying levels of quality
  - Bitdefender - licensed to 20+ other AVs!
  - Installed locally in isolated VM
Outline

1. Introduction to Malware Evasion
2. AVLeak
   1. Introduction
   2. Demo
   3. Results
   4. Discussion
3. Evasion Mitigation & Detection Research
4. Conclusion
DEMO
Outline

1. Introduction to Malware Evasion
2. AVLeak
   1. Introduction
   2. Demo
   3. Results
   4. Discussion
3. Evasion Mitigation & Detection Research
4. Conclusion
Environmental Artifacts

- Hardcoded program names, computer names, product IDs, MACs, etc
- Fake processes “running” and open windows

- **argv[0]:**
  - K: `C:\{random letters}.exe`
  - AVG: `C:\...\mwsmp1.exe`
  - BD: `C:\TESTAPP.EXE`
  - VBA: `C:\SELF.EXE`

- **GetComputerName():**
  - K: `NfZtFbPfH`
  - AVG: `ELICZ`
  - BD: `tz`
  - VBA: `MAIN`
File System & Registry

• Used API to recursively dump FS and registry
  • BD: A_E_O_FANTOMA_DE_FISIER_CARE_VA_SA_ZICA_NU_EXISTA (Romanian: “this is a ghost file which will tell you [that] it doesn’t exist.bat”), TZEAPĂ_A_LA_BATMAN.EXE (“Batman’s Spike.exe” [with Romanian keyboard specific misspelling]), C:\BATMAN, NOTHING.COM
  • Kaspersky FS (random flailing on a QWERTY keyboard): C:\\Documents and Settings\Administrator\My Documents\{koio.mpg, muuo.mp3, qcse.xls, dvzrv.rar, rpplL.jpg, siso.xlsx, iykk.doc …}
    - STD_OUTxe, Dummy.exe, bat, welcome.exe, Arquivos de programas
  • Kaspersky file headers: <KL Autogenerated>
  • Fake installs of other AV products, file sharing clients, games
  • AVG Product ID: “76588–371–4839594–51979”

• Far Manager installs in Kaspersky and VBA
  - “Far Manager … for former USSR countries … as freeware...”
Other AV Products

• Kaspersky has installs for 20+ other AVs

• Bitdefender AV installs
  – Anti Virus, Bitdefender (4 different versions), Complus Applications, F-PROT95, Grisoft, Inoculate, Kaspersky Lab, McAfee, Network Associates, Norton Antivirus, Panda Software, Softwin, Symantec, TBAV, Trend Micro, and Zone Lab
Network Emulation

- Kaspersky, AVG, and Bitdefender emulate network connectivity
- Respond with success to invalid internet queries
- Downloaded executables from all three after HTTP connections
  - Reverse engineered to find more artifacts
  - Probably a way of “baiting” malware
Hardcoded Start Times

- Kaspersky: 11:01:19, July 13, 2012
  - GetSystemTimeAsFileTime: 0:0:0.00, 0/0/2000
- Bitdefender:
  - GetSystemTimeAsFileTime: 0:0:0.00
    January 1, 2008
  - GetSystemTime doesn’t work!

- NtQuerySystemTime doesn’t work!
Process Introspection

• Heap metadata, addresses, periodicity of allocations
• Contents of uninitialized memory
• Process data structures - PEB, TEB, etc
• Process size
• Data left on stack after function calls
  – Second Part To Hell’s - “Dynamic Anti-Emulation using Blackbox Analysis”
• DLLs in memory after LoadLibrary()
Fake Library Code

- Fake library code in all four AVs
- `GetProcAddress()` – dump bytes at returned pointer
- Obscure or excepting instructions are used to trigger library function emulation when picked up by CPU emulator
AVG Code

```
mov    edi, edi         ; WinAPI hot patch point
push   ebp              ; function prologue
mov    ebp, esp         ; function prologue
nop
lock   mov   ebx, 0xff(1b lib #)(2b function #)
pop    ebp              ; function epilogue
ret    (size of args)  ; stack cleanup
nop...
```
CPU Red Pills

- Save CPU state before, run instruction, save CPU state after
- Denial of service with unimplemented instructions
- Interesting area for continued research

```assembly
mov    dword_427DC0, eax
mov    dword_427DC4, ebx
mov    dword_427DC8, ecx
mov    dword_427DCC, edx
mov    word_427DD0, cs
mov    word_427DD2, ds
mov    word_427DD4, es
mov    word_427DD6, fs
mov    word_427DD8, gs
mov    word_427DDA, ss
mov    dword_427DDC, esi
mov    dword_427DE0, edi
mov    dword_427DE4, esp
mov    dword_427DE8, ebp
push   eax
pushf
pop    eax
mov    dword_427DF0, eax
pop    eax
nop
mov    dword_427D80, eax
mov    dword_427D84, ebx
mov    dword_427D88, ecx
mov    dword_427D8C, edx
mov    word_427D90, cs
mov    word_427D92, ds
mov    word_427D94, es
mov    word_427D96, fs
mov    word_427D98, gs
mov    word_427D9A, ss
mov    dword_427D9C, esi
mov    dword_427DA0, edi
mov    dword_427DA4, esp
mov    dword_427DA8, ebp
```
Outline

1. Introduction to Malware Evasion
2. AVLeak
   1. Introduction
   2. Demo
   3. Results
   4. Discussion
3. Evasion Mitigation & Detection Research
4. Conclusion
Common Themes

• Extremely simple detection methods are sufficient for evasion
• Hardcoded environmental artifacts clearly from programmers
• Attempts to “bait” malware
• Lack of heuristic malware classification for emulation-detection behavior
  – Kaspersky does a little bit with its detection of in-memory DLL scanning
Malware Discovery

- Advanced malware authors are already using these artifacts

58a5fa7f2928a7eb24d73b3059d2221e2acd83a - Analysis ...
https://totalhash.cymru.com/analysis/?... ▼
Jan 24, 2014 - BAT CCFMceg CCF49Ch4 CCFF9 CCFMceg \cd^@Z ceeddbbaa`Y ... \A_E_O_FANTOMA_DE_FISIER_CARE也许 Vapor_NU_EXISTS.BAT ...

Analysis | #totalhash - Team Cymru
https://totalhash.cymru.com/analysis/?... ▼
Jan 2, 2014 - File type, PE32 executable for MS Windows (GUI) Intel 80386 32-bit. Language, 040804b0. Section .text md5: ...

4166c77a7f7891ce8756b9784c46a2da2d511dd - Analysis ...
https://totalhash.cymru.com/analysis/?... ▼
Jan 24, 2014 - File type, PE32 executable for MS Windows (GUI) Intel 80386 32-bit. Language, 0409046b. Section .text md5: ...

e094d944954303f06d769b899a46e650cc347dc4f - Analysis ...
https://totalhash.cymru.com/analysis/?... ▼
Jan 1, 2014 - ... BSMSx:TR B:Q+= "bTs p= bY/KB+G -.C8nQA c.ae) C:\A_E_O_FANTOMA_DE_FISIER_CARE也许 Vapor_NU_EXISTS.BAT California#0!

6 results (0.33 seconds)
Did you mean: "<kl auto generated>"

Analysis - Malwr - Malware Analysis by Cuckoo Sandbox
https://malwr.com/.../ZmM0ZTg0Zg5OTk0NGM1ODYmFkMtQ2ZjM2...
Apr 24, 2014 - EXE: wswhacker.dllMZ. This program cannot be run in DOS mode. <KL Autogenerated>. MSIMG32.dll. AlphaBlend. DllInitialize. GradientFill.

0b621aa5c4e63b3579eea52f0422bb9f - Malwr - Malware ...
https://malwr.com/.../ODc2ZDZiZjkwYWU2NGYzZjkkODc4OTczNWE3...
7 days ago - Error: Analysis failed: The package "modules.packages.exe" start function raised an error: Unable to execute the initial process, analysis ...

39ef896e2e1a9cd27d96d16d4b55dada7d21112f - Analysis ...
https://totalhash.cymru.com/analysis/?... ▼
Jan 22, 2015 - ... IsWow64Process KERNEL32.dll <KL Autogenerated> _closeLoadLibraryA LockResource lstrcmp lstrcypW lstrcpyW LzStart MoveFileExA ...

Malware Analysis Database - totalhash
https://totalhash.com/analysis/?...
Aug 14, 2014 - ... DLL kkS_1Y(W <KL Autogenerated> #~nel %0ra# IA/78=V LCMapStringA _lcreate I g"YY: S-R LoadLibraryA LoadLibraryExA LoadResource ...

Analysis | #totalhash
totalhash.com/analysis/?361693130dcaab810c8abe2d550f147b796745d
Nov 4, 2014 - Creates File, C:\Documents and Settings\Administrator\Local Settings\Temp\2445_appcompat.txt. Creates File, PIPE\lsarpc. Creates Process ...
Malware Insights - EvilBunny

- EvilBunny (Animal Farm APT) was evading Bitdefender in 2011
- “TESTAPP” = process name in Bitdefender - Kaspersky?
- Discovered by Marion Marschalek
Evasion

- AVLeak-derived fingerprints make it extremely easy to evade detection
- 100% evasion rate in testing

```c
#include "avleak.h"

int main(int argc, char * argv[]){
    char target[30] = {0};
    int len=30;
    GetComputerName(target,(LPDWORD)&len);
    if(strcmp(target, "tz") == 0){
        exit(1);
    } else{
        printf("Real System, dropping EICAR\n");
        EICAR();
    }
}
```
Emulator Exploitation

- Beyond evasion: break out of emulators?
- Emulators provide browser-like primitives for exploit development
- Exploitation = code exec + privesc + AV context execution

Project Zero

News and updates from the Project Zero team

Tuesday, June 23, 2015

Analysis and Exploitation of an ESET Vulnerability

Do we understand the risk vs. benefit trade-offs of security software?
Tavis Ormandy, June 2015

Introduction

Many antivirus products include emulation capabilities that are intended to allow unpackers to run for a few cycles before signatures are applied. ESET NOD32 uses a minfilter or kext to intercept all disk I/O, which is analyzed and then emulated if executable code is detected.

Attackers can cause I/O via Web Browsers, Email, IM, file sharing, network storage, USB, or hundreds of other vectors. Whenever a message, file, image or other data is received, it’s likely some untrusted data passes through the disk. Because it’s so easy for attackers to trigger emulation of untrusted code, it’s critically important that the emulator is robust and isolated.

Unfortunately, analysis of ESET emulation reveals that is not the case and it can be trivially compromised. This report discusses the development of a remote root exploit for an ESET vulnerability and demonstrates how attackers could compromise ESET users. This is not a theoretical risk, recent evidence suggests a growing interest in anti-virus products from advanced attackers.
Outline

1. Introduction to Malware Evasion
2. AVLeak
3. Evasion Mitigation & Detection Research
   1. Evasion detection
   2. Evasion mitigation
      1. Detect then mitigate
      2. Transparent analysis
4. Conclusion
Evasion Detection

• 2008: look for *known* evasion techniques each CPU instruction
  – Lau et al.’s DSD-Tracer
• 2009: divergence in CPU instruction traces
  – Kang et al.
• 2010: divergence in system call traces
  – Balzarotti et al.
Evasion Detection Continued

• 2011: Jaccard distance calculations over normalized traces of persistent changes to system state  
  – Lindorfer et al.’s Disarm

• 2014: Text comparison and computer vision inspired hierarchical comparison of “transient behavioral profiles”  
  – Kirat et al.’s BareCloud

• 2015: Bioinformatics-inspired sequence alignment  
  – Kirat & Vigna’s Malgene
Trends in Evasion Detection

• Analyses are both increasingly abstracted and advanced
  – Abstraction: instruction traces to syscall traces to persistent changes to system state to “transient behavioral profiles”
  – Advanced: trace divergence to Jaccard distance to text/vision hierarchical comparison to bioinformatics algorithms
Outline

1. Introduction to Malware Evasion
2. AVLeak
3. Evasion Mitigation & Detection Research
   1. Evasion detection
   2. Evasion mitigation
      1. Detect then mitigate
      2. Transparent analysis
4. Conclusion
Evasion Mitigation: Detect then mitigate

• 2006: QEMU-like Code rewriting
  – Vasudevan et al.’s Cobra
• 2007: Rootkit-like code hooking
  – Willems et al.’s CWSandbox
• 2009: Dynamic state modification
  – Kang et al.
Outline

1. Introduction to Malware Evasion
2. AVLeak
3. Evasion Mitigation & Detection Research
   1. Evasion *detection*
   2. Evasion *mitigation*
      1. Detect then mitigate
      2. Transparent analysis
4. Conclusion
Evasion Mitigation: Hypervisors

- 2008: Xen with system call monitoring
  - Dinaburg et al.’s Ether
- 2009: custom hypervisor
  - Nguyen et al.’s MAVMM
- 2014: Xen with breakpoint injection
  - Lengyel et al.’s DRAKVUF
Evasion Mitigation: Bare-Metal

- **2011:** In-system kernel driver
  - Kirat et al.’s BareBox
- **2012 / 2014:** Network traffic & disk activity capture with forensic tools
  - Royal’s NVMTrace
  - Kirat et al.’s BareCloud
- **2016:** RAM and disk activity capture through hardware instrumentation
  - Spensky et al.’s LO-PHI
Evasion Mitigation: In-SMM Analysis

- 2013: Passive observation from SMM
  - Zhang et al.'s Spectre
- 2015: Debugging from SMM
  - Zhang et al.'s MalT
- 2016: Memory capture from SMM and PCI-bus capture
  - Leach et al.'s Hops
Trends in Evasion Mitigation

- Detect then mitigate: endless cat-and-mouse game; intractable
- Hypervisors: more transparent, but still relatively easily detected
- Bare metal: transparency comes at the cost of visibility; more research to be done
- SMM: potentially best of both worlds - transparency and visibility
Outline

1. Introduction to Malware Evasion
2. AVLeak
3. Evasion Mitigation & Detection Research
4. Conclusion
Parting Thoughts
Parting Thoughts

• Defense: How can we exploit evasive behaviors to detect malware?
Parting Thoughts

- **Defense**: How can we exploit evasive behaviors to detect malware?
- **Offense**: How can malware find resilient difficult-to-mitigate fingerprints? How viable is emulator exploitation?
Parting Thoughts

• Defense: How can we exploit evasive behaviors to detect malware?
• Offense: How can malware find resilient difficult-to-mitigate fingerprints? How viable is emulator exploitation?
• Philosophical: What are the fundamental limitations of transparent dynamic analysis?
Thank You

Jailbreak Brewing & Ryan Speers for inviting me!

• RPI Research Team:
  – Jeremy Blackthorne
  – Andrew Fasano
  – Patrick Biernat
  – Dr. Bülent Yener
  – Dr. Greg Hughes

• Help & Inspiration:
  – Marion Marshalek
  – Rolf Rolles
  – Alex Ionescu
  – Bruce Dang
  – Dr. Sergey Bratus
BACKUP SLIDES
& ADDITIONAL CONTENT
Additional Slides

- Bibliography & Further Reading
- Additional Attacks
- Function Emulation
- Malware Discovery
- Software Engineering
AV Leak:
Fingerprinting Antivirus Emulators Through Black-Box Testing

Jeremy Blackthorne  Alexei Bulazel  Andrew Fasano  Patrick Biernat  Bülent Yener
Rensselaer Polytechnic Institute

For a more thorough exposition of AV Leak, check out our WOOT ’16 paper: Blackthorne, Bulazel, Fasano, Biernat, and Yener - “AV Leak: Fingerprinting Antivirus Emulators Through Black Box Testing”

https://www.usenix.org/conference/woot16/workshop-program
Specific Version Info

- Kaspersky Antivirus 15.0.2.480
- Emsisoft Commandline Scanner 10.0.0.5366 (specific Bitdefender engine version unclear - See http://www.av-comparatives.org/av-vendors/)
- AVG 2015.0.6173
- VBA Windows/CL 3.12.26.4
OS API Inconsistency

- Functions fail, return failure, cause analysis to stop, etc...
  - Don’t often return unique multibyte fingerprints, doesn’t vastly benefit from AVLeak over prior testing schemes
- Lack of permissions enforcement for FS
- Clipboard manipulation
- Window / GUI interaction not emulated
- AVG FormatMessage()
  - “\text{MID}[\text{dwMessageId in hex}]”
Timing Fingerprinting

- GetTickCount, GetSystemTime, GetSystemTimeAsFileTime, NtQuerySystemTime, QueryPerformanceCounter, rdtsc, rdtscp
- No need for sophisticated timing attacks
- Time is another environmental artifact
  – Static unchanging start values
- Kaspersky & AVG: attempt to be realistic
- Bitdefender & VBA: totally dysfunctional
Bitdefender Code

```assembly
push 0x0
push (three byte #)
call 0xffff(two byte #)
add esp, 0x8
jmp 0xffff(two byte #)
int3... ; Int3 instructions between functions is unique
         ; in the Windows system DLLs I examined, nops
         ; were present between functions.

push 0x0060(two byte #)
push 0xBF80001
ret ; ret to 0xBF80001
```
VBA Code

```plaintext
mov   edi, edi   ; WinAPI hot patch point
nop
nop
jecxz $+0        ; jmp ecx==0 to next instr
jmp $+0        ; jmp to next instr
mov DWORD PTR 0xFFF1[2 byte export #], 0x406
ret (size of args) ; stack cleanup
hlt... ; Hlt instructions between functions
```
Kaspersky Code

• Kaspersky would generate random bytes per run after first few bytes of each function. Looking at code would frequently result in heuristic malware classifications.

```
mov edi, edi
push (address of function)
(random bytes generated per run)
nop...
```
Thai Malware

- Googled Windows product ID found in AVG’s registry
  - 76588-371-4839594-51979
- Found AVG-evasive malware hosted on website for Thai school - likely hacked
- Contained hundreds of AVG fingerprints
- Two uploads to VT before us since 2012

AVG function emulation
AVLeak Architecture

• Test cases written in C
• Write once, run anywhere against any AV
• Python API to build scriptable test cases
  - Dynamic testing scripts
  - Integrate with other applications

• Easy to integrate new AVs
  - Automated with scripts
Design

- AVLeak is designed to be easy to use and portable
  - Anyone who can write C code can write tests
  - Tests work against any AV
- AVLeak Python code automates the process of compiling binaries, scanning them, and reconstructing results
  - AVs can detect varying numbers of dropped malware samples per run, so it is almost always necessary to compile multiple test binaries
Example Command Line Invocation

```
$ python run_test.py k test/environment/argv0.c --printmax 100 -n 7 --nobase
KASPERSKY OUTPUT:
C:\nxav.exe
C:\ifitgx.exe
C:\hyzglgz.exe
C:\pqxmt.exe
C:\adkxkz.exe
C:\psrepsx.exe
C:\cbqzqtch.exe
```

$ python run_test.py kaev test/environment/GetComputerName.c --printmax 20
```
BASE OUTPUT:
WIN-PN9R6J7FCOD
KASPERSKY OUTPUT:
NfZtFbPfH
AVG OUTPUT:
ELICZ
EMSISOFT OUTPUT:
tz
VBA OUTPUT:
MAIN
```

AV to test  path to code  number of bytes to print  number of times to run test  don’t run test on host
AVLeak Flow

Probe code in C

```c
#include "avleak.h"

int main(
    int argc,
    char * argv[])
{
    leak(argv[0]);
}
```
AVLeak Flow

Probe code in C

```
#include "avleak.h"

int main(
    int argc,
    char * argv[])
{
    leak(argv[0]);
}
```

AVLeak (Python)
AVLeak Flow

Probe code in C

```c
#include "avleak.h"

int main(
    int argc,
    char * argv[])
{
    leak(argv[0]);
}
```

Compiler

AVLeak (Python)
AVLeak Flow

Probe code in C

```c
#include "avleak.h"

int main(
    int argc,
    char * argv[])
{
    leak(argv[0]);
}
```

AVLeak (Python)

Compiler

Probe Binary
AVLeak Flow

Probe code in C

```c
#include "avleak.h"

int main(
    int argc,
    char * argv[])
{
    leak(argv[0]);
}
```

AVLeak (Python)

Compiler

Probe Binary

AV
AVLeak Flow

Probes code in C

```c
#include "avleak.h"
int main(
    int argc,
    char * argv[])
{
    leak(argv[0]);
}
```

AVLeak

Compiler

AV

Malware Found!

Probe Binary

Malware Found!
Example Test Case

#include "avleak.h"

int main(int argc, char * argv[])
{
    char target[30] = {0};
    DWORD len=30;
    GetUserUserName(target,&len);
    leak(target);  // easy to use API like
                   // printing to stdout
}

leak(target);
API

- An easy to use Python API provides programmatic access to AVLeak
- This can be used to create dynamic testing routines, for example: recursive exploration of file systems and registries, programmatic dumping of in memory code, red pill testing
- Can also be integrated with other libraries, such as Capstone for in-line disassembly
from AVLeak import *
http_flags = ["HTTP_QUERY_ACCEPT",
            "HTTP_QUERY_ACCEPT_CHARSET",
            "HTTP_QUERY_ACCEPT_ENCODING",
            ... ]

def test_http(av):
    for flag in http_flags:
        result = av.leak(
            testfile = "HttpQueryInfo_flags.c",
            string = flag,
            printmax = 20)
        print flag, "-", result
int main() {
    HANDLE hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    char out[30] = {0};
    int count = 0;
    if(hSnapshot != INVALID_HANDLE_VALUE) {
        PROCESSENTRY32 pe32;
        pe32.dwSize = sizeof(PROCESSENTRY32);
        if(Process32First(hSnapshot, &pe32)) {
            do {
                sprintf(out, "%d %s", pe32.th32ProcessID, pe32.szExeFile);
                #ifdef AV // inside AV, N_AV incremented for each process
                  if (count == N_AV) { leak(out); exit(0); }
                #else // real system
                  leak(out);
                #endif
                count++;
            } while(Process32Next(hSnapshot, &pe32));
            DONE(); // we have gone through all processes, don’t scan more
        }
        CloseHandle(hSnapshot);
    }
}