AVLeak: Fingerprinting Antivirus Emulators Through Black-Box Testing

Jeremy Blackthorne, Alexei Bulazel, Andrew Fasano, Patrick Biernat, Bülent Yener

Alexei Bulazel and Andrew Fasano

@av_leak
Introduction

• Research group from Rensselaer Polytechnic Institute (RPI) under Dr. Bülent Yener

• Jeremy Blackthorne - PhD candidate
• Alexei Bulazel - recent MS graduate
• Andrew Fasano - undergraduate researcher (graduated)
• Patrick Biernat - undergraduate researcher
• Dr. Bülent Yener - advisor
Outline

1. Introduction
2. Problem & Motivation
3. Background & Prior Work
4. AVLeak
5. Results & Demo
6. Conclusions
Problem

• Modern AV software uses dynamic (“sandbox”) analysis to scan the 1,000,000+ new malware binaries created every day

• Consumer AV emulators are conceptually easy to evade

• If emulation can be detected, malware can behave benignly to avoid detection

• There is not an efficient method to “fingerprint” consumer AV emulators
Motivation

• 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
Outline

1. Introduction
2. Problem & Motivation
3. Background & Prior Work
4. AVLeak
5. Results & Demo
6. Conclusions
Background

• Packers can generate millions of unique binaries that behave identically while evading static signatures

• Dynamic (sandbox) analysis allows AV engines to identify known signatures or heuristically classify previously unknown malware

• Extensive prior research on detecting high-end emulators and VMs - QEMU, VMWare, Xen, Bochs, etc

• Little prior work on consumer AV emulators
Classes of Consumer AV Fingerprints

- Environmental artifacts
  - Hardcoded username, registry entries, processes names

- OS API inconsistency
  - Failures and incorrect return values

- Network emulation
  - Hardcoded responses and inconsistencies
Classes of Consumer AV Fingerprints

- Timing
  - Timing skews and dilation

- Process Introspection
  - Internal inconsistencies - PEB, heap allocations, etc

- CPU “Red Pills”
  - Instructions which behave differently on an emulated CPU
Reversing AV Emulators

- Time consuming
- Expensive tools
- Expert knowledge
  - RE, AV, x86, Windows internals, malware behavior, anti-analysis
- Limited Lifespan
  - frequent updates
### Traditional Malware Sandbox / Emulator Architecture

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

Many introspection points for fingerprint extraction
Consumer AV Emulator

- x86 Emulator
- Environment
- Usermode WinAPI Emulator
- Antivirus Emulator
- Operating System
- Hardware

Malware

User Process

User Process
Consumer AV Emulator

Single introspection point: analysis report for given input binary

Analysis report:
Dropped: Trojan.Infector.BAT.ABC123
Dropped: APT1337.Backdoor.2
Dropped: CryptoLocker.Downloader.K
Prior Approach: Black Box Testing

- Extract a single bit of data 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

True or False Question: Does the emulator emulate function\_x() correctly?

AV Emulator
Prior Approaches: Black Box Testing

True or False Question: Does the emulator emulate function_x() correctly?

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

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

True or False 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>

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

AV Emulator
Prior Approaches: Black Box Testing

True or False 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()
```

Malware Detected (function\_x() not emulated correctly)

No Malware Detected (function\_x() emulated correctly)

Exit()
Evasive Malware: Case Study

- EvilBunny (Animal Farm APT) was using fingerprints to evade Bitdefender in 2011
- Bitdefender calls processes under analysis “TESTAPP”

EvilBunny doesn’t run when called “TESTAPP”
Outline

1. Introduction
2. Problem & Motivation
3. Background & Prior Work
4. AVLeak
5. Results & Demo
6. Conclusions
Introducing AVLeak

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

• Design: Test cases in C, automated with Python, Python API

• Goals:
  – Fingerprint the AV itself
  – Ease of use
  – Abstract AV interaction from the programmer
  – Scriptable API
  – Find fingerprints in seconds not hours
Introducing AVLeak

• Novel approach to leak bytes values from inside AV emulators

• Map malware names to byte values

• Use malware detections to exfiltrate specific byte values per run

<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>
AVLeak’s Innovation

Question: What is the username in the emulator?

AV Emulator
username="emu"
AVLeak’s Innovation

Question: What is the username in the emulator?

AV Emulator

```
username="emu"
```

<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>a</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>z</td>
</tr>
</tbody>
</table>
AVLeak’s Innovation

Question: What is the username in the emulator?

<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>

AV Emulator

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

```python
for c in GetUserName():
    Drop(MalwareArray[c])
```
AVLeak’s Innovation

Question: What is the username in the emulator?

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

<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>
AVLeak’s Innovation

Question: What is the username in the emulator?

GetUserName()

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

AV Emulator

\texttt{username="emu"}

\begin{verbatim}
for c in GetUserName():
    Drop(MalwareArray[c])
\end{verbatim}
AV Leak’s Innovation

Question: What is the username in the emulator?

AV Emulator
username="emu"

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

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

Question: What is the username in the emulator?

GetUserName()

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

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

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

AV Emulator
username="emu"
AVLeak’s Innovation

Question: What is the username in the emulator?

GetUserName()

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

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
  - Identified by uploading EICAR droppers
- Bitdefender emulator licensed to 20+ other AVs
Outline

1. Introduction
2. Problem & Motivation
3. Background & Prior Work
4. AVLeak
5. Results & Demo
6. Conclusions
Classes of Consumer AV Fingerprints

- 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
Environmental Artifacts

- 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

- 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,...
  - 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...”
Hardcoded Start Times

- **Kaspersky:** 11:01:19, July 13, 2012
- **AVG:** 1:40:41.16, May 23, 2011
- **VBA:** 1:31:12.123, November 3, 2014
  - `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!
Fake Library Code

- Fake library code in all four AVs
- GetProcAddress – dump bytes at pointer
- Obscure instructions are used to trigger library function emulation

AVG:
```assembly
mov edi, edi
push ebp
mov ebp, esp
nop
lock mov ebx,
      0xff(1b lib #)(2b func #)
pop ebp ; epilogue
ret (size of args)
nop...
```
Outline

1. Introduction
2. Problem & Motivation
3. Background & Prior Work
4. AVLeak
5. Results & Demo
6. Conclusions
Common Themes

- Checking for simple fingerprints enables malware to evade detection
- Hardcoded environmental artifacts are clearly left by programmers as jokes, or as “bait” for malware
- AVs don’t do heuristic malware classification based on emulation-detection behavior
Low Budget Malware Discovery

- Advanced malware authors are already using these artifacts

Analysis | #totalhash - Team Cymru
https://totalhash.cymru.com/analysis/?...
Jan 24, 2014 - File type, PE32 executable for MS Windows (GUI) Intel 80386 32-bit. Language, 040904b0. Section .text md5: ...

4166c77a7f7891ce8756fb9784c46a2da2d511dd - Analysis ...
https://totalhash.cymru.com/analysis/?...
Jan 24, 2014 - File type, PE32 executable for MS Windows (GUI) Intel 80386 32-bit. Language, 040904b0. Section .text md5: ...

e094d944954303f06d769b89a46e650cc347dc4f - Analysis ...
https://totalhash.cymru.com/analysis/?...
Jan 1, 2014 - ... BMXs:TR B:Q+= "bsTs p" =y/KB+G }.C8nQA c.ae) C:\ A_E_O_FANTOMA_DE_FISIERCARE_VA_SA_ZICA_NU_EXISTA.BAT California #0!
Future Work

• More emulators, more tests
• Use AVLeak for vulnerability research against emulators (breakout exploits)
  – See Tavis Ormandy and Joxeart 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 kernel 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
Selected References


Thank You

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

• Help & Inspiration:
  – Marion Marshalek
  – Rolf Rolles
  – Alex Ionescu
  – Bruce Dang
  – Dr. Sergey Bratus
Questions?

Kaspersky Lab - Packin’ The K