One FlAw over the Cuckoo's Nest

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
- . . .

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?)

Motivation (II)

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 $\ddot\smile$
- 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

Motivation (III)

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

Motivation (III)

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?

Motivation (III)

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.

Motivation (IV)

Can we avoid the detection of a VMs/sandbox environment?

Motivation (IV)

Can we avoid the detection of a VMs/sandbox environment?
Yes, we can! (at least, we should try. . .)

Motivation (IV)

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*)

Motivation (VI)

Why DBI? Its disadvantages

Main disadvantages

- Overhead (by the instrumentation during execution)
- ⇓ performance (analyst hopelessness!)
- Single execution path analysed

Motivation (VII)

Summary of contributions

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

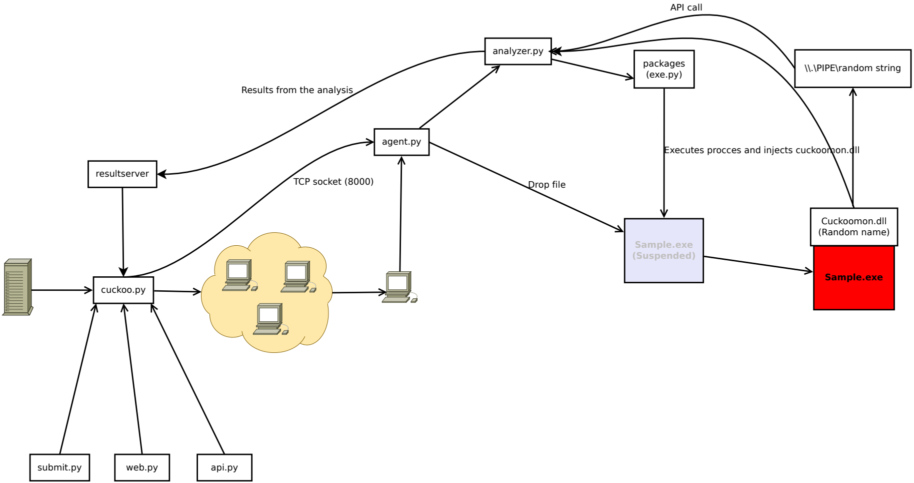
1. Motivation
2. Previous Concepts
   - Cuckoo Sandbox
   - Dynamic Binary Instrumentation: The Pin Framework
3. On the Anti-VMs & Anti-Sandboxing Techniques
   - VM Detection
   - Sandboxing detection
4. Mixing Cuckoo Sandbox and Pin DBI
   - Sticking both Programs
   - Introducing `PinVMShield`
5.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

Cuckoo Sandbox (I)

What is Cuckoo Sandbox?

- Automated malware analysis tool
- Written in Python
- Reporting system (API calls, registry access, network activity)
- Extensible
- OpenSource

Cuckoo Sandbox (II)



Dynamic Binary Instrumentation: The Pin Framework (I)

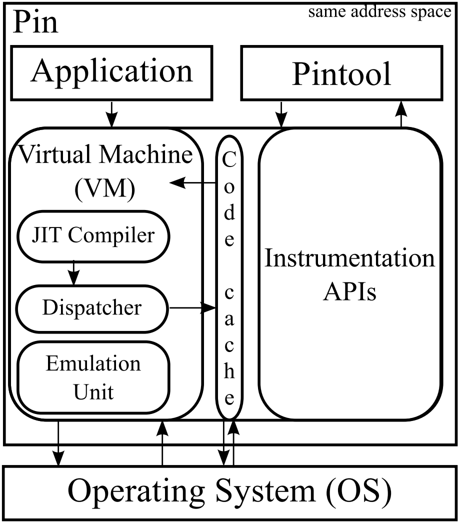
`http://www.pintools.org`

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?



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

On the Anti-VMs & Anti-Sandboxing Techniques (I)

How can an execution inside a VM be detected?

On the Anti-VMs & Anti-Sandboxing Techniques (I)

How can an execution inside a VM be detected?

Detection ways

- Seek VME artifacts in processes, system files and/or registry

On the Anti-VMs & Anti-Sandboxing Techniques (I)

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

On the Anti-VMs & Anti-Sandboxing Techniques (I)

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

On the Anti-VMs & Anti-Sandboxing Techniques (I)

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

- VMWare
  - “VMTools” service
  - References in system files to “VMWare” and vmx
  - References in the registry to “VMWare”
- VirtualBox
  - `VBoxService.exe` process (“VirtualBoxGuestAdditions”)
  - References in the registry to “VBox”
- MS Virtual PC
  - `vmsrvc.exe`, `vpcmap.exe`, `vmusrvc.exe` processes
  - References in the registry to “Virtual”

On the Anti-VMs & Anti-Sandboxing Techniques (III)

Artifacts in memory

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`

On the Anti-VMs & Anti-Sandboxing Techniques (V)

Specific features of virtualised hardware

- 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

- 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

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

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

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
- Magic number. . . CONSTANT (WTF!)

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

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

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`)

Outline

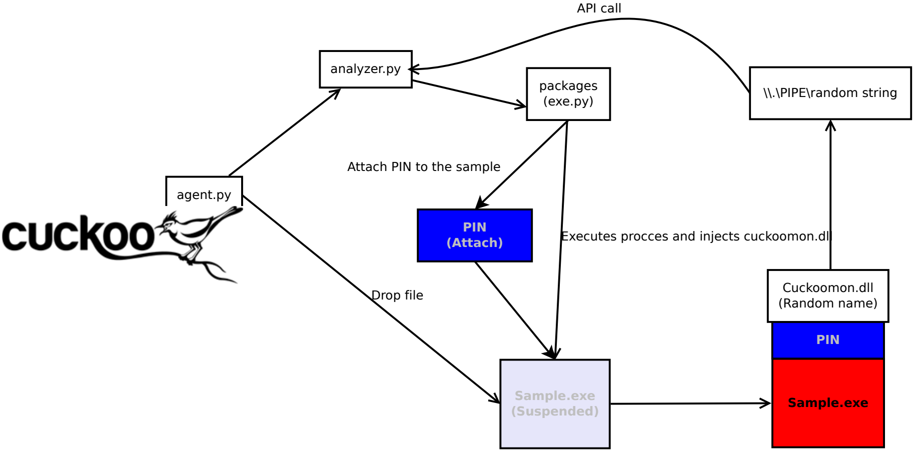
1. Motivation
2. Previous Concepts
   - Cuckoo Sandbox
   - Dynamic Binary Instrumentation: The Pin Framework
3. On the Anti-VMs & Anti-Sandboxing Techniques
   - VM Detection
   - Sandboxing detection
4. Mixing Cuckoo Sandbox and Pin DBI
   - Sticking both Programs
   - Introducing `PinVMShield`
5.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

Mixing Cuckoo Sandbox and Pin DBI (I)

- 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

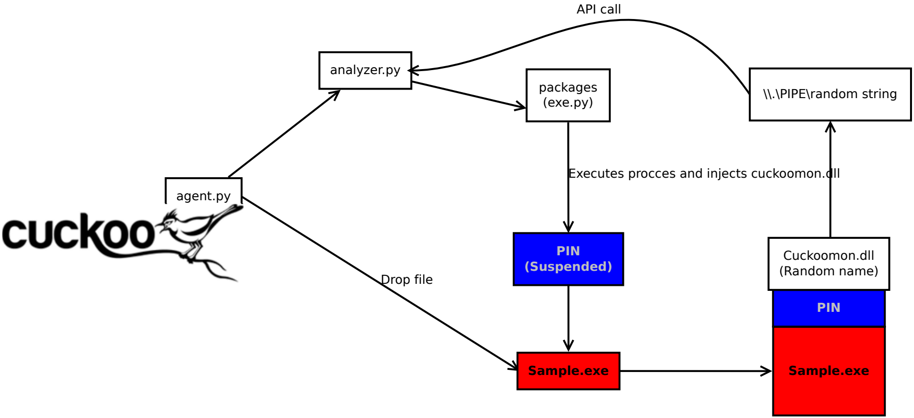
Mixing Cuckoo Sandbox and Pin DBI (II)

Attach to suspended process



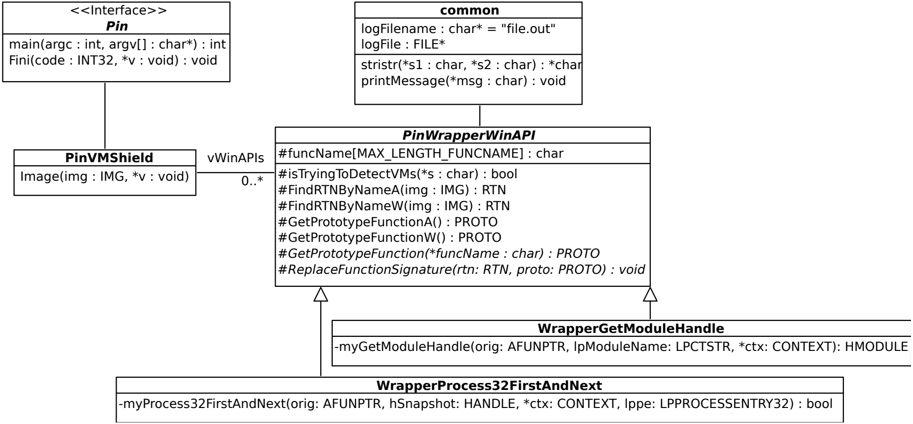
Mixing Cuckoo Sandbox and Pin DBI (III)

Pin integrated into a package



Mixing Cuckoo Sandbox and Pin DBI (IV)

Introducing PinVMShield (1)



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

common
logFilename : char* = "file.out"
logFile : FILE*
stristr(*s1 : char, *s2 : char) : *char
printMessage(*msg : char) : void

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

vWinAPIs
0..*

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
-myGetModuleHandle(orig: AFUNPTR, lpModuleName: LPCTSTR, *ctx: CONTEXT): HMODULE

WrapperProcess32FirstAndNext
-myProcess32FirstAndNext(orig: AFUNPTR, hSnapshot: HANDLE, *ctx: CONTEXT, lppe: LPPROCESSENTRY32) : bool

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

Introducing PinVMShield (2)

APIs fooled

- GetUserNameA/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

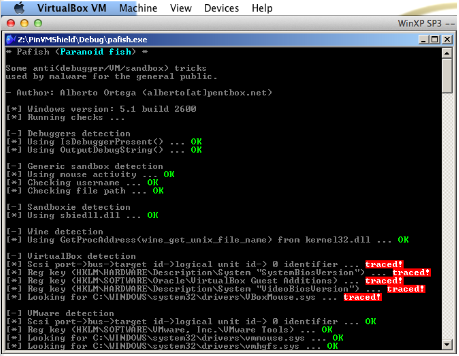
5 Case Study: the pafish tool

6 Related Work

7 Conclusions and Future Work

Case Study: the pafish tool (I)

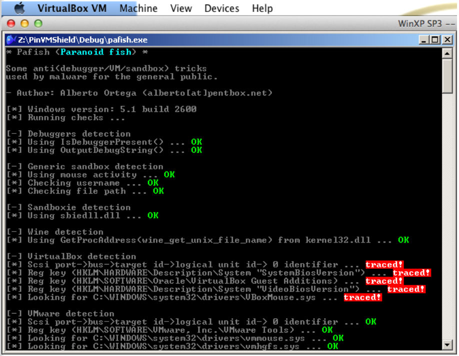
- Tool that incorporates several detections for vms and sandboxing
- Developed by Alberto Ortega
- In v.0.2.5.1 (the one of case study):
  - Generic Sandbox
  - Sandboxie
  - QEMU
  - Wine
  - VirtualBox
  - VMWare



```
* Pafish (Paranoid fish) *

Some anti(debugger/VM/sandbox) tricks
used by malware for the general public.

- Author: Alberto Ortega (alberto[at]pentbox.net)

[*] Windows version: 5.1 build 2600
[*] Running checks ...

[-] Debuggers detection
[*] Using IsDebuggerPresent() ... OK
[*] Using OutputDebugString() ... OK

[-] Generic sandbox detection
[*] Using mouse activity ... OK
[*] Checking username ... OK
[*] Checking file path ... OK

[-] Sandboxie detection
[*] Using sbiedll.dll ... OK

[-] Wine detection
[*] Using GetProcAddress(wine_get_unix_file_name) from kernel32.dll ... OK

[-] VirtualBox detection
[*] Scsi port->bus->target id->logical unit id-> 0 identifier ... traced!
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... traced!
[*] Reg key (HKLM\SOFTWARE\Oracle\VirtualBox Guest Additions) ... traced!
[*] Reg key (HKLM\HARDWARE\Description\System "VideoBiosVersion") ... traced!
[*] Looking for C:\WINDOWS\system32\drivers\VBoxMouse.sys ... traced!

[-] VMware detection
[*] Scsi port->bus->target id->logical unit id-> 0 identifier ... OK
[*] Reg key (HKLM\SOFTWARE\VMware, Inc.\VMware Tools) ... OK
[*] Looking for C:\WINDOWS\system32\drivers\vmmouse.sys ... OK
[*] Looking for C:\WINDOWS\system32\drivers\vmhgfs.sys ... OK
```

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

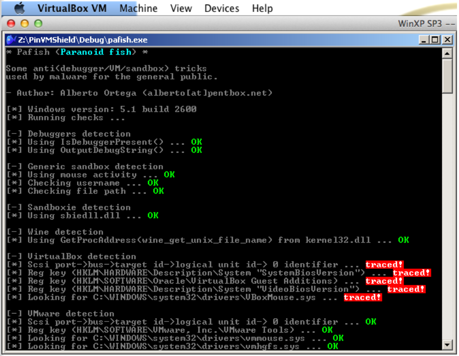


```
* Pafish (Paranoid fish) *

Some anti(debugger/VM/sandbox) tricks
used by malware for the general public.

- Author: Alberto Ortega (alberto[at]pentbox.net)

[*] Windows version: 5.1 build 2600
[*] Running checks ...

[-] Debuggers detection
[*] Using IsDebuggerPresent() ... OK
[*] Using OutputDebugString() ... OK

[-] Generic sandbox detection
[*] Using mouse activity ... OK
[*] Checking username ... OK
[*] Checking file path ... OK

[-] Sandboxie detection
[*] Using sbiedll.dll ... OK

[-] Wine detection
[*] Using GetProcAddress(wine_get_unix_file_name) from kernel32.dll ... OK

[-] VirtualBox detection
[*] Scsi port->bus->target id->logical unit id-> 0 identifier ... traced!
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... traced!
[*] Reg key (HKLM\SOFTWARE\Oracle\VirtualBox Guest Additions) ... traced!
[*] Reg key (HKLM\HARDWARE\Description\System "VideoBiosVersion") ... traced!
[*] Looking for C:\WINDOWS\system32\drivers\VBoxMouse.sys ... traced!

[-] VMware detection
[*] Scsi port->bus->target id->logical unit id-> 0 identifier ... OK
[*] Reg key (HKLM\SOFTWARE\VMware, Inc.\VMware Tools) ... OK
[*] Looking for C:\WINDOWS\system32\drivers\vmmouse.sys ... OK
[*] Looking for C:\WINDOWS\system32\drivers\vmhgfs.sys ... OK
```

Do you wanna know more about the blue fish?

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



```
* Pafish (Paranoid fish) *

Some anti(debugger/VM/sandbox) tricks
used by malware for the general public.

- Author: Alberto Ortega (alberto[at]pentbox.net)

[*] Windows version: 5.1 build 2600
[*] Running checks ...

[-] Debuggers detection
[*] Using IsDebuggerPresent() ... OK
[*] Using OutputDebugString() ... OK

[-] Generic sandbox detection
[*] Using mouse activity ... OK
[*] Checking username ... OK
[*] Checking file path ... OK

[-] Sandboxie detection
[*] Using sbiedll.dll ... OK

[-] Wine detection
[*] Using GetProcAddress(wine_get_unix_file_name) from kernel32.dll ... OK

[-] VirtualBox detection
[*] Scsi port->bus->target id->logical unit id-> 0 identifier ... traced!
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... traced!
[*] Reg key (HKLM\SOFTWARE\Oracle\VirtualBox Guest Additions) ... traced!
[*] Reg key (HKLM\HARDWARE\Description\System "VideoBiosVersion") ... traced!
[*] Looking for C:\WINDOWS\system32\drivers\VBoxMouse.sys ... traced!

[-] VMware detection
[*] Scsi port->bus->target id->logical unit id-> 0 identifier ... OK
[*] Reg key (HKLM\SOFTWARE\VMware, Inc.\VMware Tools) ... OK
[*] Looking for C:\WINDOWS\system32\drivers\vmmouse.sys ... OK
[*] Looking for C:\WINDOWS\system32\drivers\vmhgfs.sys ... OK
```

Do you wanna know more about the blue fish?
→ attend Alberto's session! (tomorrow afternoon)

Case Study: the `pafish` tool (II)

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)

CWSandox

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

Related Work (I)

CWSandox

- 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)

PinVMShield

- Integrated with Cuckoo Sandbox

✓ Avoids Cuckoo (and other) detections commonly realised by malware

✓ Not currently detected! $\ddot\smile$

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

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

Conclusions and Future Work (I)

PinVMShield

- Integrated with Cuckoo Sandbox

✓ Avoids Cuckoo (and other) detections commonly realised by malware

✓ Not currently detected! ☺

X Main drawback: runtime

X Coding C++ is like a pain in the ass

We do have more control on malware (binary) execution

Conclusions and Future Work (II)

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`

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

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

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

One FlAw over the Cuckoo's Nest

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)