AVLeak: Fingerprinting Antivirus Emulators For Advanced Malware Evasion

Alexei Bulazel
@av_leak
Outline

1. Introduction
2. Background
3. AVLeak
4. Results & Demo
5. Conclusions

More content than I can fit in a 25 minute briefing available in extra slides published online. USENIX WOOT ’16 publication also out next week.
Problem

- Automated dynamic analysis (aka “sandbox analysis”) is essential against 1,000,000+ new pieces of malware per day

- Malware can behave benignly to avoid detection if it can determine that it’s being analyzed
  - Over 80% of malware exhibited evasive behavior in 2nd half of 2015

- Extensive prior work on detecting traditional emulators and virtual machines: VMWare, QEMU, Xen, VirtualBox, Bochs, etc

- Little prior work on consumer AV emulators
Motivation

• Consumer AV emulators are intuitively easy to evade
• But... no one has demonstrated a good approach

• Existing methods to extract fingerprints from emulators are inefficient:
  – Reverse engineering
    • Too hard
  – Black-box dynamic analysis
    • Too slow

• Our goal: Automate and accelerate fingerprint discovery
AVLeak

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

1. Introduction
2. Background
3. AVLeak
4. Results & Demo
5. Conclusions
Fingerprinting Analysis Systems

- Environmental artifacts
  - Hardcoded strings for username/computer name/environment variables, file system, registry entries, processes
- OS API inconsistency
  - Functions that fail, return hardcoded values, generally don’t behave correctly
- Network emulation
  - Inconsistencies with real network behavior, hardcoded responses to network traffic
- Timing
  - Timing skews, dilation, inconsistencies across observations
- Process Introspection
  - Internal process traits - uninitialized memory, data left on stack or in registers after function calls, PEB/TEB, DLLs in memory
- CPU “Red Pills”
  - Instructions which behave differently on an emulated CPU
## Traditional Malware Sandbox

Many attack points, can look at open source implementations

<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>User Process</td>
<td>User Process</td>
</tr>
<tr>
<td>EMU</td>
<td>Virtualized Hardware</td>
<td>User Process</td>
<td>User Process</td>
</tr>
<tr>
<td>Cuckoo</td>
<td>Malware Sandbox</td>
<td>User Process</td>
<td>User Process</td>
</tr>
<tr>
<td>Windows</td>
<td>Operating System or Hypervisor</td>
<td>User Process</td>
<td>User Process</td>
</tr>
</tbody>
</table>

Hardware
Consumer AV Emulator

- Malware
- x86 Emulator
- Environment
- Usermode WinAPI Emulator
- User Process
- User Process
- Antivirus Emulator
- Operating System
- Hardware
Consumer AV Emulator

Totally opaque, only output is analysis report

Analysis report:
Dropped: Trojan.Infector.BAT.ABC123
Dropped: APT1337.Backdoor.2
Dropped: CryptoLocker.Downloader.K
Prior Approaches: Mutation

- Mutate a binary with packers and obfuscators to avoid detection
- Requires no knowledge of AV internals
- Pentester-focused, often pack Metasploit payloads: Veil-Evasion, SideStep, peCloak, etc...
Reversing AV Emulators

- Time consuming
- Expensive tools
- Expert knowledge – RE, AV, x86, Windows internals, malware behavior, anti-analysis
- Limited Lifespan - frequent updates
Prior Approaches: Black Box Testing

- Find emulator fingerprints for targeted evasion
- Prior approaches - single true/false query per run
  - Arne Swinnen & Alaeddine Mesbahi - One Packer To Rule Them All (Black Hat ’14)
  - Kyle Adams - Evading Code Emulation (BSidesLV ’14)
  - Daniel Sauder - Why Antivirus Software Fails (DeepSec ’14)
  - Emeric Nasi - Bypass Antivirus Dynamic Analysis (white paper ‘14)
Prior Approaches: Black Box Testing

Question: Does the emulator emulate function_x() correctly?

AV Emulator
Question: Does the emulator emulate function_x() correctly?

```
if function_x() != EXPECTED:
    DropMalware()
else:
    Exit()
```

<table>
<thead>
<tr>
<th>Malware</th>
<th>TRUE</th>
<th>Malware</th>
<th>FALSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Malware</td>
<td>FALSE</td>
<td>No Malware</td>
<td>TRUE</td>
</tr>
</tbody>
</table>
Prior Approaches: Black Box Testing

Question: Does the emulator emulate function_x() correctly?

<table>
<thead>
<tr>
<th>Malware</th>
<th>TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Malware</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

AV Emulator

```python
if function_x() != EXPECTED:
    DropMalware()
else:
    Exit()
```
Prior Approaches: Black Box Testing

Question: Does the emulator emulate function_x() correctly?

<table>
<thead>
<tr>
<th>Malware</th>
<th>TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Malware</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

AV Emulator

```
if function_x() != EXPECTED:
    DropMalware()
else:
    Exit()
```

If malware detected, function_x() not emulated correctly.
If no malware detected, function_x() emulated correctly.
Outline

1. Introduction
2. Background
3. AV Leak
4. Results & Demo
5. Conclusions
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

Virus Database

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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>
AVLeak’s Innovation

Map malware samples to byte values

AV Emulator

username="emu"
AVLeak’s Innovation

Map malware samples to byte values

AV Emulator
username="emu"

getUserName()

A  Morris
B  Code Red
C  Zeus
...

a  Conficker
...
z  Brain
# AVLeak’s Innovation

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

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

| A | Morris  |
| B | Code Red |
| C | Zeus    |
|   |         |
| a | Conficker |
|   |         |
| z | Brain   |
AVLeak’s Innovation

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])
AVLeak’s Innovation

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])
```
AVLeak’s Innovation

Map malware samples to byte values

Table:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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>

Question: What is the username in the emulator?

AV Emulator

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

for c in GetUser\Name():
    Drop(MalwareArray[c])
AV Leak's Innovation

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])

GetUserName()

A Morris
B Code Red
C Zeus
...
a Conficker
...
z Brain
AVLeak’s Innovation

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()

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

Map malware samples to byte values.
AVLeak’s Innovation

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"

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

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
2. Background
3. AVLeak
4. Results & Demo
5. Conclusions
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:\...\mwsmpl.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”), TZEAPA_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

- 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_427DDE, edi
mov     dword_427DE0, ebp
push    eax
push    ebx
pop     eax
mov     dword_427DE8, ebp
mov     dword_427DFO, eax
pop     eax
nop
mov     dword_427D10, eax
mov     dword_427D14, ebx
mov     dword_427D18, ecx
mov     dword_427D1C, edx
mov     word_427D20, cs
mov     word_427D22, ds
mov     word_427D24, es
mov     word_427D26, fs
mov     word_427D28, gs
mov     word_427D2A, ss
mov     dword_427D2C, esi
mov     dword_427D2E, edi
mov     dword_427D30, ebp
mov     dword_427D38, ebp
```
Outline

1. Introduction
2. Background
3. AVLeak
4. Results & Demo
5. Conclusions
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
**Low Budget Malware Discovery**

- Advanced malware authors are already using these artifacts

---

6 results (0.33 seconds)

**Did you mean: "<kl auto generated>"**

**Analysis - Malwr - Malware Analysis by Cuckoo Sandbox**
https://malwr.com/.../ZmM0ZTg0Zjg5OTk0NGM1OGI0YmFkMTQ2ZjM2...

**Analysis - Malwr - Malware Analysis by Cuckoo Sandbox**
https://malwr.com/.../ODc2ZDZiZjkYWU2NGYzZjg0ZDc4OTczNWE3...
7 days ago - Error: Analysis failed: The package "modules.packages.exe" start function raised an error: Unable to execute the initial process, analysis ...

**Analysis - Malwr - Malware Analysis by Cuckoo Sandbox**
https://malwr.com/.../IsW0w64Process KERNEL32.dll <KL Autogenerated> _lcose LoadLibraryA LockResource lstrcmp lstrcmp lstrcpy w LZStart MoveFileExA ...

**Malware Analysis Database - totalhash**
https://totalhash.com/analysis/?...
Aug 14, 2014 - DLL kKS_1Y(W <KL Autogenerated> #~nel %10r#j IA|78=v LComMapStringA _lcreat ! g"YY:S+R LoadLibraryA LoadLibraryExA LoadResource ...

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

---

Black Hat 2016
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();
    }
}```
Future Work

• More emulators, more tests
• Use AVLeak for vulnerability research against emulators (breakout exploits)
  – See Tavis Ormandy and Joxean Koret’s work

---

**Project Zero**

News and updates from the Project Zero team at Google

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 minifilter 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.
Conclusion

- Pushed the state of the art in emulator fingerprinting
- Presented a survey of emulator fingerprints across six categories
- Demonstrated real world examples of malware exploiting these fingerprints
- Discussed directions in future research
Thank You

• 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
Questions?

Blackthorne, Bulazel, Fasano, Biernat, and Yener - “AVLeak: Fingerprinting Antivirus Emulators Through Black Box Testing” published next week at 10th USENIX Workshop on Offensive Technologies (WOOT ’16)

Kaspersky Lab - Packin’ The K

More content than can fit in a 25 minute briefing available in extra slides published online.
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 AVLeak, check out our WOOT ’16 paper: Blackthorne, Bulazel, Fasano, Biernat, and Yener - “AVLeak: Fingerprinting Antivirus Emulators Through Black Box Testing”

https://www.usenix.org/conference/woot16/workshop-program
Joxean Koret & Elias Bachaalany
The Antivirus Hacker’s Handbook

• This book is awesome, everything you could ever want to know about AV
• Wish I had a copy while doing this research, would have answered a lot of questions
Prior Black-Box Testing Presentations

- Arne Swinnen & Alaeddine Mesbahi, One Packer To Rule Them All, Black Hat 2014
  - https://www.youtube.com/watch?v=gtLMXxZErWE
  - https://www.youtube.com/watch?v=tkOtBkvS9xY
- Daniel Sauder, Why Antivirus Software Fails, DeepSec 2014
  - https://www.youtube.com/watch?v=oe-KPageZrl
- Emeric Nasi, Bypass Antivirus Dynamic Analysis, self-published white paper
Additional Presentations

• Christopher Kruegel - Full System Emulation: Achieving Successful Automated Dynamic Analysis of Evasive Malware (Black Hat 2014) [malware evasion]

• Georg Wicherski - Dirtbox, an x86/Window Emulator (REcon 2010 / Black Hat 2010) [emulating evasive malware]

• Ke Sun, Xiaoning Li, Ya Ou - Break Out of The Truman Show: Active Detection and Escape of Dynamic Binary Instrumentation (Black Hat Asia 2016) [interesting ideas for process introspection style attacks]
Blog Posts

• Rolf Rolles - Memory Lane: Hacking Renovo [exploiting packed code detection to leak data from an academic system]
• Marion Marschalek - EvilBunny: Malware Instrumented By Lua [EvilBunny writeup]
• Eugene Kaspersky - Emulation: A Headache To Develop and Emulate To Exterminate [rare public statements about emulation from Kaspersky Lab CEO and founder]
• Tavis Ormandy - Analysis and Exploitation of an ESET Vulnerability [emulator breakout exploits]
• Second Part To Hell - Dynamic Anti-Emulation using Blackbox Analysis [emulator detection through undocumented register states, no prior access to the emulator necessary]
Papers

• Jeremy Blackthorne & Dr. Bülent Yener - Reverse Engineering Anti-Virus Emulators through Black-box Analysis [technical report on an AVLeak prototype]
• Katsunari Yoshioka, et al - Your Sandbox is Blinded [novel attack on network-connected sandboxes]
• Peter Ferrie - Attacks on Virtual Machine Emulators / More Attacks on Virtual Machine Emulators [catalog of attacks on popular virtualization systems]
• Garfinkel et al - Compatibility is Not Transparency: VMM Detection Myths and Realities [why it’s “fundamentally infeasible” to build an undetectable virtualization system]
• Kevin Hamlen et al - Exploiting An Antivirus Interface; Filiol et al - Evaluation methodology and theoretical model for antiviral behavioural detection strategies; Filiol - Malware pattern scanning schemes secure against black-box analysis; Borello et al - From the design of a generic metamorphic engine to a black-box classification of antivirus detection techniques [black box testing to discover static signatures]
Rolf Rolles answered a reverse engineering Stack Exchange question one my collaborators on this research made early on in our efforts. Following through with Rolf’s advice, we tried REing some emulators, it was extremely challenging as they are immensely complex, and so large that IDA lags significantly and sometimes even crashes during analysis.

http://reverseengineering.stackexchange.com/questions/2805/detecting-an-emulator-using-the-windows-api

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-discernible 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.
Advanced Attacks on High End System

- Traditional malware sandboxes (ie: Anubis, Cuckoo, and various high-end enterprise network protection systems) generally run a full Windows OS install on top of virtualized hardware (such as CPU-level hypervisors, VMware / VirtualBox, or QEMU). Malware analysis may be carried out through in-box instrumentation in user or kernel mode, or through out-of-box system introspection.

- These systems generally return rich analysis reports detailing specific malware actions (ie: files created, mutexes opened, registry entries opened). Attackers may exploit these reports to exfiltrate system fingerprints by using observed artifacts in their actions (ie: creating a file named after an observed artifact, thereby leaking the artifact through the analysis report).

- Further, as Yoshioka et al demonstrate in “Your Sandbox is Blinded: Impact of Decoy Injection to Public Malware Analysis Systems” (https://www.jstage.jst.go.jp/article/ipsjjip/19/0/19_0_153/_pdf), emulator artifacts may also be leaked through web traffic in network connected analysis systems. Despite this vulnerability, these systems are often network connected, as malware may not show its full behavior if run in an isolated system.
Advanced Attacks on High End System

Sandbox fingerprints:
Username: SndbxUsr
Usermode process: debug_process.exe
Driver: analysis_driver.sys
Advanced Attacks on High End System


Sandbox fingerprints:
Username: SndbxUsr
Usermode process: debug_process.exe
Driver: analysis_driver.sys

Analysis report:
Created file: SndbxUsr.txt
Opened mutex: debug_process.exe
Edited registry: analysis_driver.sys
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
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...
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()
  – “MID[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

```
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

```vba
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
```

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:\nixav.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

```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]);
}
```

AVLeak (Python)

Compiler
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

```
#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]);
}
Example Test Case

```c
#include "avleak.h"

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

```python
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);
    }
}