Iñaki Rodríguez-Gastón†, Ricardo J. Rodríguez‡

All wrongs reversed

inaki@sensepost.com, rjrodriguez@fi.upm.es
@virtualminds_es ※ @RicardoJRdez

†SensePost
London, UK
‡Universidad Politécnica de Madrid
Madrid, Spain

1 de Noviembre, 2013

No cON Name 2013
Barcelona (España)
whoarewe: command not found

- CLS member (2001)
- Ph.D. by UZ (2013)
- Working for UPM
- Trainee @ NcN, RootedCON, HIP
- Speaker @ NcN, HackLU, RootedCON, STIC CCN, HIP

- CISSP, CEH, GWAPT
- Security analyst @ SensePost
- Malware lover
- mlw.re staff
- Trainee @ 44CON
- ...

I. Rodríguez-Gastón, R.J. Rodríguez
One FIAw over the Cuckoo’s Nest
Outline

1. Motivation
2. Previous Concepts
   - Cuckoo Sandbox
   - Dynamic Binary Instrumentation: The Pin Framework
3. On the Anti-VMs & Anti-Sandboxing Techniques
   - VM Detection
   - Sandboxing detection
4. Mixing Cuckoo Sandbox and Pin DBI
   - Sticking both Programs
   - Introducing PinVMShield
5. Case Study: the pafish tool
6. Related Work
7. Conclusions and Future Work
Outline

1. **Motivation**
2. **Previous Concepts**
   - Cuckoo Sandbox
   - Dynamic Binary Instrumentation: The Pin Framework
3. **On the Anti-VMs & Anti-Sandboxing Techniques**
   - VM Detection
   - Sandboxing detection
4. **Mixing Cuckoo Sandbox and Pin DBI**
   - Sticking both Programs
   - Introducing PinVMShield
5. **Case Study: the pafish tool**
6. **Related Work**
7. **Conclusions and Future Work**
Motivation (I)

- Malware are **increasing in number and complexity**
- **Targeted attacks** also grown (specially industry and government espionage)

**How do we currently fight against malware?**

- Firstly, to understand how a sample works (**what is it doing?**)
- Then, to figure out **how it can be removed**
- Lastly, to avoid future infections (**can we detect it again?**)
Figuring out what it is doing...

- Manual analysis
  - Intensive
  - Time-consuming
Motivation (II)

Figuring out what it is doing...

- **Manual analysis**
  - Intensive
  - Time-consuming
  - Good if you are paid per working hour 😊

- **Automatic analysis**
  - Just take a seat, and relax...
  - Real problem here: automation of malware analysis tasks
Motivation (II)

Figuring out what it is doing...

- **Manual analysis**
  - Intensive
  - Time-consuming
  - Good if you are paid per working hour 😊

- **Automatic analysis**
  - Just take a seat, and relax...
  - Real problem here: automation of malware analysis tasks
  - Only manual analysis for weird (or interesting) samples
Sandbox Environments

- Computer resources are tightly controlled and monitored
- Current trending of malware analysis
- Commercial and free-license solutions
  - Sandboxie
  - JoeBox
  - CWSandbox
  - Cuckoo Sandbox
  - PyBox
- Virtual Machine and Sandbox: a good combination
Sandbox Environments

- Computer resources are tightly controlled and monitored
- Current trending of malware analysis
- Commercial and free-license solutions
  - Sandboxie
  - JoeBox
  - CWSandbox
  - Cuckoo Sandbox
  - PyBox
- Virtual Machine and Sandbox: a good combination

Do malware samples detect VMs/sandbox environments?
### Sandbox Environments

- Computer resources are tightly controlled and monitored
- Current **trending** of malware analysis
- Commercial and free-license solutions
  - Sandboxie
  - JoeBox
  - CWSandbox
  - Cuckoo Sandbox
  - PyBox
- **Virtual Machine and Sandbox**: a good combination

Do malware samples detect VMs/sandbox environments? Yes, they do.
Can we avoid the detection of a VMs/sandbox environment?
Can we avoid the detection of a VMs/sandbox environment? Yes, we can! (at least, we should try...)
Can we avoid the detection of a VMs/sandbox environment? Yes, we can! (at least, we should try...) We’re gonna do it in a fancy way...

using Dynamic Binary Instrumentation 😊

**Dynamic Binary Instrumentation (DBI)**
- Analyse the runtime behaviour of a binary
- Executes arbitrary code during normal execution of a binary
**Motivation (V)**

Why DBI? Its advantages

**Binary instrumentation: advantages**

- Programming language (totally) **independent**
- **Machine-mode** vision
- We can instrument **proprietary software**
Motivation (V)
Why DBI? Its advantages

**Binary instrumentation: advantages**
- Programming language (totally) independent
- Machine-mode vision
- We can instrument proprietary software

**Dynamic Instrumentation: advantages**
- No need to recompile/relink each time
- Allow to find *on-the-fly code*
- Dynamically generated code
- Allow to instrument a process in execution already (*attach*)
Main disadvantages

- **Overhead** (by the instrumentation during execution)
- ↓ **performance** (analyst hopelessness!)
- Single execution path analysed
Our goal in this work

- Develop a Dynamic Binary Analysis (DBA) tool
  - Integrated with Cuckoo Sandbox
Motivation (VII)

Summary of contributions

Our goal in this work

- Develop a Dynamic Binary Analysis (DBA) tool
  - Integrated with Cuckoo Sandbox
  - Protects Cuckoo for being detected...
Motivation (VII)

Summary of contributions

Our goal in this work

- Develop a Dynamic Binary Analysis (DBA) tool
  - Integrated with Cuckoo Sandbox
  - Protects Cuckoo for being detected...
  - ...and also for (some) VMs detection
Outline

1. **Motivation**

2. **Previous Concepts**
   - Cuckoo Sandbox
   - Dynamic Binary Instrumentation: The Pin Framework

3. **On the Anti-VMs & Anti-Sandboxing Techniques**
   - VM Detection
   - Sandboxing detection

4. **Mixing Cuckoo Sandbox and Pin DBI**
   - Sticking both Programs
   - Introducing PinVMShield

5. **Case Study: the pafish tool**

6. **Related Work**

7. **Conclusions and Future Work**
Cuckoo Sandbox (I)

<table>
<thead>
<tr>
<th>What is Cuckoo Sandbox?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automated malware analysis tool</td>
</tr>
<tr>
<td>Written in Python</td>
</tr>
<tr>
<td>Reporting system (<em>API calls, registry access, network activity</em>)</td>
</tr>
<tr>
<td>Extensible</td>
</tr>
<tr>
<td>OpenSource</td>
</tr>
</tbody>
</table>
Cuckoo Sandbox (II)

Previous Concepts

Cuckoo Sandbox

I. Rodríguez-Gastón, R.J. Rodríguez

One FlAw over the Cuckoo’s Nest

1 Nov’13 14 / 39
Dynamic Binary Instrumentation: The Pin Framework

http://www.pintools.org

What is Pin?

- Framework designed by Intel
- Allows to build easy-to-use, portable, transparent and efficient instrumentation tools (DBA, or Pintools)
- Recall: instrumentation enables the execution of arbitrary code during run-time of a binary
- Extensive API for doing whatever you can imagine
- Used for things like:
  - Instruction profiling
  - Performance evaluation
  - Bug detection
  - And malware analysis (here we are 😃)
Dynamic Binary Instrumentation: The Pin Framework (II)

How does Pin work?

---

Pin

Application

Virtual Machine (VM)

JIT Compiler

Dispatcher

Emulation Unit

Operating System (OS)

same address space

Pintool

Instrumentation APIs
Outline

1 Motivation

2 Previous Concepts
   - Cuckoo Sandbox
   - Dynamic Binary Instrumentation: The Pin Framework

3 On the Anti-VMs & Anti-Sandboxing Techniques
   - VM Detection
   - Sandboxing detection

4 Mixing Cuckoo Sandbox and Pin DBI
   - Sticking both Programs
   - Introducing PinVMShield

5 Case Study: the pafish tool

6 Related Work

7 Conclusions and Future Work
How can an execution inside a VM be detected?
How can an execution inside a VM be detected?

**Detection ways**

- Seek VME artifacts in processes, system files and/or registry
How can an execution inside a VM be detected?

**Detection ways**

- Seek VME artifacts in processes, system files and/or registry
- Seek VME artifacts in memory
How can an execution inside a VM be detected?

Detection ways:

- Seek VME artifacts in processes, system files and/or registry
- Seek VME artifacts in memory
- Seek specific features of virtualised hardware
How can an execution inside a VM be detected?

Detection ways

- Seek VME artifacts in processes, system files and/or registry
- Seek VME artifacts in memory
- Seek specific features of virtualised hardware
- Seek CPU instructions specific to VME
## On the Anti-VMs & Anti-Sandboxing Techniques (II)

Artifacts in processes, system files and/or registry

### Some examples

<table>
<thead>
<tr>
<th>VMWare</th>
</tr>
</thead>
<tbody>
<tr>
<td>“VMTools” service</td>
</tr>
<tr>
<td>References in system files to “VMWare” and vmx</td>
</tr>
<tr>
<td>References in the registry to “VMWare”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VirtualBox</th>
</tr>
</thead>
<tbody>
<tr>
<td>VBoxService.exe process (&quot;VirtualBoxGuestAdditions&quot;)</td>
</tr>
<tr>
<td>References in the registry to “VBox”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MS Virtual PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmsrvc.exe, vpcmap.exe, vmusrvc.exe processes</td>
</tr>
<tr>
<td>References in the registry to “Virtual”</td>
</tr>
</tbody>
</table>
The Red Pill

- Software developed by Joanna Rutkwoska, 2004
- Uses the **SIDT instruction** (Store Interrupt Descriptor Table)
  - VMWare: IDT in 0xFFxxxxxx
  - VirtualPC: IDT in 0xE8xxxxxx
  - In real machines: Windows (0x80FFFFFF), Linux (0xC0FFFFFF)

Other options: GDT, LDT

- GDT, LDT also displaced in virtual environments
- **Scoopy** tool (http://www.trapkit.de)
  - (IDT == 0xC0) || IDT == 0x80
  - GDT == 0xC0
  - LDT == 0x00
Specific virtualised hardware

- Network controller
- USBs controller
- Host controller
- ...

On the Anti-VMs & Anti-Sandboxing Techniques (V)
Specific features of virtualised hardware
Specific virtualised hardware

- Network controller
- USBs controller
- Host controller
- ...

Seek specific “fingerprints”

- SCSI device type
- Network controller MAC
- Host controller type
- ...

On the Anti-VMs & Anti-Sandboxing Techniques (V)
Specific features of virtualised hardware
On the Anti-VMs & Anti-Sand boxing Techniques (V)

Specific features of virtualised hardware

- Specific virtualised hardware
  - Network controller
  - USBs controller
  - Host controller
  - ...

- Seek specific “fingerprints”
  - SCSI device type
  - Network controller MAC
  - Host controller type
  - ...

- Doo tool (also seeks Class IDs in the registry)
On the Anti-VMs & Anti-Sandboxing Techniques (VI)

CPU instructions specific to VME

- Some VMs **add/use own instructions to communicate host/guest**
Some VMs add/use own instructions to communicate host/guest
Seek host/guest communication channel
On the Anti-VMs & Anti-Sandboxing Techniques (VI)

CPU instructions specific to VME

- Some VMs add/use own instructions to communicate host/guest
- Seek host/guest communication channel
- Jerry tool
- VMDetect tool
Some VMs add/use own instructions to communicate host/guest
Seek host/guest communication channel
Jerry tool
VMDetect tool
Magic number... CONSTANT (WTF!)

```c
mov eax, 564D5868h ; "VMXh"
mov ebx, 0
mov ecx, 0Ah
mov edx, 5658 ; "VX"
in eax, dx
cmp ebx, 564D5868h
```
Sandbox

- Binary execution in controlled environment
- Examples: Sandboxie, Norman Sandbox Analyser, Anubis, Cuckoo, WinJail...
- They have some common and recognisable issues:
  - DLLs loaded
  - Read of ProductID key
  - Windows username (API `GetUserName`)
  - Window handle (API `FindWindow`)

On the Anti-VMs & Anti-Sandboxing Techniques (VIII)
Outline

1 Motivation

2 Previous Concepts
   - Cuckoo Sandbox
   - Dynamic Binary Instrumentation: The Pin Framework

3 On the Anti-VMs & Anti-Sandboxing Techniques
   - VM Detection
   - Sandboxing detection

4 Mixing Cuckoo Sandbox and Pin DBI
   - Sticking both Programs
   - Introducing PinVMShield

5 Case Study: the pafish tool

6 Related Work

7 Conclusions and Future Work
Every file has a package

Best place for the integration:
- Attaching Pin to the suspended process
- Directly executing the sample with Pin

Pin and cuckoomon together
Mixing Cuckoo Sandbox and Pin DBI (II)

Attach to suspended process

- `agent.py`
- `analyzer.py`
- `packages (exe.py)`
- `\\PIPE\random string`
- `Sample.exe`
- `Cuckoomon.dll (Random name)`
- `PIN (Attach)`
- Drop file
- API call
- Executes process and injects `cuckoomon.dll`
- Attach PIN to the sample
Mixing Cuckoo Sandbox and Pin DBI (III)

Pin integrated into a package

- agent.py
- analyzer.py
- packages (exe.py)
- Drop file
- API call
- PIN (Suspended)
- Sample.exe
- Cuckoomon.dll (Random name)
- \PIPE\random string

Executes process and injects cuckoomon.dll
Mixing Cuckoo Sandbox and Pin DBI (IV)
Introducing PinVMShield (1)

<<Interface>>

Pin
main(argc : int, argv[] : char*) : int
Fini(code : INT32, *v : void) : void

PinVMShield
Image(img : IMG, *v : void)

PinWrapperWinAPI
#funcName[MAX_LENGTH_FUNCNAME] : char
#isTryingToDetectVMs(*s : char) : bool
#FindRTNByNameA(img : IMG) : RTN
#FindRTNByNameW(img : IMG) : RTN
#GetPrototypeFunctionA() : PROTO
#GetPrototypeFunctionW() : PROTO
#GetPrototypeFunction(*funcName : char) : PROTO
#ReplaceFunctionSignature(rtn: RTN, proto: PROTO) : void

WrapperGetModuleHandle

WrapperProcess32FirstAndNext
Mixing Cuckoo Sandbox and Pin DBI (...): our Tool

Introducing PinVMShield (2)

APIs fooled

- GetUser NameA/W
- GetUserNameExA/W
- RegQueryValueA/W
- RegQueryValueExA/W
- RegOpenKeyA/W
- RegOpenKeyExA/W
- GetModuleHandleA/W
- GetModuleHandleExA/W
- GetFileAttributesA/W
- Process32First /
- Process32Next
- FindWindowA/W
- FindWindowExA/W
- CreateFileA/W
- CreateNamedPipeA/W
- GetCursorPos

Alpha version available for download: (soon)
https://bitbucket.org/rjrodriguez/pinvmshield/
Outline

1. Motivation
2. Previous Concepts
   - Cuckoo Sandbox
   - Dynamic Binary Instrumentation: The Pin Framework
3. On the Anti-VMs & Anti-Sandboxing Techniques
   - VM Detection
   - Sandboxing detection
4. Mixing Cuckoo Sandbox and Pin DBI
   - Sticking both Programs
   - Introducing PinVMSHield
5. Case Study: the pafish tool
6. Related Work
7. Conclusions and Future Work
Case Study: the pafish tool (I)

- Tool that incorporates several detections for vms and sandboxing
- Developed by Alberto Ortega
- In v.0.2.5.1 (the one of case study):
  - Generic Sandbox
  - Sandboxie
  - QEMU
  - Wine
  - VirtualBox
  - VMWare
Case Study: the pafish tool (I)

- Tool that incorporates several detections for vms and sandboxing
- Developed by Alberto Ortega
- In v.0.2.5.1 (the one of case study):
  - Generic Sandbox
  - Sandboxie
  - QEMU
  - Wine
  - VirtualBox
  - VMWare

Do you wanna know more about the blue fish?
Tool that incorporates several detections for vms and sandboxing

Developed by Alberto Ortega

In v.0.2.5.1 (the one of case study):
- Generic Sandbox
- Sandboxie
- QEMU
- Wine
- VirtualBox
- VMWare

Do you wanna know more about the blue fish?
→ attend Alberto’s session! (tomorrow afternoon)
It’s demo time!
Outline

1 Motivation

2 Previous Concepts
   - Cuckoo Sandbox
   - Dynamic Binary Instrumentation: The Pin Framework

3 On the Anti-VMs & Anti-Sandboxing Techniques
   - VM Detection
   - Sandboxing detection

4 Mixing Cuckoo Sandbox and Pin DBI
   - Sticking both Programs
   - Introducing PinVMShield

5 Case Study: the pafish tool

6 Related Work

7 Conclusions and Future Work
### Related Work (I)

**CWSandbox**

- Sandbox environment
- **Three design criteria**: automation, effectiveness and correctness
- Performs a dynamic analysis
- API hooking
CWSandbox

- Sandbox environment
- **Three design criteria**: automation, effectiveness and correctness
- Performs a **dynamic analysis**
- API hooking
- It is detected by sandbox detection techniques
Related Work (II)

Sandbox + DBI

- Pin as DBI framework
- Own-created sandbox environment
- Two execution environments:
  - **Testing**: binary execution is traced. Traces are checked against some security policies
  - **Real**: binary execution is monitored avoiding harmful behaviours
Related Work (II)

**Sandbox + DBI**

- **Pin as DBI framework**
- **Own-created sandbox environment**
- **Two execution environments:**
  - **Testing**: binary execution is traced. Traces are checked against some security policies
  - **Real**: binary execution is monitored avoiding harmful behaviours

Our solution also monitors the execution but...
Related Work (II)

Sandbox + DBI

- Pin as DBI framework
- Own-created sandbox environment
- Two execution environments:
  - Testing: binary execution is traced. Traces are checked against some security policies
  - Real: binary execution is monitored avoiding harmful behaviours

Our solution also monitors the execution but... besides avoids sandbox detection!
Outline

1. Motivation
2. Previous Concepts
   - Cuckoo Sandbox
   - Dynamic Binary Instrumentation: The Pin Framework
3. On the Anti-VMs & Anti-Sandboxing Techniques
   - VM Detection
   - Sandboxing detection
4. Mixing Cuckoo Sandbox and Pin DBI
   - Sticking both Programs
   - Introducing PinVMShield
5. Case Study: the pafish tool
6. Related Work
7. Conclusions and Future Work
Conclusions and Future Work (I)

PinVMShield

- Integrated with Cuckoo Sandbox
## Conclusions and Future Work (I)

### PinVMShield

- **Integrated with Cuckoo Sandbox**
- ✔️ **Avoids Cuckoo (and other) detections** commonly realised by malware
## Conclusions and Future Work (I)

<table>
<thead>
<tr>
<th>PinVMShield</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Integrated with Cuckoo Sandbox</td>
</tr>
<tr>
<td>✓ Avoids Cuckoo (and other) detections commonly realised by malware</td>
</tr>
<tr>
<td>✓ Not currently detected! 😊</td>
</tr>
</tbody>
</table>
### PinVMShield

- **Integrated with Cuckoo Sandbox**
- ✓ Avoids Cuckoo (and other) detections commonly realised by malware
- ✓ Not currently detected! 😊
- X Main drawback: runtime
PinVMShield

- Integrated with Cuckoo Sandbox
- Avoids Cuckoo (and other) detections commonly realised by malware
- Not currently detected! 😊
- Main drawback: runtime
- Coding C++ is like a pain in the ass
Conclusions and Future Work (I)

**PinVMShield**

- Integrated with Cuckoo Sandbox
- Avoids Cuckoo (and other) detections commonly realised by malware
- Not currently detected! 😊
- Main drawback: runtime
- Coding C++ is like a pain in the ass

We do have more control on malware (binary) execution
## Future Work

- Find a logo
- Stand-alone app
- Improve anti-detection techniques *(not only hooking...)*
Conclusions and Future Work (II)

Future Work

- Find a logo
- **Stand-alone app**
- Improve anti-detection techniques (*not only hooking...*)
- Replace (totally) *cuckoomon.dll*
Future Work

- Find a logo
- Stand-alone app
- Improve anti-detection techniques (*not only hooking...*)
- Replace (totally) cuckoomon.dll
- Add anti-DBI techniques
Future Work

- Find a logo
- **Stand-alone app**
- Improve anti-detection techniques (*not only hooking...*)
- Replace (totally) cuckoomon.dll
- Add anti-DBI techniques
- Test in real malware samples
Conclusions and Future Work (II)

Future Work

- Find a logo
- Stand-alone app
- Improve anti-detection techniques (not only hooking...)
- Replace (totally) cuckoomon.dll
- Add anti-DBI techniques
- Test in real malware samples

Acknowledgments

- Alberto Ortega (pafish)
- NcN staff
Iñaki Rodríguez-Gastón†, Ricardo J. Rodríguez‡

All wrongs reversed

inaki@sensepost.com, rjrodriguez@fi.upm.es
@virtualminds_es ※ @RicardoJRdez

†SensePost
London, UK
‡Universidad Politécnica de Madrid
Madrid, Spain

1 de Noviembre, 2013

No cON Name 2013
Barcelona (España)