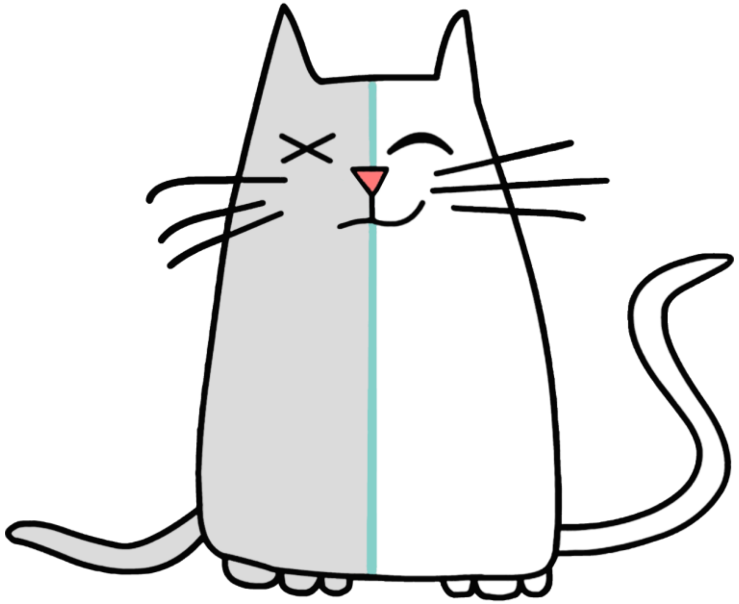


HyperDbg Debugger

A debugger designed for analyzing, fuzzing, and reversing

Who Am I?

Sina Karvandi

Researcher, Institute For Research In Fundamental Sciences (**IPM**)

Spring 2022

پژوهشگاه‌دانش‌های‌بنیادی (مرکز تحقیقات فیزیک نظری و ریاضیات)

IPM

INSTITUTE FOR RESEARCH IN FUNDAMENTAL SCIENCES

“

If debugging is the process of removing software bugs, then programming must be the process of putting them in.

Edsger Dijkstra

Before start...

Website

https://hyperdbg.com

Documentation

https://docs.hyperdbg.com

Doxygen

https://doxygen.hyperdbg.com

Source code (GitHub)

https://github.com/HyperDbg

Social Networks

https://twitter.com/HyperDbg

https://youtube.com/c/HyperDbg

Why a debugger?

Programming Research

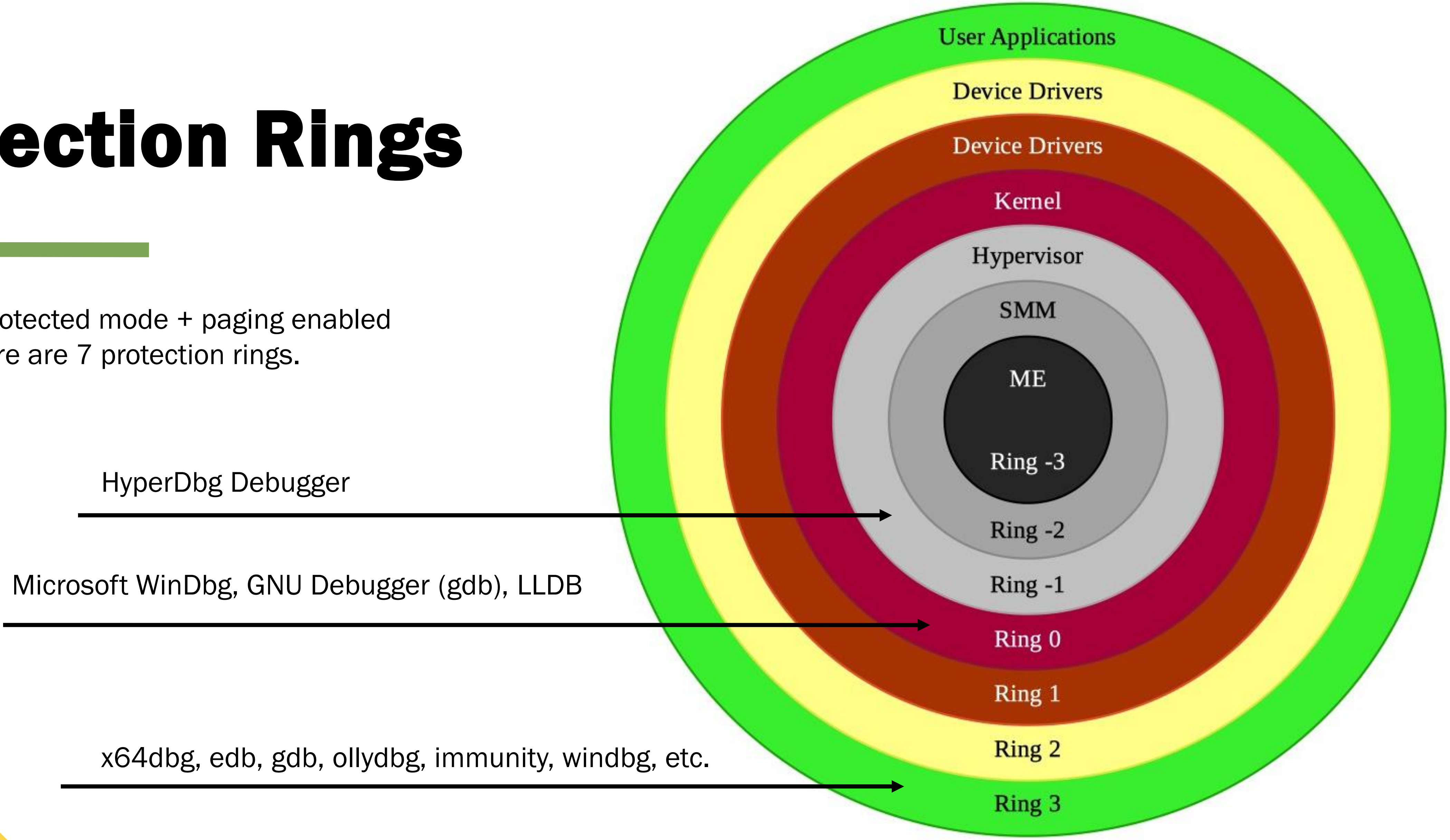
- Finding bugs,
- OS/Application level functionality test
- A platform to use modern processor features,
- Performance monitoring & statistical analysis
- etc.

Security Research

- The main tools for reverse engineering,
- Analyzing system and application behaviors,
- Fuzzing assistant,
- Discover and fix vulnerability,
- etc.

Protection Rings

In modern protected mode + paging enabled systems, there are 7 protection rings.



IA Negative Rings

Kernel Debugger Family Members

WinDbg,

- Over 30 years of develop
- Windows is made by using WinDbg
- Not open-source but its source code leaked multiple times

LLDB,

- Mostly used as OS X debugger by researchers
- Open-source

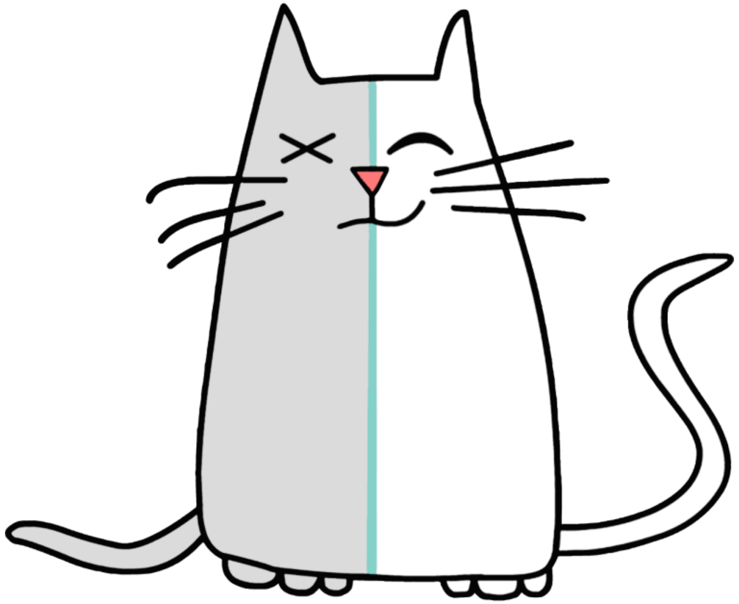
GDB,

- Main kernel debugger for Linux
- Open-source

A new family member

HyperDbg Debugger,

- More privileged (Rings)
- Unique features
- An efficient and complicated design
- Hidden by its nature
- Academic innovation combined with practical implementation
- Open-source



You can debug WinDbg or any other kernel debug with HyperDbg :)

Concepts

Concepts

Execution Modes

- VMI Mode
  - Virtual Machine Introspection Mode
  - Also known as Local Debugging
- Debugger Mode
- Transparent Mode

Events

- Everything in HyperDbg is an event
  - Breakpoints are events
  - EPT hooks are events
  - Syscall executions are events
  - etc.
- Consist of zero to n actions
- Either
  - Conditional
  - Unconditional

Actions

- Each action is either
  - **Break**
  - **Script**
  - **Custom code**

Windbg
HyperDbg
imgflip.com

Events and Features

Features based on emulating systems’ behavior

Hooking system-Calls

Hooking system calls is possible by using !syscall command

Hooking return of system-calls

Hooking the result of system-calls is possible by using !sysret command.

!syscall command

Description

Triggers when the debugging machine executes a **syscall** instruction or, in other words, when Windows tries to run a system call, this event will be triggered.

Features

- Fast & Transparent

Read more

- https://docs.hyperdbg.com/commands/extension-commands/syscall
- https://docs.hyperdbg.com/design/features/vmm-module/design-of-syscall-and-sysret

!sysret command

Description

Triggers when the debugging machine executes a **sysret** instruction or, in other words, when Windows tries to return to user-mode from a previous **syscall**.

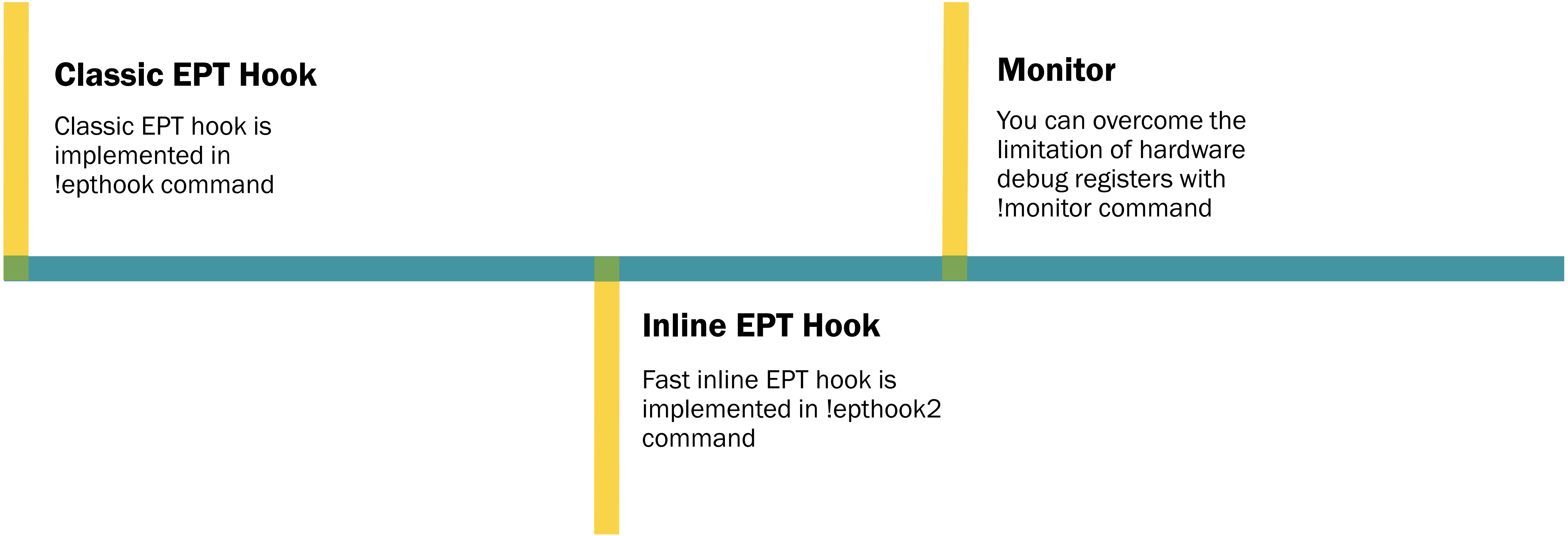
Features

- Fast & Transparent

Read more

- https://docs.hyperdbg.com/commands/extension-commands/sysret
- https://docs.hyperdbg.com/design/features/vmm-module/design-of-syscall-and-sysret

Features based on Virtual Machine Extensions - VMX



Classic EPT Hook

Classic EPT hook is implemented in !epthook command

Inline EPT Hook

Fast inline EPT hook is implemented in !epthook2 command

Monitor

You can overcome the limitation of hardware debug registers with !monitor command

!epthook command

Description

Puts a hidden breakpoint (0xcc) on the target function in user-mode and kernel-mode without modifying the content of memory in the case of reading/writing.

Features

- Resist on anti-debugging methods related to memory hashing
- Hook without limitation (inline-hooking problems)

Read more

- https://docs.hyperdbg.com/commands/extension-commands/epthook
- https://docs.hyperdbg.com/design/features/vmm-module/design-of-epthook

!epthook2 command

Description

Puts an in-line, detours-style kernel EPT hidden hook.

Features

- Resist on anti-debugging methods related to memory hashing
- Super fast

Read more

- https://docs.hyperdbg.com/commands/extension-commands/epthook2
- https://docs.hyperdbg.com/design/features/vmm-module/design-of-epthook2

!monitor command

Description

Monitors read or write or read/write to a range of addresses. If any read or write on your range address (memory), it will be triggered.

Features

- Without any limitation in size
- Without any limitation in quantity

Read more

- https://docs.hyperdbg.com/commands/extension-commands/monitor
- https://docs.hyperdbg.com/design/features/vmm-module/design-of-monitor

Intercepting Special Instructions

Intercept and modify CPUID

Intercepting and modifying CPUID is possible using !cupid command

Intercept access to performance counter register

Any access to performance counter registers is monitored using !pmc command.

Intercept and modify hypercalls

It's possible to use !vmcall command to monitor hypercalls.

Intercept timing instructions

If any user-mode/kernel-mode application use RDTSC or RDTSCP then it's monitored using !tsc command.

!cpuid command

Description

Triggers when the debugging machine executes a **CPUID** instruction in any level of execution (kernel-mode or user-mode).

Features

- Transparent monitoring CPUID execution

Read more

- https://docs.hyperdbg.com/commands/extension-commands/cpuid

!vmcall command

Description

Triggers when the debugging machine executes **VMCALL** instruction.

Features

- Generate a log from vmcalls

Read more

- https://docs.hyperdbg.com/commands/extension-commands/vmcall

!tsc & !pmc command

Description

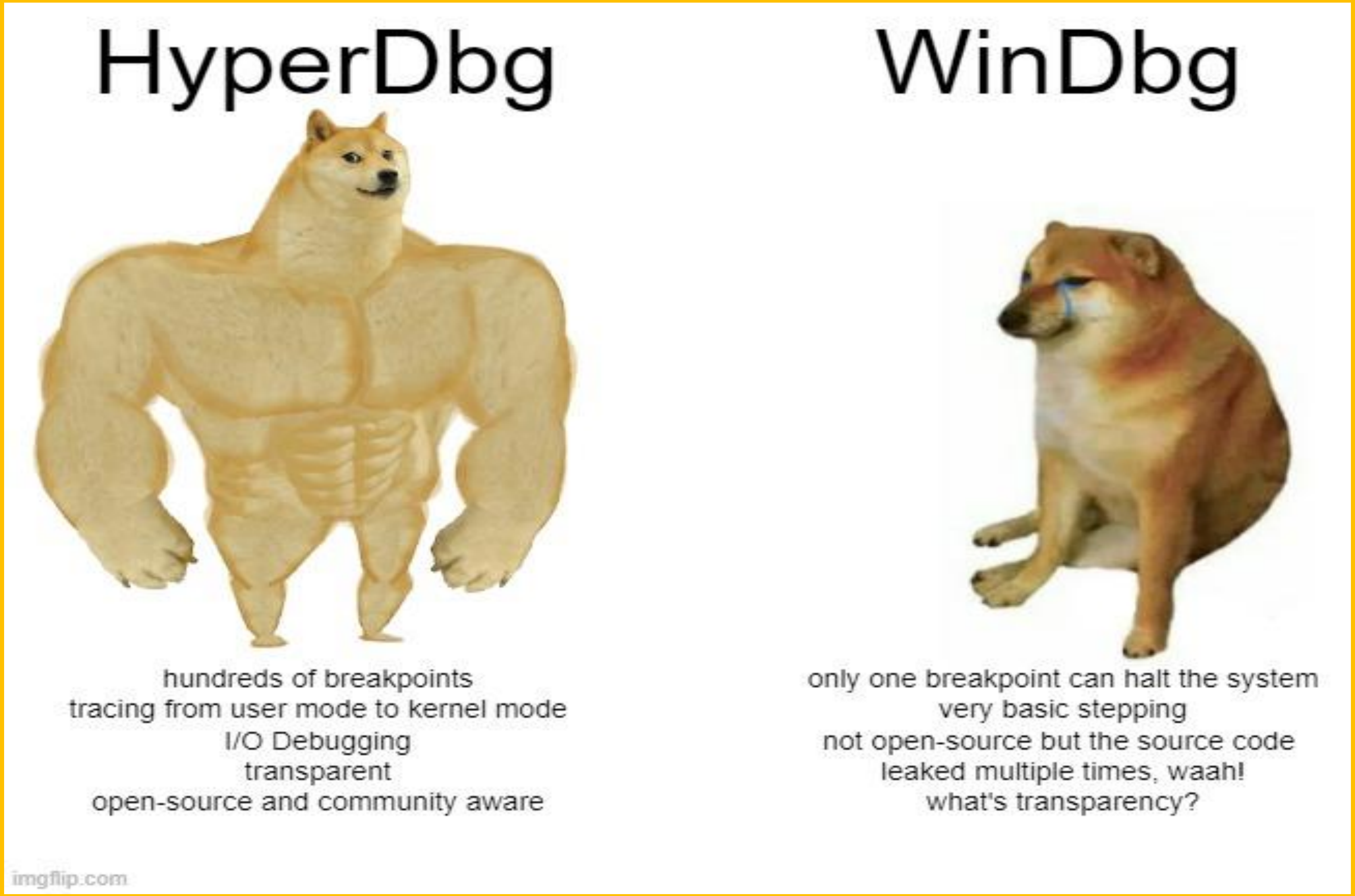
Triggers when the debugging machine executes **RDTSC** or **RDTSCP** instructions in any execution level (kernel-mode or user-mode).

Features

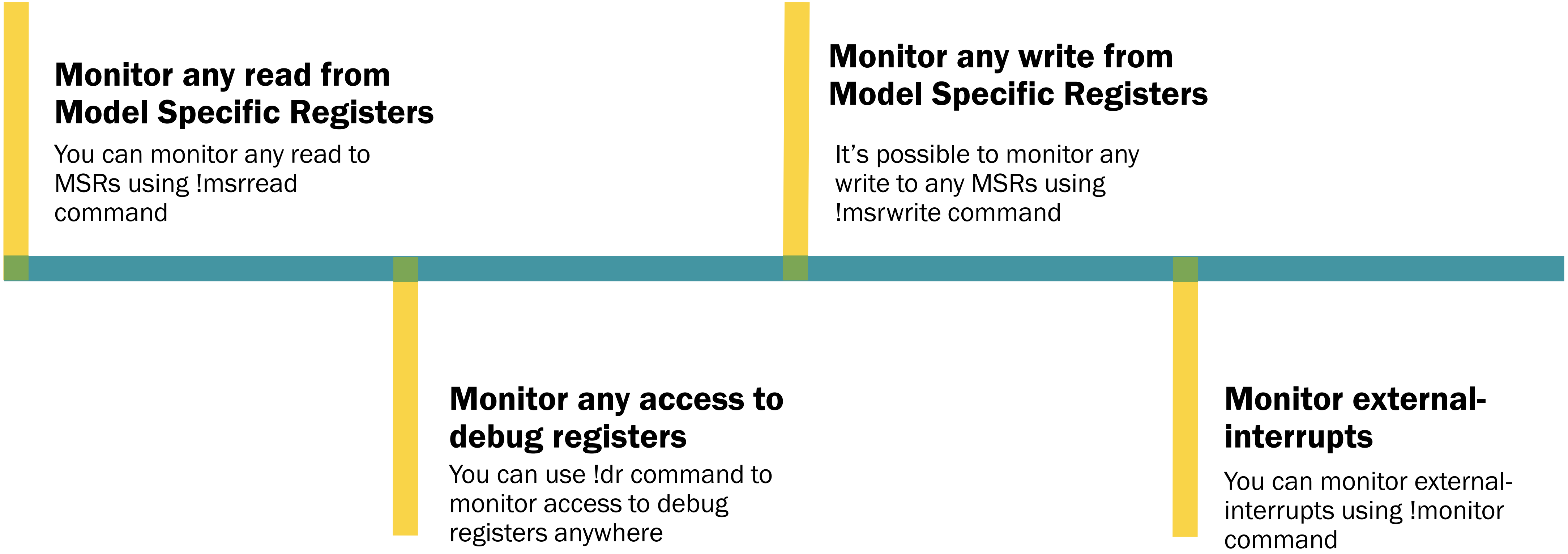
- Monitor performance counter usage
- Monitor rdtsc/rdtscp

Read more

- https://docs.hyperdbg.com/commands/extension-commands/tsc
- https://docs.hyperdbg.com/commands/extension-commands/pmc


HyperDbg
WinDbg
hundreds of breakpoints
tracing from user mode to kernel mode
I/O Debugging
transparent
open-source and community aware
only one breakpoint can halt the system
very basic stepping
not open-source but the source code
leaked multiple times, waah!
what's transparency?
imgflip.com

Monitoring systems’ behavior



Monitor any read from Model Specific Registers

You can monitor any read to MSRs using !msrread command

Monitor any access to debug registers

You can use !dr command to monitor access to debug registers anywhere

Monitor any write from Model Specific Registers

It’s possible to monitor any write to any MSRs using !msrwrite command

Monitor external-interrupts

You can monitor external-interrupts using !monitor command

!msrread & !msrwrite command

Description

Triggers when the debugging machine executes an **RDMSR** instruction or, in other words, when Windows or a driver tries to read a Model-Specific Register (MSR).

Features

- Detects any change using rdmsr and wrmsr

Read more

- https://docs.hyperdbg.com/commands/extension-commands/msrread
- https://docs.hyperdbg.com/commands/extension-commands/msrwrite

!dr command

Description

Triggers, when the debugging machine accesses one of the hardware debug registers.

Features

- Detect access to debug registers e.g., detect anti-debugging methods

Read more

- https://docs.hyperdbg.com/commands/extension-commands/dr

!exception & !interrupt command

Description

Triggers when the debugging machine encounters an exception (**faults, traps, aborts**) or NMI or interrupt. This command applies to only the first 32 entries of IDT (Interrupt Descriptor Table). If you need to hook entries between 32 to 255 of IDT, you should use !interrupt instead.

Features

- Detect all exception before operating system is notified

Read more

- https://docs.hyperdbg.com/commands/extension-commands/exception
- https://docs.hyperdbg.com/commands/extension-commands/interrupt
- https://docs.hyperdbg.com/design/features/vmm-module/design-of-exception-and-interrupt

Monitor and Modify I/O

Monitor I/O Inputs

Monitor any inputs to I/O ports using !ioin command

Monitor I/O Outputs

Monitor any outputs to I/O ports using !ioout command

!ioin & !ioout command

Description

Triggers when the debugging machine executes **IN or IN*** instructions or, in other words, when Windows or a driver tries to use I/O ports.

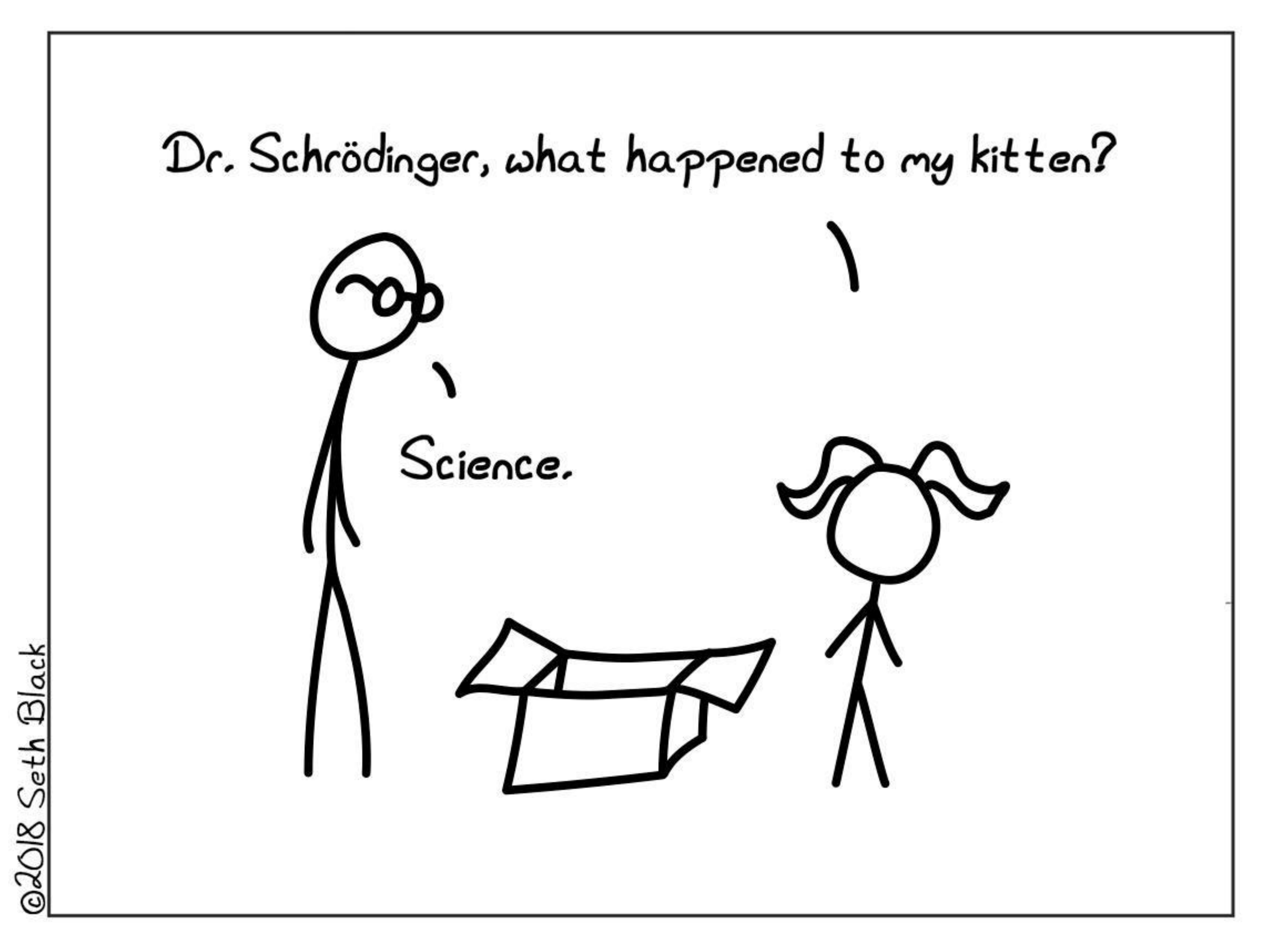
Features

- You can monitor all I/O ports

Read more

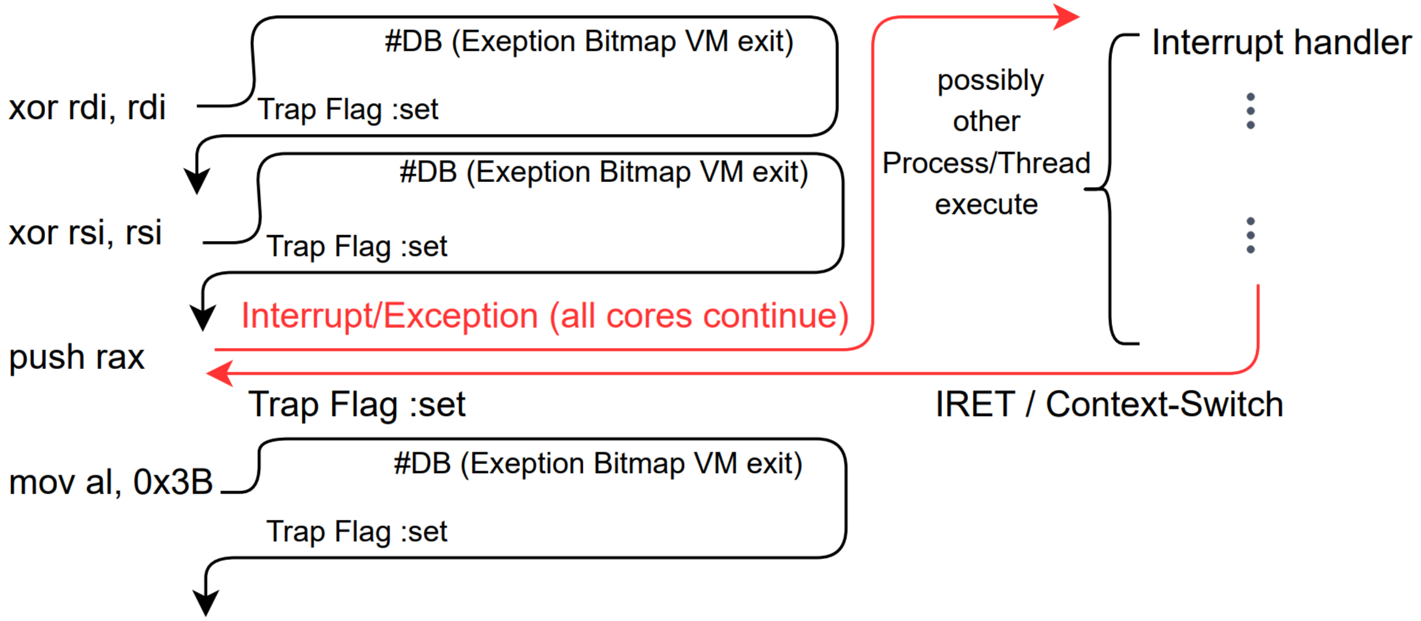
- https://docs.hyperdbg.com/commands/extension-commands/ioin
- https://docs.hyperdbg.com/commands/extension-commands/ioout

Stepping in HyperDbg



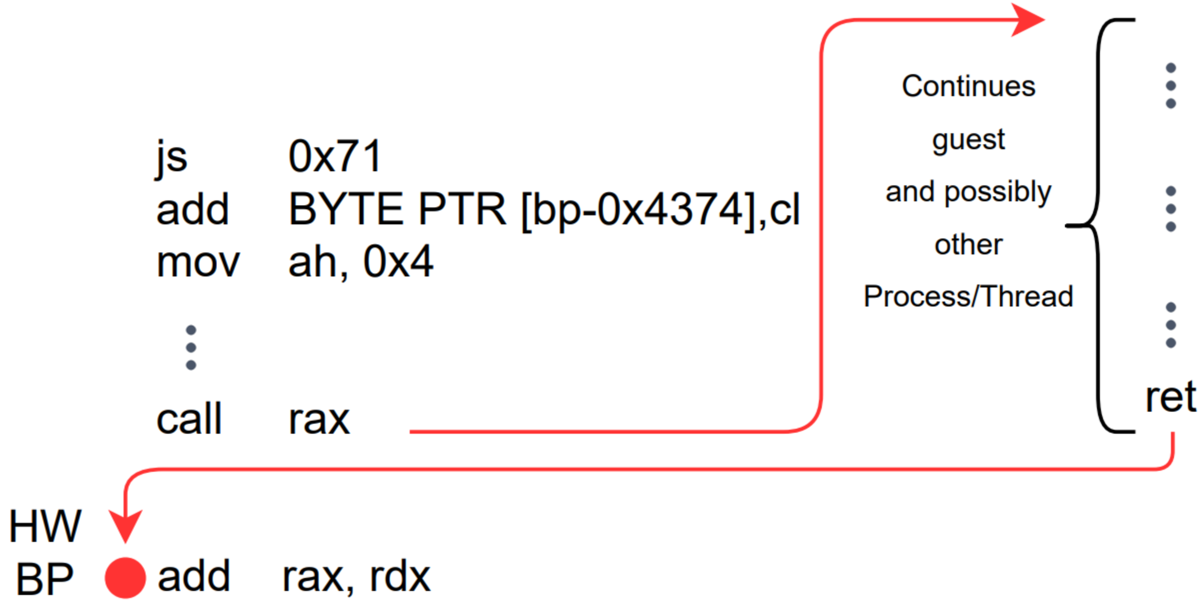
Step-in (t command)

- Like normal debugger but uses hypervisors to intercept events
- All cores are continued
- Trap flag is used


xor rdi, rdi
#DB (Exeption Bitmap VM exit)
Trap Flag :set
xor rsi, rsi
#DB (Exeption Bitmap VM exit)
Trap Flag :set
Interrupt/Exception (all cores continue)
possibly other Process/Thread execute
Interrupt handler
push rax
Trap Flag :set
IRET / Context-Switch
mov al, 0x3B
#DB (Exeption Bitmap VM exit)
Trap Flag :set

Step-over (p command)

- Like normal debugger but uses hypervisors to intercept events
- All cores are continued
- Trap flag is used
- Uses Hardware Debug Registers for calls



```
js      0x71
add     BYTE PTR [bp-0x4374],cl
mov     ah, 0x4
 ⋮
call    rax
```

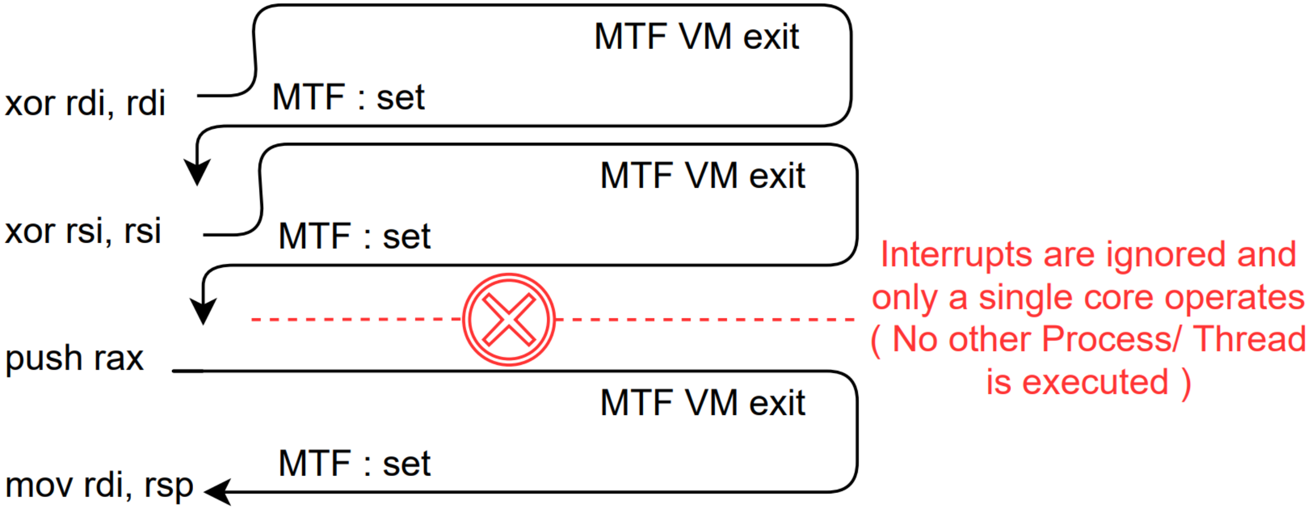
Continues guest and possibly other Process/Thread

```
⋮
⋮
⋮
ret
```

HW BP ● add rax, rdx

Instrument Step-in (i command)

- Only in HyperDbg
- Only the current executing core is continued
- MTF is used
- No interrupts is allowed
- No other threads/processes executes
- Guarantees only the current thread, executes just one instruction



Interrupts are ignored and only a single core operates (No other Process/ Thread is executed)

A
VMX-root Compatible Script Engine

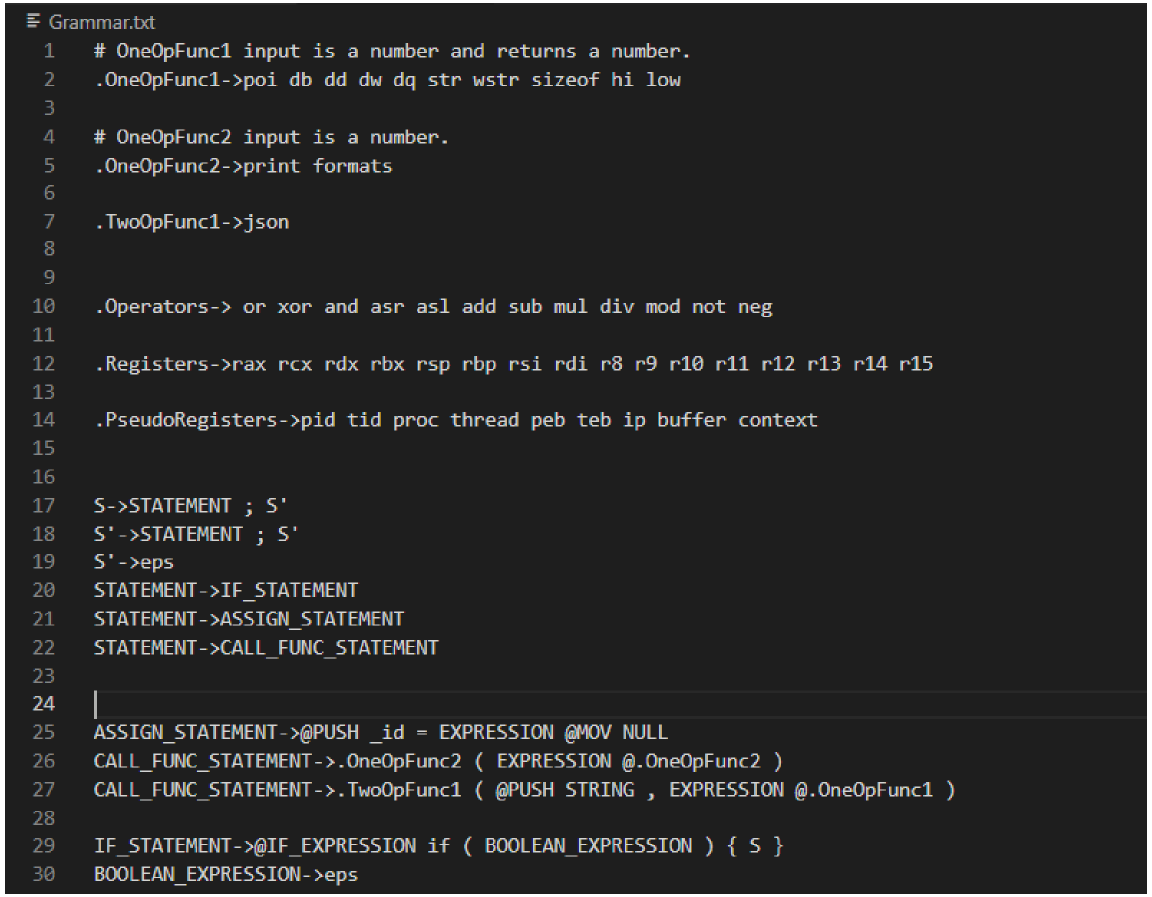
Our Powerful Script Engine

- HyperDbg's script engine is designed to work on vmx-root
- A MASM-Style language, combined with C keywords and features **(if, else, for, etc.)**
- We designed everything from scratch like basic operating system spinlock, memory check, even functions like **sprintf** and **strlen**.
- There is a term called "unsafe behavior" in HyperDbg.

  Read more : https://docs.hyperdbg.com/tips-and-tricks/considerations/the-unsafe-behavior
- RFLAGS.IF bit is cleared ! No interrupt ! No page-fault (#PF).

Our Powerful Script Engine

- LL(1) and LALR(1) parsers are used to reach the most possible performance
- Grammar of Script Engine can be customized



```
# OneOpFunc1 input is a number and returns a number.
.OneOpFunc1->poi db dd dw dq str wstr sizeof hi low

# OneOpFunc2 input is a number.
.OneOpFunc2->print formats

.TwoOpFunc1->json


.Operators-> or xor and asr asl add sub mul div mod not neg

.Registers->rax rcx rdx rbx rsp rbp rsi rdi r8 r9 r10 r11 r12 r13 r14 r15

.PseudoRegisters->pid tid proc thread peb teb ip buffer context


S->STATEMENT ; S'
S'->STATEMENT ; S'
S'->eps
STATEMENT->IF_STATEMENT
STATEMENT->ASSIGN_STATEMENT
STATEMENT->CALL_FUNC_STATEMENT


ASSIGN_STATEMENT->@PUSH _id = EXPRESSION @MOV NULL
CALL_FUNC_STATEMENT->.OneOpFunc2 ( EXPRESSION @.OneOpFunc2 )
CALL_FUNC_STATEMENT->.TwoOpFunc1 ( @PUSH STRING , EXPRESSION @.OneOpFunc1 )

IF_STATEMENT->@IF_EXPRESSION if ( BOOLEAN_EXPRESSION ) { S }
BOOLEAN_EXPRESSION->eps
```

Keywords

Keyword	Description
poi	Pointer-sized data from the specified address.
hi	High 16 bits
low	Low 16 bits
db	Low 8 bits
dd	Low 16 bits
dw	Low 32 bits
dq	64 bits
sizeof	Size of the target variable
not	Flip each and every bit
neg	True/False logic flipping

Pre-defined Functions

Function	Description
Print	Print the result of an expression.
Printf	Print the result like classic **printf**.
Pause	Halt the system and give control to the debugger.
EnableEvent	Enable an event.
DisableEvent	Disable an event.

Pseudo-registers

Pseudo-register	Description
$pid	The process ID (PID) of the current process.
$proc	The address of the current process (that is, the address of theEPROCESS block).
$tid	The thread ID for the current thread.
$thread	The address of the current thread. In kernel-mode debugging, this address is the address of the ETHREAD block.
$peb	The address of the process environment block (PEB) of the current process.
$teb	The address of the thread environment block (TEB) of the current thread.
$ip	The instruction pointer register (rip).
$buffer	The pre-allocated buffer if the user requests a safe buffer.
$context	The context of the triggered event (It has a different meaning in each event).

Challenges: User requests an invalid address

What if the user entered an invalid address?

- CPU never knows whether an address is valid or invalid unless it access the address.
- #PF are disabled in vmx-root mode (RFLAGS.IF Cleared).
- If we access an invalid address in user-mode, then the program crashes, one way to avoid these crashes is to use try { } catch { } which uses Windows SEH mechanism.
- Using SEH is a bottleneck as it is SLOW.
- If we access an invalid address in kernel-mode then a BSOD happens.
- If we access an invalid address in vmx-root mode then system halts !

TSX and page-table traversing to rescue

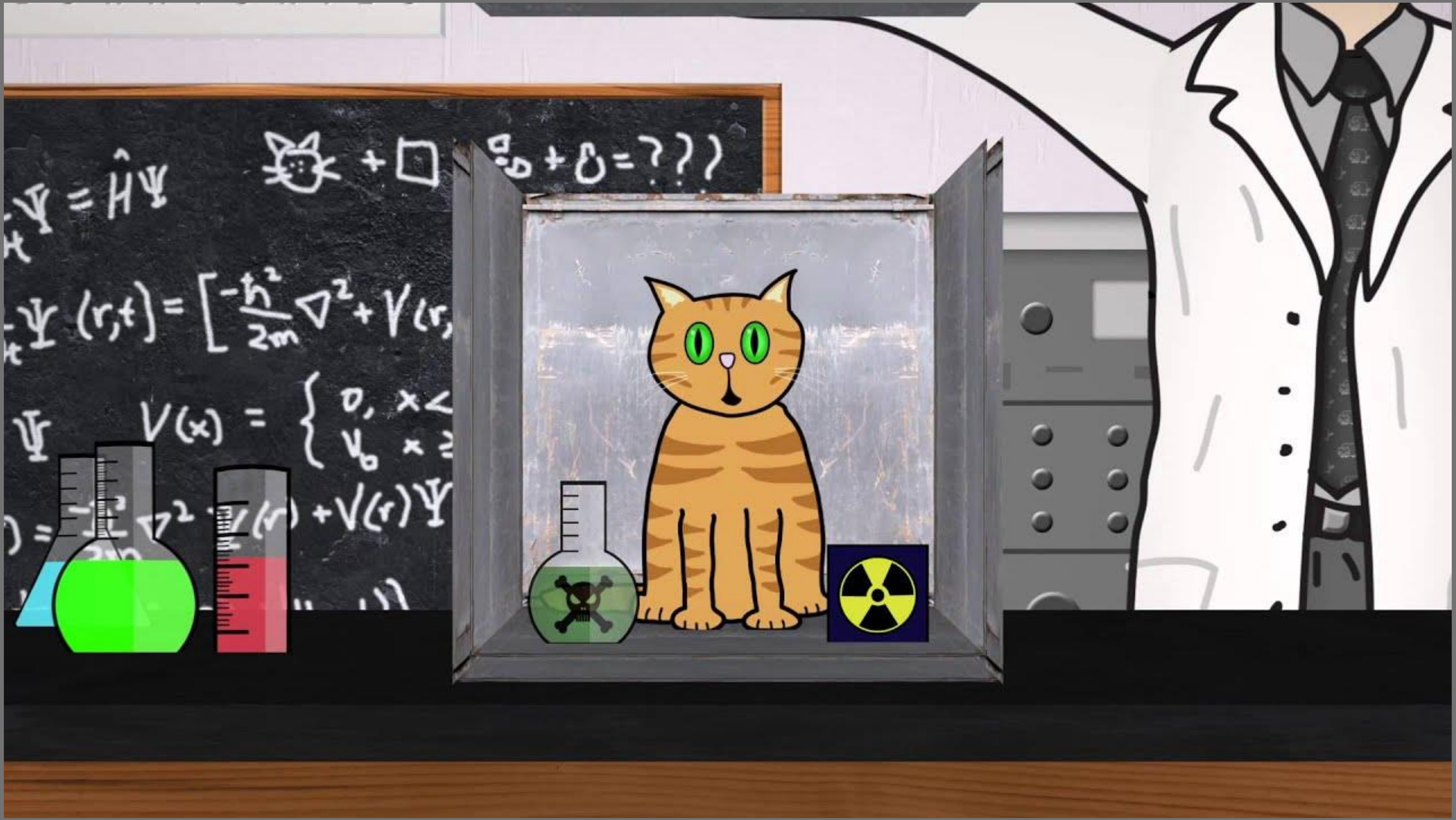
- Current version of script-engine operates in kernel-mode and vmx-root mode.
- First, we check whether the target system supports Intel Transactional Synchronization Extensions.
- If it supports TSX (RTM) then we create a transaction by using **xbegin… xend.**
- If the transaction failed then it shows that the address was invalid and if it is successful then it shows that the address is valid.

TSX and page-table traversing to rescue

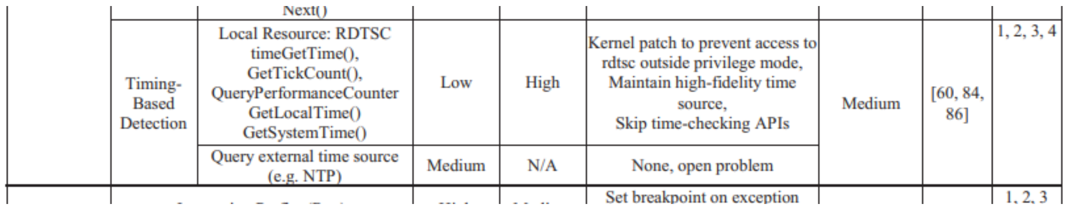
- If the target system didn’t support TSX, then, we traverse each page-table (pml4 → pdpt → pd → pt).
- If the page address was valid and was PRESENT, then the address is valid; otherwise, it’s invalid.
- Using TSX is super fast and using the above methods we solved the problem of accessing invalid addresses in script engine by adding a check before accessing the address.

HYPERDBG
LIGHTYEAR
SPACE RANGER
HYPERDBG EVERYWHERE!
imgflip.com

Transparency



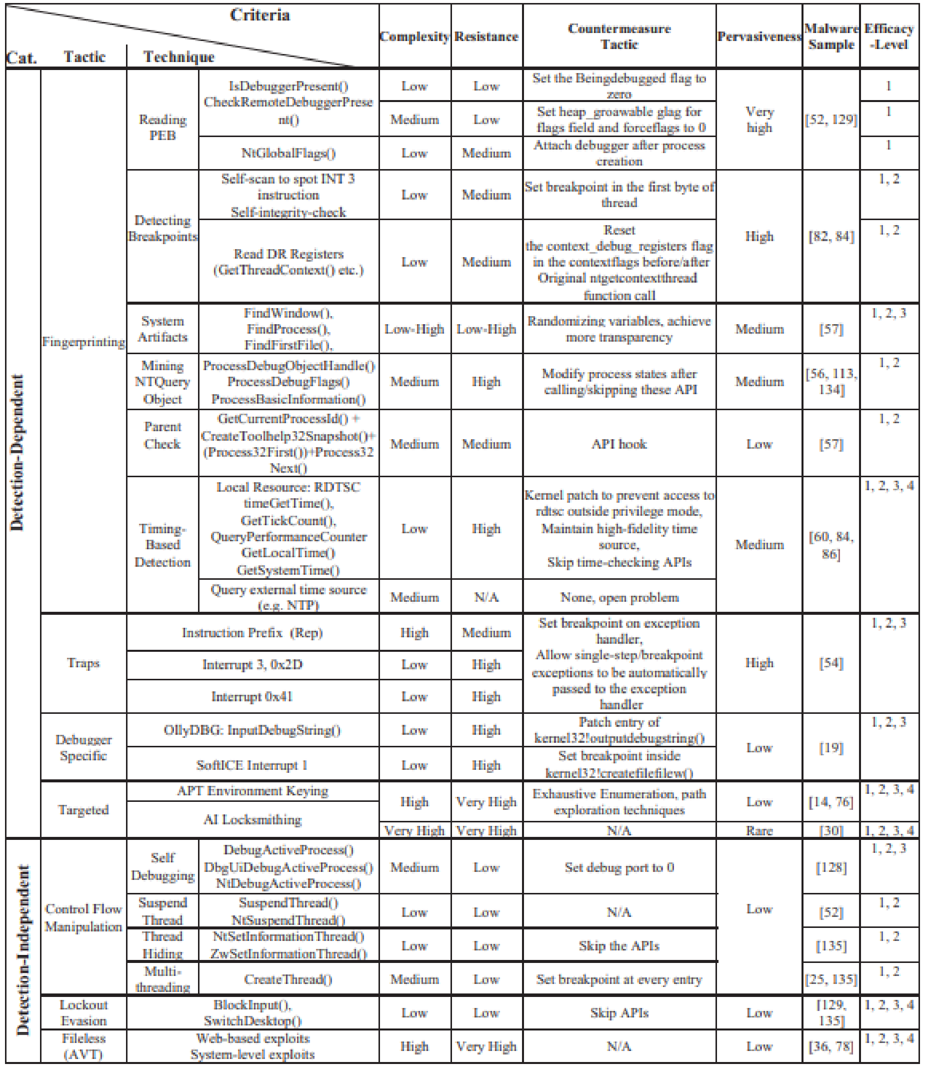
Anti-Malware and Anti-Debugging



	Next()						
Timing-Based Detection	Local Resource: RDTSC timeGetTime(), GetTickCount(), QueryPerformanceCounter GetLocalTime() GetSystemTime()	Low	High	Kernel patch to prevent access to rdtsc outside privilege mode, Maintain high-fidelity time source, Skip time-checking APIs	Medium	[60, 84, 86]	1, 2, 3, 4
	Query external time source (e.g. NTP)	Medium	N/A	None, open problem			
				Set breakpoint on exception			1, 2, 3

Afianian, Amir, et al. "Malware dynamic analysis evasion techniques: A survey." ACM Computing Surveys (CSUR) 52.6 (2019): 1-28.

Table 1. Classification and Comparison of Malware Anti-Debugging Techniques



Cat.	Tactic	Technique		Complexity	Resistance	Countermeasure Tactic	Pervasiveness	Malware Sample	Efficacy -Level

Detection-Dependent	Fingerprinting	Reading PEB	IsDebuggerPresent()
			CheckRemoteDebuggerPresent()
				Low	Low	Set the Beingdebugged flag to zero	Very high	[52, 129]	1

				Medium	Low	Set heap_groawable glag for flags field and forceflags to 0			1
			NtGlobalFlags()	Low	Medium	Attach debugger after process creation			1
		Detecting Breakpoints	Self-scan to spot INT 3 instruction Self-integrity-check	Low	Medium	Set breakpoint in the first byte of thread	High	[82, 84]	1, 2
			Read DR Registers (GetThreadContext() etc.)	Low	Medium	Reset the context_debug_registers flag in the contextflags before/after Original ntgetcontextthread function call			1, 2
		System Artifacts	FindWindow(), FindProcess(), FindFirstFile(),	Low-High	Low-High	Randomizing variables, achieve more transparency	Medium	[57]	1, 2, 3

		Mining NTQuery Object	ProcessDebugObjectHandle()
			ProcessDebugFlags()
			ProcessBasicInformation()
				Medium	High	Modify process states after calling/skipping these API	Medium	[56, 113, 134]	1, 2

		Parent Check	GetCurrentProcessId() + CreateToolhelp32Snapshot()+ (Process32First())+Process32Next()	Medium	Medium	API hook	Low	[57]	1, 2
		Timing-Based Detection	Local Resource: RDTSC timeGetTime(), GetTickCount(), QueryPerformanceCounter GetLocalTime() GetSystemTime()	Low	High	Kernel patch to prevent access to rdtsc outside privilege mode, Maintain high-fidelity time source, Skip time-checking APIs	Medium	[60, 84, 86]	1, 2, 3, 4
			Query external time source (e.g. NTP)	Medium	N/A	None, open problem			
	Traps	Instruction Prefix (Rep)		High	Medium	Set breakpoint on exception handler, Allow single-step/breakpoint exceptions to be automatically passed to the exception handler	High	[54]	1, 2, 3
		Interrupt 3, 0x2D		Low	High				
		Interrupt 0x41		Low	High				
	Debugger Specific	OllyDBG: InputDebugString()		Low	High	Patch entry of kernel32!outputdebugstring()	Low	[19]	1, 2, 3
		SoftICE Interrupt 1		Low	High	Set breakpoint inside kernel32!createfilefilew()			
	Targeted	APT Environment Keying		High	Very High	Exhaustive Enumeration, path exploration techniques	Low	[14, 76]	1, 2, 3, 4
		AI Locksmithing							
				Very High	Very High	N/A	Rare	[30]	1, 2, 3, 4

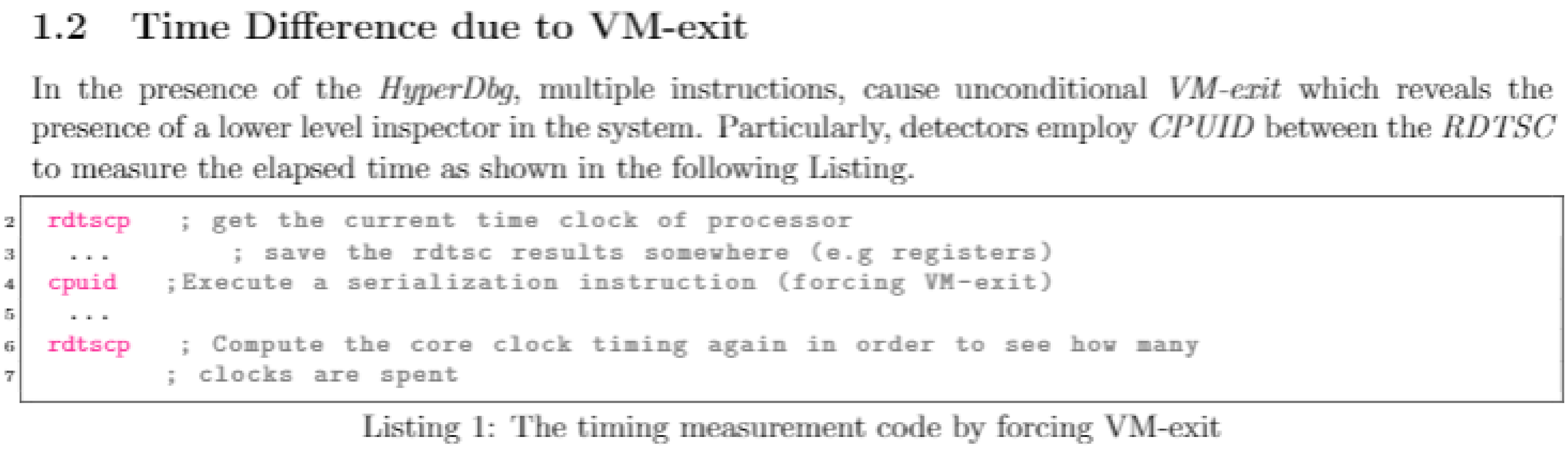
Detection-Independent	Control Flow Manipulation	Self Debugging	DebugActiveProcess()
			DbgUiDebugActiveProcess()
			NtDebugActiveProcess()
				Medium	Low	Set debug port to 0	Low	[128]	1, 2, 3

		Suspend Thread	SuspendThread()
			NtSuspendThread()
				Low	Low	N/A		[52]	1, 2

		Thread Hiding	NtSetInformationThread()
			ZwSetInformationThread()
				Low	Low	Skip the APIs		[135]	1, 2

		Multi-threading	CreateThread()	Medium	Low	Set breakpoint at every entry		[25, 135]	1, 2
	Lockout Evasion	BlockInput(), SwitchDesktop()		Low	Low	Skip APIs	Low	[129, 135]	1, 2, 3, 4
	Fileless (AVT)	Web-based exploits System-level exploits		High	Very High	N/A	Low	[36, 78]	1, 2, 3, 4

Timestamp Check



1.2 Time Difference due to VM-exit

In the presence of the *HyperDbg*, multiple instructions, cause unconditional *VM-exit* which reveals the presence of a lower level inspector in the system. Particularly, detectors employ *CPUID* between the *RDTSC* to measure the elapsed time as shown in the following Listing.

```
rdtscp    ; get the current time clock of processor
 ...          ; save the rdtsc results somewhere (e.g registers)
cpuid    ;Execute a serialization instruction (forcing VM-exit)
 ...
rdtscp    ; Compute the core clock timing again in order to see how many
         ; clocks are spent
```

Listing 1: The timing measurement code by forcing VM-exit

CPUID without HyperDbg

10,000 instances
Follow Gaussian Curve by Interpolation

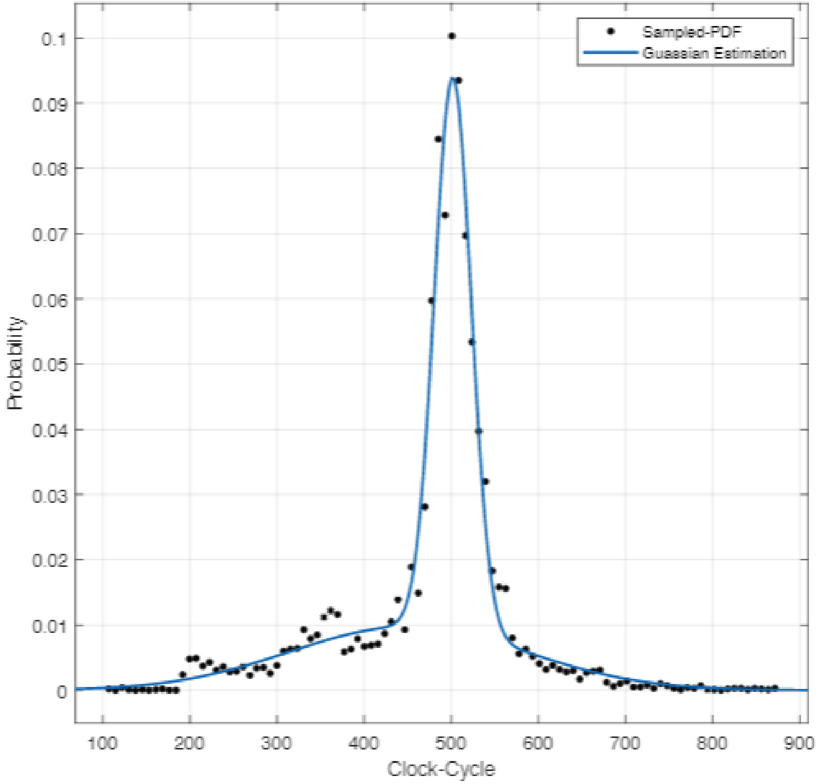


Figure 1: PDF distribution and sampled data of timing measurement without activated HyperDbg

CPUID with HyperDbg

10,000 instances
Follow Gaussian Curve by Interpolation

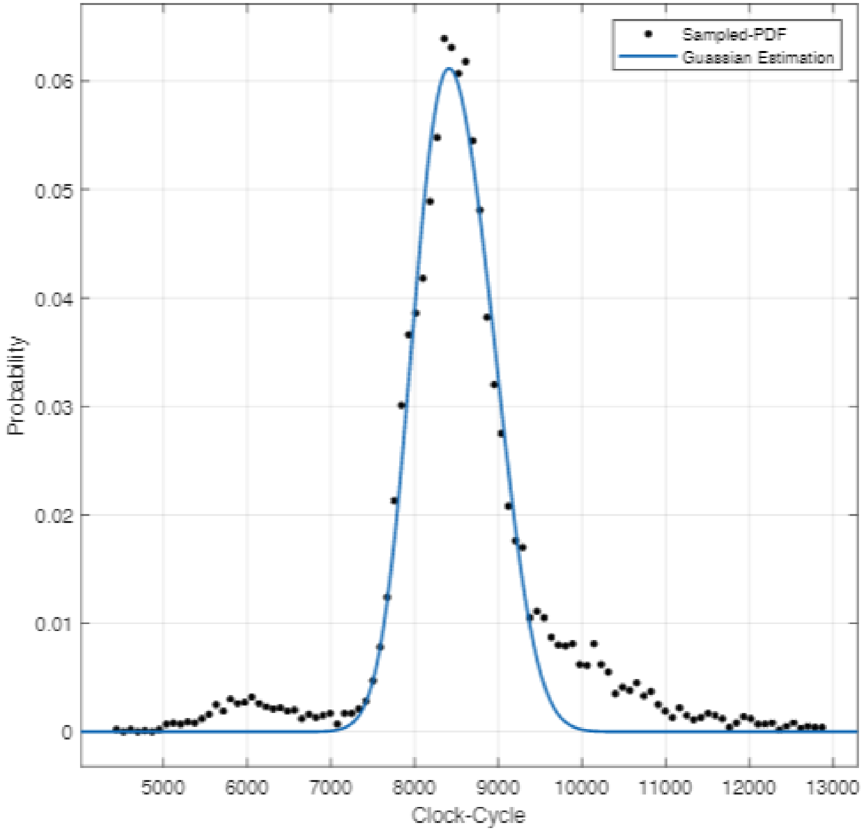


Figure 2: PDF distribution and sampled data of timing measurement with activated HyperDbg

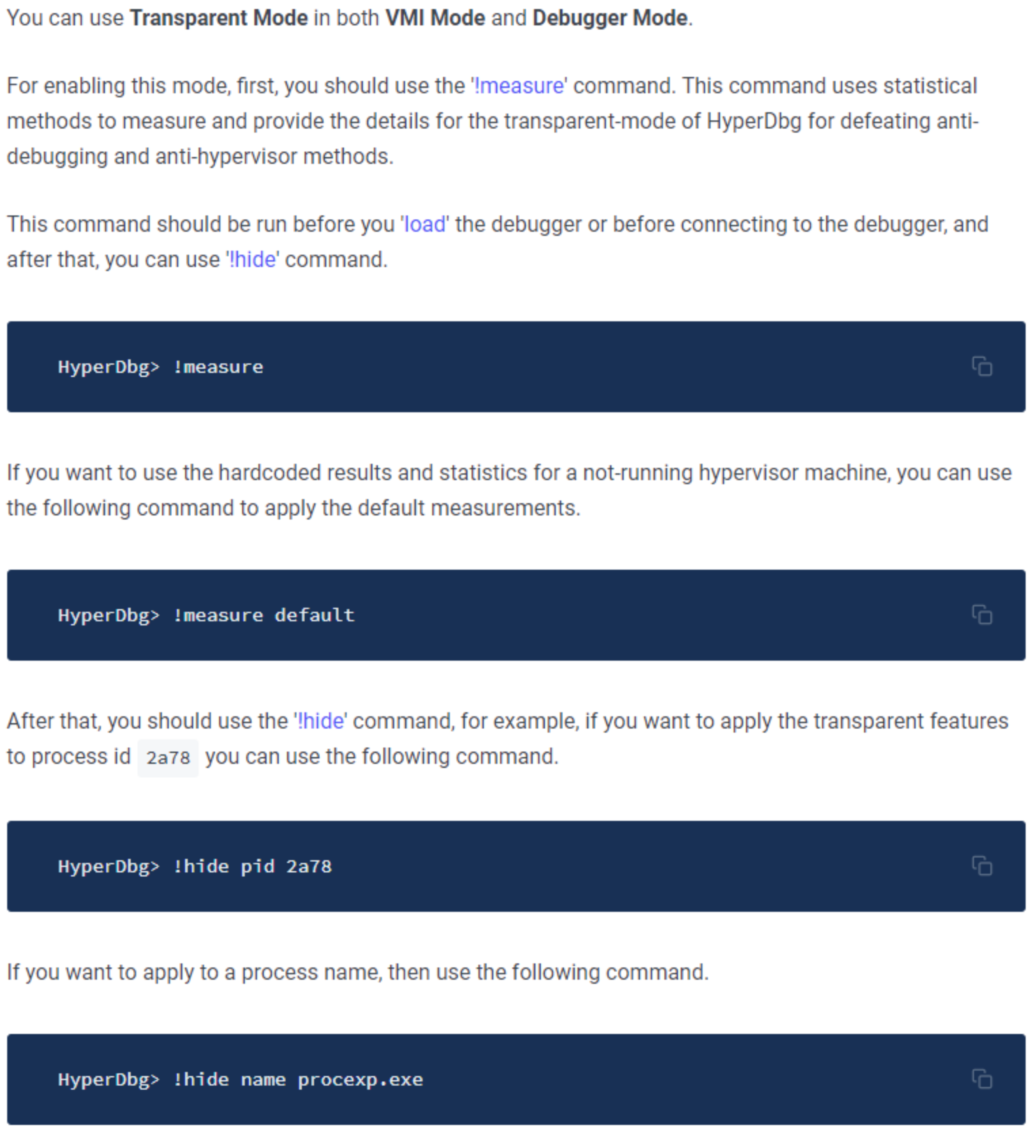
Automate the measurement Procedure

!measure

!measure default

!hide pid 2487

!hide name proexp.exe



You can use **Transparent Mode** in both **VMI Mode** and **Debugger Mode**.

For enabling this mode, first, you should use the '!measure' command. This command uses statistical methods to measure and provide the details for the transparent-mode of HyperDbg for defeating anti-debugging and anti-hypervisor methods.

This command should be run before you 'load' the debugger or before connecting to the debugger, and after that, you can use '!hide' command.

```
HyperDbg> !measure
```

If you want to use the hardcoded results and statistics for a not-running hypervisor machine, you can use the following command to apply the default measurements.

```
HyperDbg> !measure default
```

After that, you should use the '!hide' command, for example, if you want to apply the transparent features to process id `2a78` you can use the following command.

```
HyperDbg> !hide pid 2a78
```

If you want to apply to a process name, then use the following command.

```
HyperDbg> !hide name procexp.exe
```

Procedure Diagram

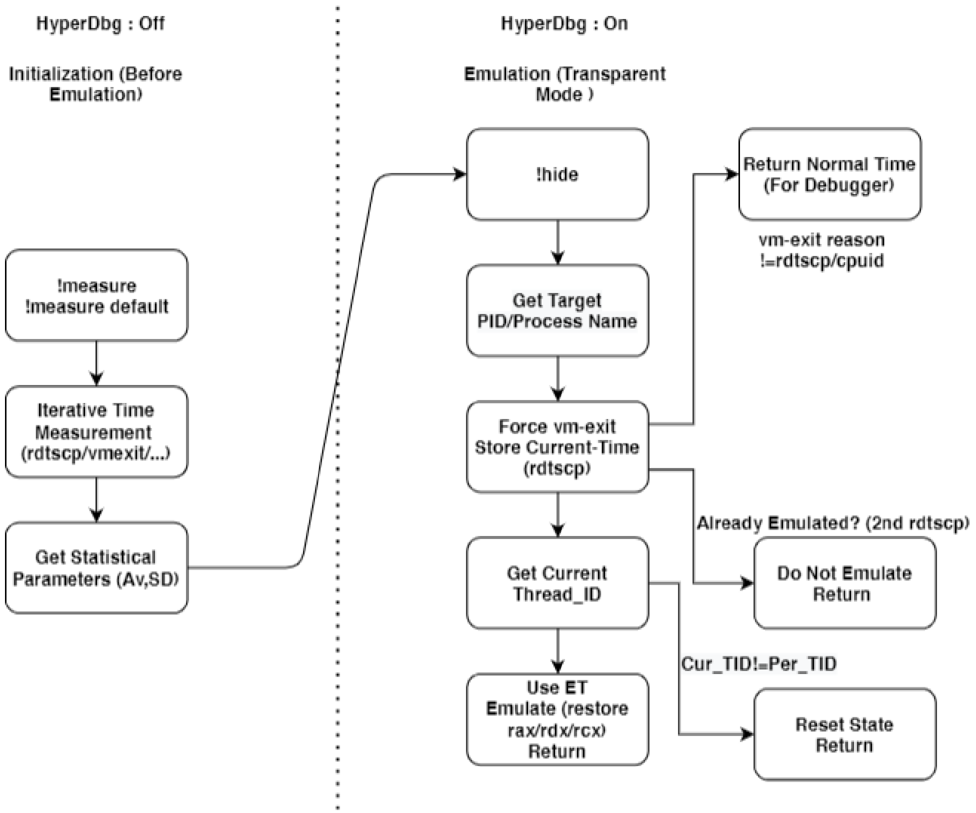
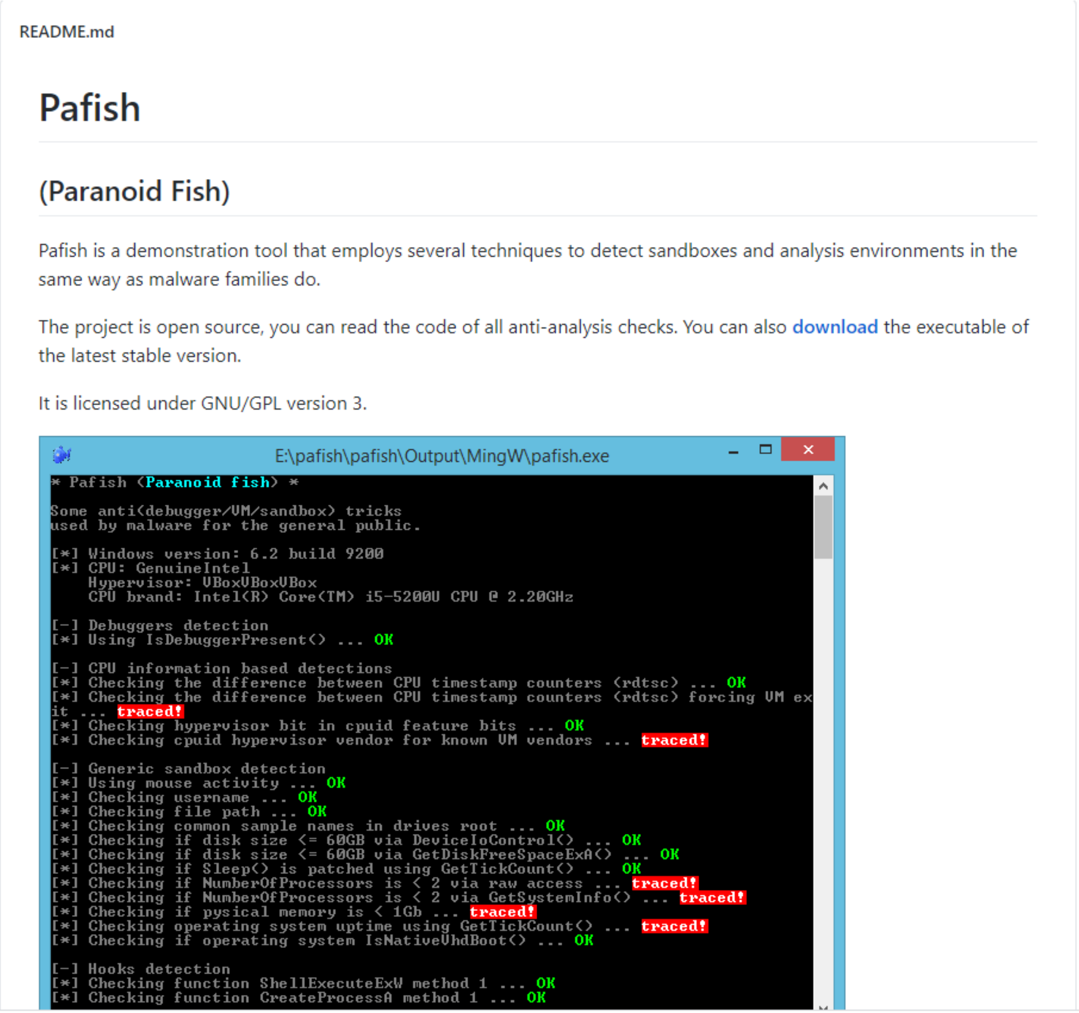

HyperDbg : Off
Initialization (Before Emulation)
!measure
!measure default
Iterative Time Measurement (rdtscp/vmexit/...)
Get Statistical Parameters (Av,SD)
HyperDbg : On
Emulation (Transparent Mode)
!hide
Get Target PID/Process Name
Force vm-exit Store Current-Time (rdtscp)
Get Current Thread_ID
Use ET Emulate (restore rax/rdx/rcx) Return
Return Normal Time (For Debugger)
vm-exit reason !=rdtscp/cpuid
Already Emulated? (2nd rdtscp)
Do Not Emulate Return
Cur_TID!=Per_TID
Reset State Return

Figure 4: State Diagram Process of *rdtsc/rdtscp* emulation by *HyperDbg*

Evaluation on Pafish



README.md

Pafish

(Paranoid Fish)

Pafish is a demonstration tool that employs several techniques to detect sandboxes and analysis environments in the same way as malware families do.

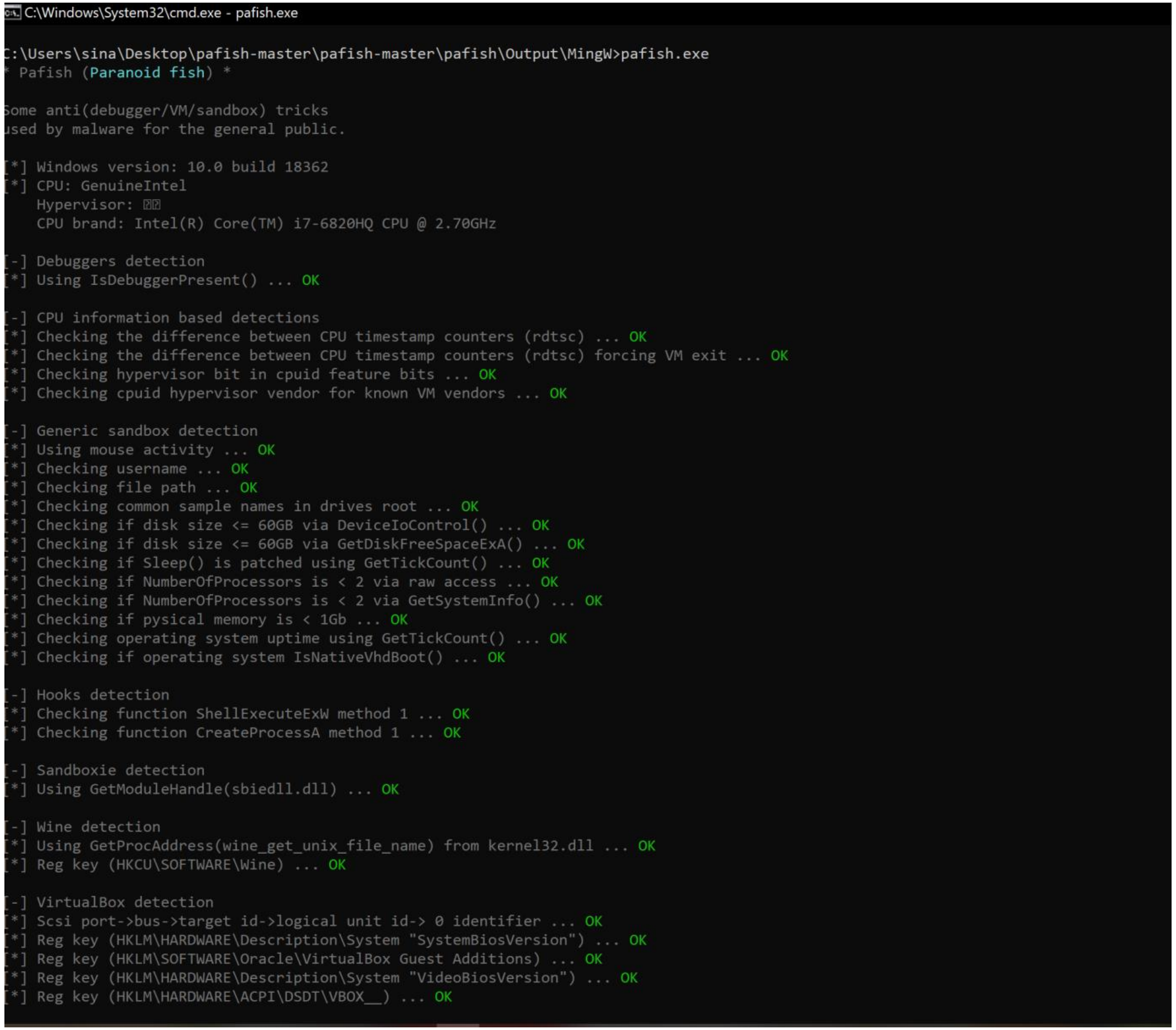
The project is open source, you can read the code of all anti-analysis checks. You can also download the executable of the latest stable version.

It is licensed under GNU/GPL version 3.

E:\pafish\pafish\Output\MingW\pafish.exe

```
* Pafish (Paranoid fish) *

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

[*] Windows version: 6.2 build 9200
[*] CPU: GenuineIntel
    Hypervisor: VBoxVBoxVBox
    CPU brand: Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz

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

[-] CPU information based detections
[*] Checking the difference between CPU timestamp counters (rdtsc) ... OK
[*] Checking the difference between CPU timestamp counters (rdtsc) forcing VM ex
it ... traced!
[*] Checking hypervisor bit in cpuid feature bits ... OK
[*] Checking cpuid hypervisor vendor for known VM vendors ... traced!

[-] Generic sandbox detection
[*] Using mouse activity ... OK
[*] Checking username ... OK
[*] Checking file path ... OK
[*] Checking common sample names in drives root ... OK
[*] Checking if disk size <= 60GB via DeviceIoControl() ... OK
[*] Checking if disk size <= 60GB via GetDiskFreeSpaceExA() ... OK
[*] Checking if Sleep() is patched using GetTickCount() ... OK
[*] Checking if NumberOfProcessors is < 2 via raw access ... traced!
[*] Checking if NumberOfProcessors is < 2 via GetSystemInfo() ... traced!
[*] Checking if pysical memory is < 1Gb ... traced!
[*] Checking operating system uptime using GetTickCount() ... traced!
[*] Checking if operating system IsNativeVhdBoot() ... OK

[-] Hooks detection
[*] Checking function ShellExecuteExW method 1 ... OK
[*] Checking function CreateProcessA method 1 ... OK
```

Demo Time

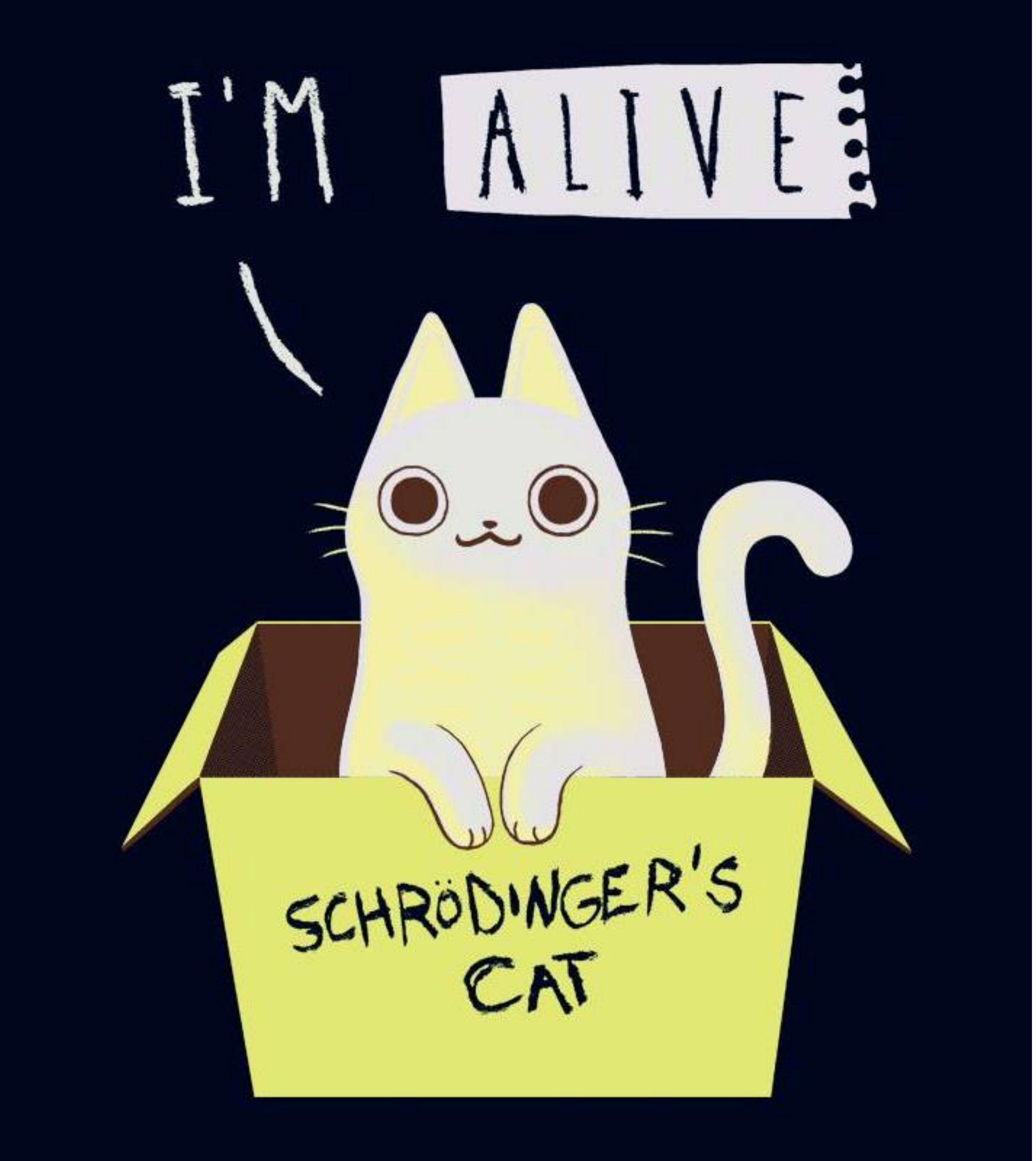


C:\Windows\System32\cmd.exe - pafish.exe

```
C:\Users\sina\Desktop\pafish-master\pafish-master\pafish\Output\MingW>pafish.exe
* Pafish (Paranoid fish) *

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

[*] Windows version: 10.0 build 18362
[*] CPU: GenuineIntel
    Hypervisor: ⍰⍰
    CPU brand: Intel(R) Core(TM) i7-6820HQ CPU @ 2.70GHz

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

[-] CPU information based detections
[*] Checking the difference between CPU timestamp counters (rdtsc) ... OK
[*] Checking the difference between CPU timestamp counters (rdtsc) forcing VM exit ... OK
[*] Checking hypervisor bit in cpuid feature bits ... OK
[*] Checking cpuid hypervisor vendor for known VM vendors ... OK

[-] Generic sandbox detection
[*] Using mouse activity ... OK
[*] Checking username ... OK
[*] Checking file path ... OK
[*] Checking common sample names in drives root ... OK
[*] Checking if disk size <= 60GB via DeviceIoControl() ... OK
[*] Checking if disk size <= 60GB via GetDiskFreeSpaceExA() ... OK
[*] Checking if Sleep() is patched using GetTickCount() ... OK
[*] Checking if NumberOfProcessors is < 2 via raw access ... OK
[*] Checking if NumberOfProcessors is < 2 via GetSystemInfo() ... OK
[*] Checking if pysical memory is < 1Gb ... OK
[*] Checking operating system uptime using GetTickCount() ... OK
[*] Checking if operating system IsNativeVhdBoot() ... OK

[-] Hooks detection
[*] Checking function ShellExecuteExW method 1 ... OK
[*] Checking function CreateProcessA method 1 ... OK

[-] Sandboxie detection
[*] Using GetModuleHandle(sbiedll.dll) ... OK

[-] Wine detection
[*] Using GetProcAddress(wine_get_unix_file_name) from kernel32.dll ... OK
[*] Reg key (HKCU\SOFTWARE\Wine) ... OK

[-] VirtualBox detection
[*] Scsi port->bus->target id->logical unit id-> 0 identifier ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "SystemBiosVersion") ... OK
[*] Reg key (HKLM\SOFTWARE\Oracle\VirtualBox Guest Additions) ... OK
[*] Reg key (HKLM\HARDWARE\Description\System "VideoBiosVersion") ... OK
[*] Reg key (HKLM\HARDWARE\ACPI\DSDT\VBOX__) ... OK
```

REVERSE ENGINEER
WINDBG
HYPERDBG
imgflip.com

Any Questions?



Thank you