Tricky Sample ? Hack it easy! Applying dynamic binary instrumentation to light-weight malware behavior analysis

Maksim Shudrak

About Me

Interests

Vulnerabilities Hunting
Fuzzing
Reverse-engineering
Malware Analysis
Dynamic Binary Instrumentation

BIO

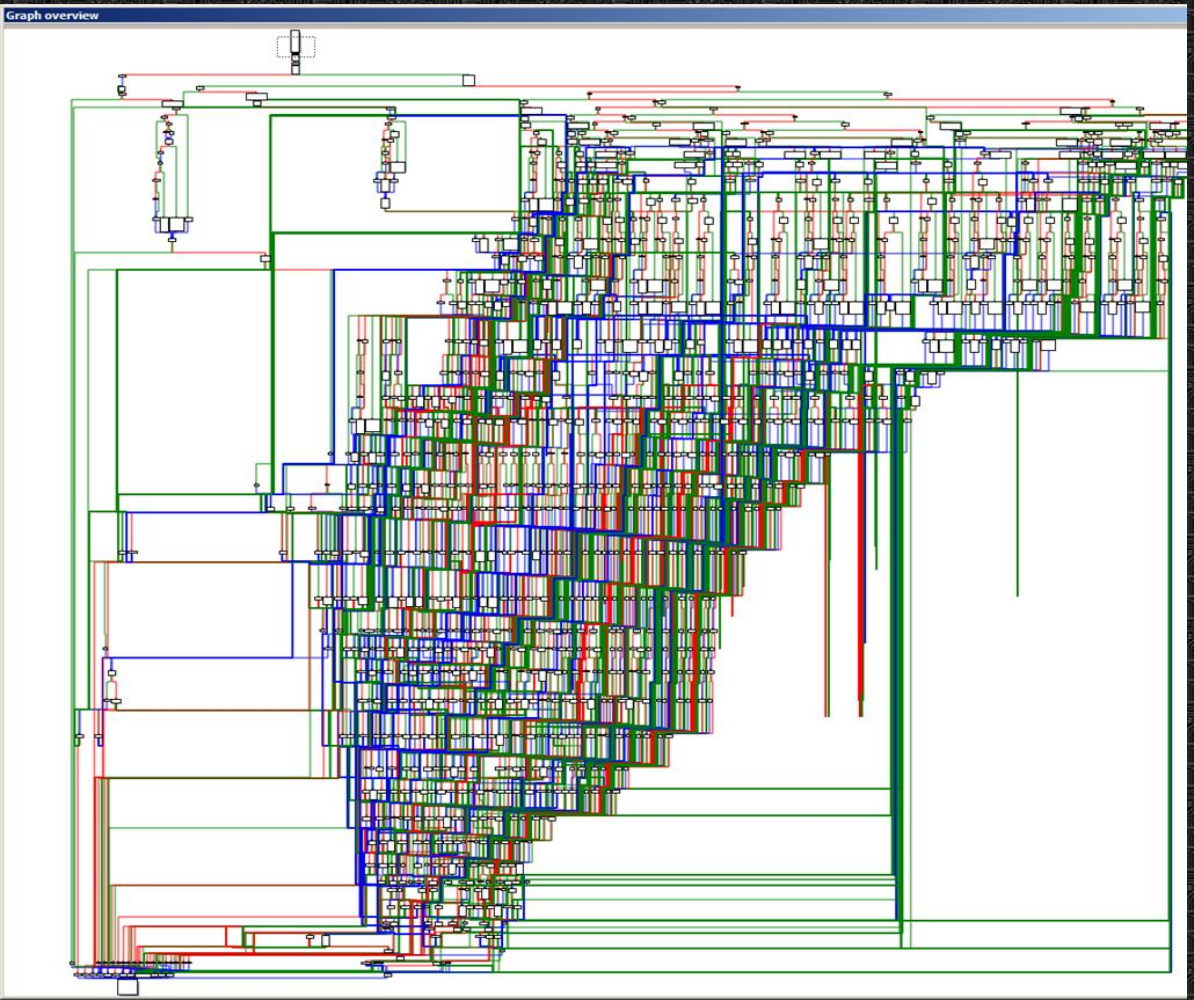
2018 – present: Senior Offensive Security Researcher
2016: Defended PhD (Vulns Hunting) in Tomsk, Russia
2015-2017: Researcher, IBM Research, Haifa, Israel
2011-2015: Security Researcher, PhD student

Projects

Drltrace – transparent API-calls tracing for malware analysis
https://github.com/mxmssh/drltrace
WinHeap Explorer – PoC for heap-based bugs detection in x86 code
https://github.com/WinHeapExplorer/WinHeap-Explorer
IDAMetrics – IDA plugin for machine code complexity assessment
https://github.com/mxmssh/IDAmetrics

Outline

- Why Dynamic Analysis?
- Current approaches
  - Runtime Overhead vs Visibility
- Dynamic Binary Instrumentation
  - Technique Overview
  - DBI Frameworks Comparison
- DrLtrace
  - How Does It Work ?
  - Usage
- Examples & Demo



Why dynamic ?

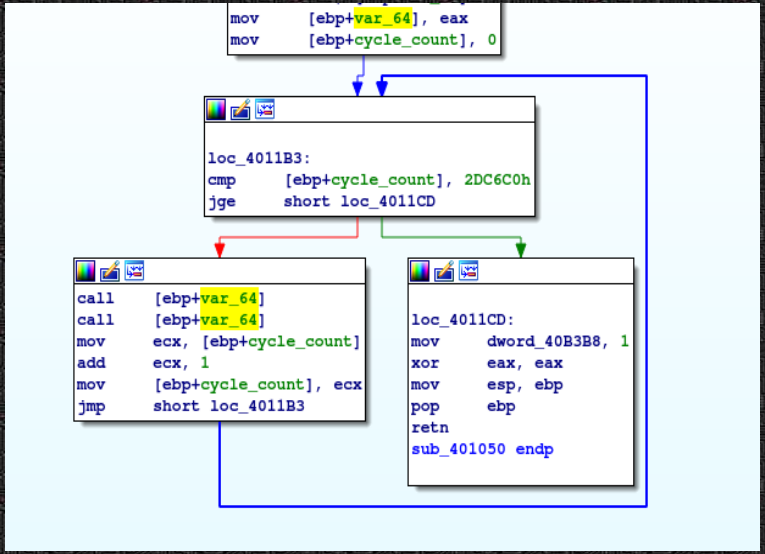
- Obfuscated & packed code hard for static analysis.
- In some cases, we need only a high-level view on malware behavior.

Current Situation in Dynamic Analysis

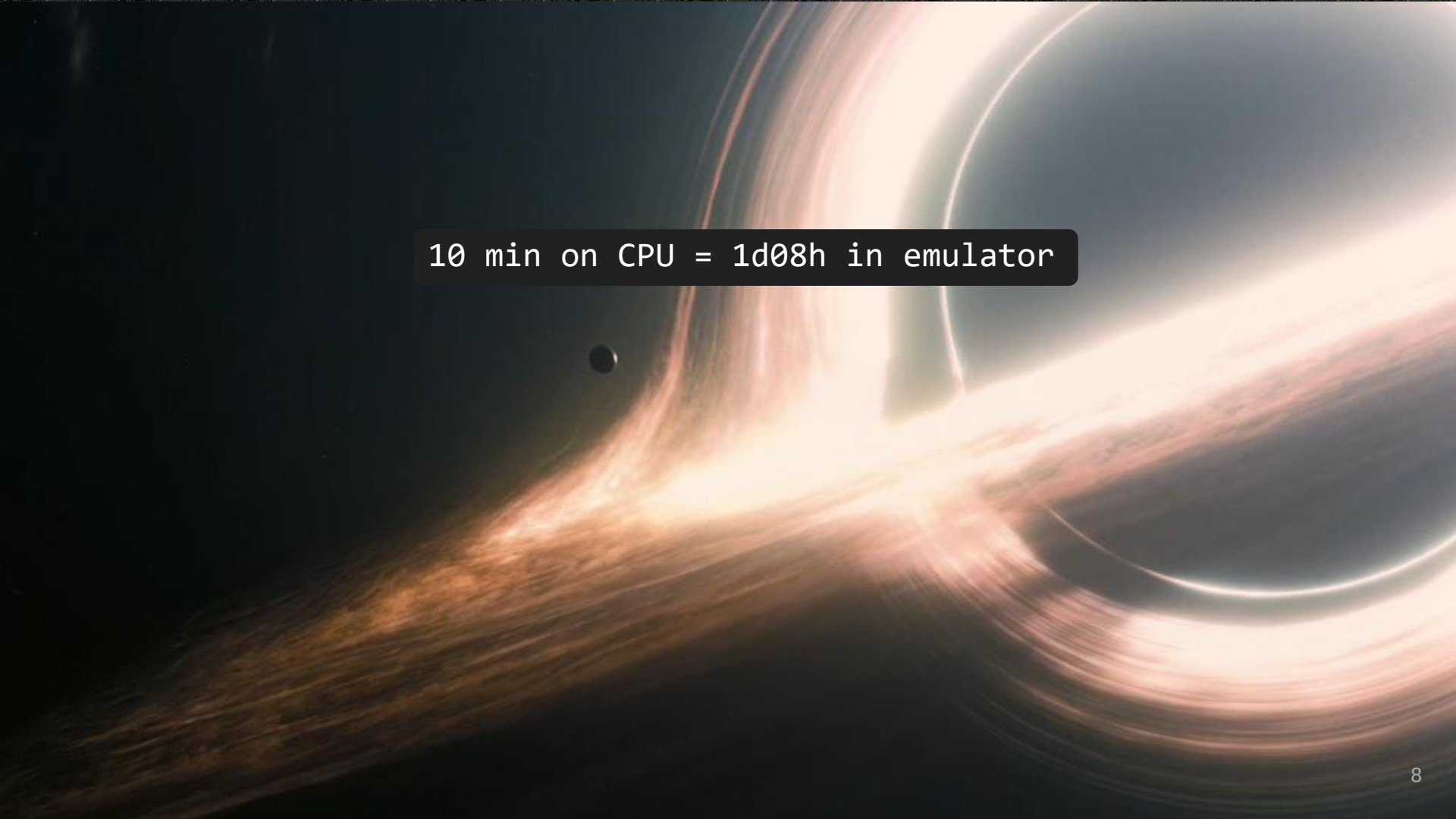
Emulation. Visibility Example

```
000AA868   8D85 64F  LEA EAX,DWORD PTR [EBP-9C]
000AA86E   C785 64F  MOV DWORD PTR [EBP-9C],400
000AA878   50        PUSH EAX
000AA879   68 F0CD0  PUSH 0BCDF0
000AA87E   FF15 500  CALL DWORD PTR [B0050]          advapi32.GetUserNameA
000AA884   85C0      TEST EAX,EAX
000AA886   75 05     JNZ SHORT 000AA88D
000AA888   E8 9BFEF  CALL 000AA728
000AA8F2   56        PUSH ESI                        ASCII "CurrentUser"
000AA8F3   68 F0CD0  PUSH 0BCDF0                     ASCII "secuser"
000AA8F8   FF15 E00  CALL DWORD PTR [B00E0]          kernel32.lstrcmpA
000AA8FE   8B1D 340  MOV EBX,DWORD PTR [B0134]       kernel32.GetProcessHeap
000AA904   85C0      TEST EAX,EAX
000AA906   75 05     JNZ SHORT 000AA90D
000AA908   E8 1BFEF  CALL 000AA728
000AA90D   6A 09     PUSH 9
000AA90F   6A 08     PUSH 8
000AA911   C745 84   MOV DWORD PTR [EBP-7C],20095124
000AA918   C745 88   MOV DWORD PTR [EBP-78],300F5B04
```

Runtime Overhead. Stalling Code




```
1 int __thiscall stalling_func(int arg1)
2 {
3   // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5   tmp_arg1 = arg1;
6   if ( arg1 )
7   {
8     if ( arg1 == 1 )
9     {
10      result = 1;
11    }
12    else
13    {
14      res = stalling_func(0);
15      res2 = res + stalling_func(tmp_arg1 - 1) + 1;
16      res3 = stalling_func(1) + 1;
17      res4 = res3 - (stalling_func(0) + 1);
18      res5 = res4 + stalling_func(1) + 1;
19      result = res5 + stalling_func(tmp_arg1 - 2) - res2;
20    }
21  }
22  else
23  {
24    result = 0;
25  }
26  return result;
27 }
```



10 min on CPU = 1d08h in emulator

Emulation. Visibility Example

```
000AA868   8D85 64F  LEA EAX,DWORD PTR [EBP-9C]
000AA86E   C785 64F  MOV DWORD PTR [EBP-9C],400
000AA878   50        PUSH EAX
000AA879   68 F0CD0  PUSH 0BCDF0
000AA87E   FF15 500  CALL DWORD PTR [B0050]          advapi32.GetUserNameA
000AA884   85C0      TEST EAX,EAX
000AA886   75 05     JNZ SHORT 000AA88D
000AA888   E8 9BFEF  CALL 000AA728
000AA8F2   56        PUSH ESI                        ASCII "CurrentUser"
000AA8F3   68 F0CD0  PUSH 0BCDF0                     ASCII "secuser"
000AA8F8   FF15 E00  CALL DWORD PTR [B00E0]          kernel32.lstrcmpA
000AA8FE   8B1D 340  MOV EBX,DWORD PTR [B0134]       kernel32.GetProcessHeap
000AA904   85C0      TEST EAX,EAX
000AA906   75 05     JNZ SHORT 000AA90D
000AA908   E8 1BFEF  CALL 000AA728
000AA90D   6A 09     PUSH 9
000AA90F   6A 08     PUSH 8
000AA911   C745 84   MOV DWORD PTR [EBP-7C],20095124
000AA918   C745 88   MOV DWORD PTR [EBP-78],300F5B04
```

Syscalls Tracing. Visibility Example

```
000AA87E    FF15 500 CALL DWORD PTR [B0050]    advapi32.GetUserNameA
```

API Tracing. Visibility Example

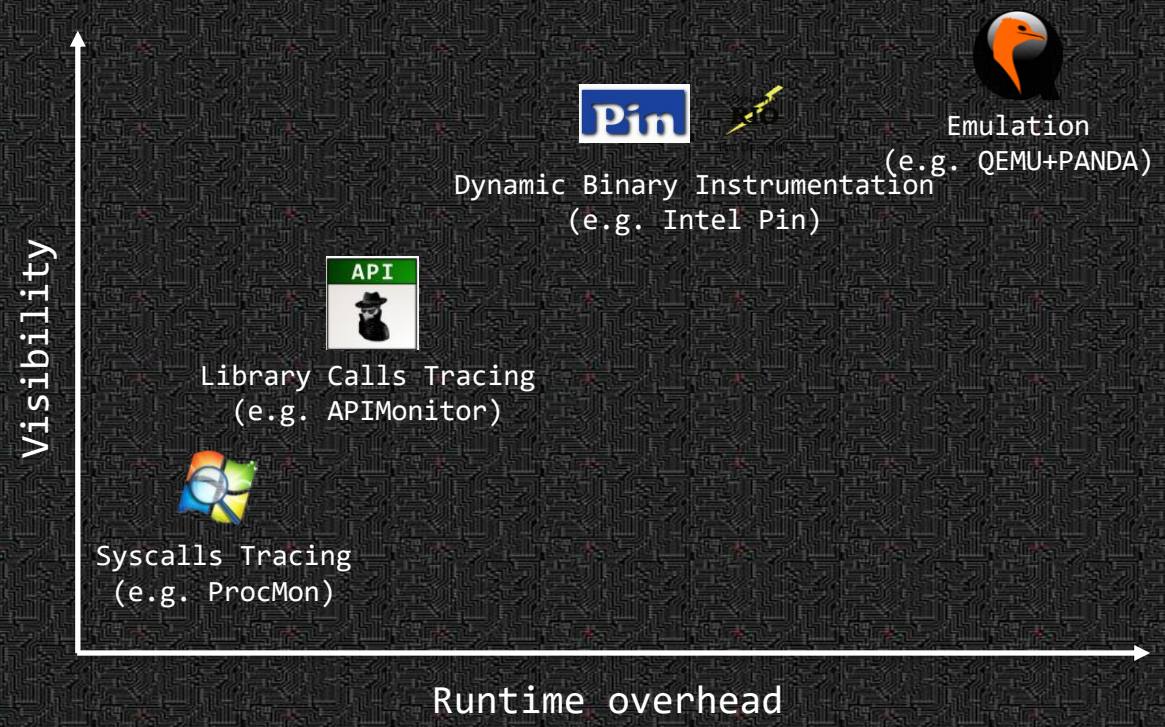
```
000AA87E    FF15 500 CALL DWORD PTR [B0050]    advapi32.GetUserNameA

                                               ASCII "CurrentUser"
                                               ASCII "secuser"
000AA8F8    FF15 E00 CALL DWORD PTR [B00E0]    kernel32.lstrcmpA
```

Ltrace for Linux

```
osboxes@osboxes:~$ ltrace ls
malloc(552)                                                                    = 0x1afc010
malloc(120)                                                                    = 0x1afc240
malloc(1024)                                                                   = 0x1afc2c0
free(0x1afc2c0)                                                                = <void>
free(0x1afc010)                                                                = <void>
__libc_start_main(0x402a00, 1, 0x7fffa84730b8, 0x413be0 <unfinished ...>
strrchr("ls", '/')                                                             = nil
setlocale(LC_ALL, "" <unfinished ...>
malloc(5)                                                                      = 0x1afc010
free(0x1afc010)                                                                = <void>
```

Current Situation in Dynamic Analysis

Current Situation in Dynamic Analysis

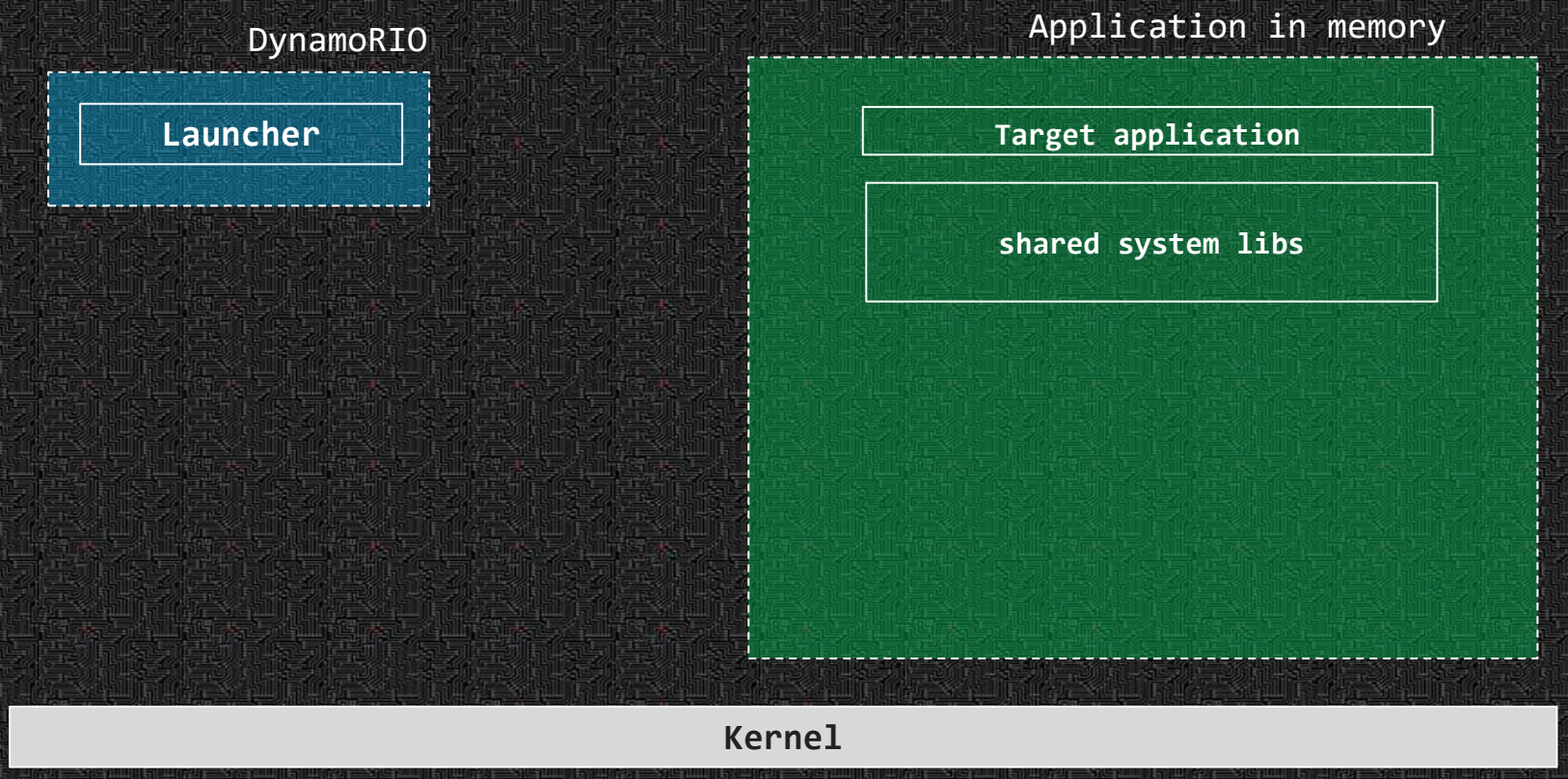


Dynamic Binary Instrumentation (DBI) is a technique of analyzing the behavior of a binary application at runtime through the injection of instrumentation code.

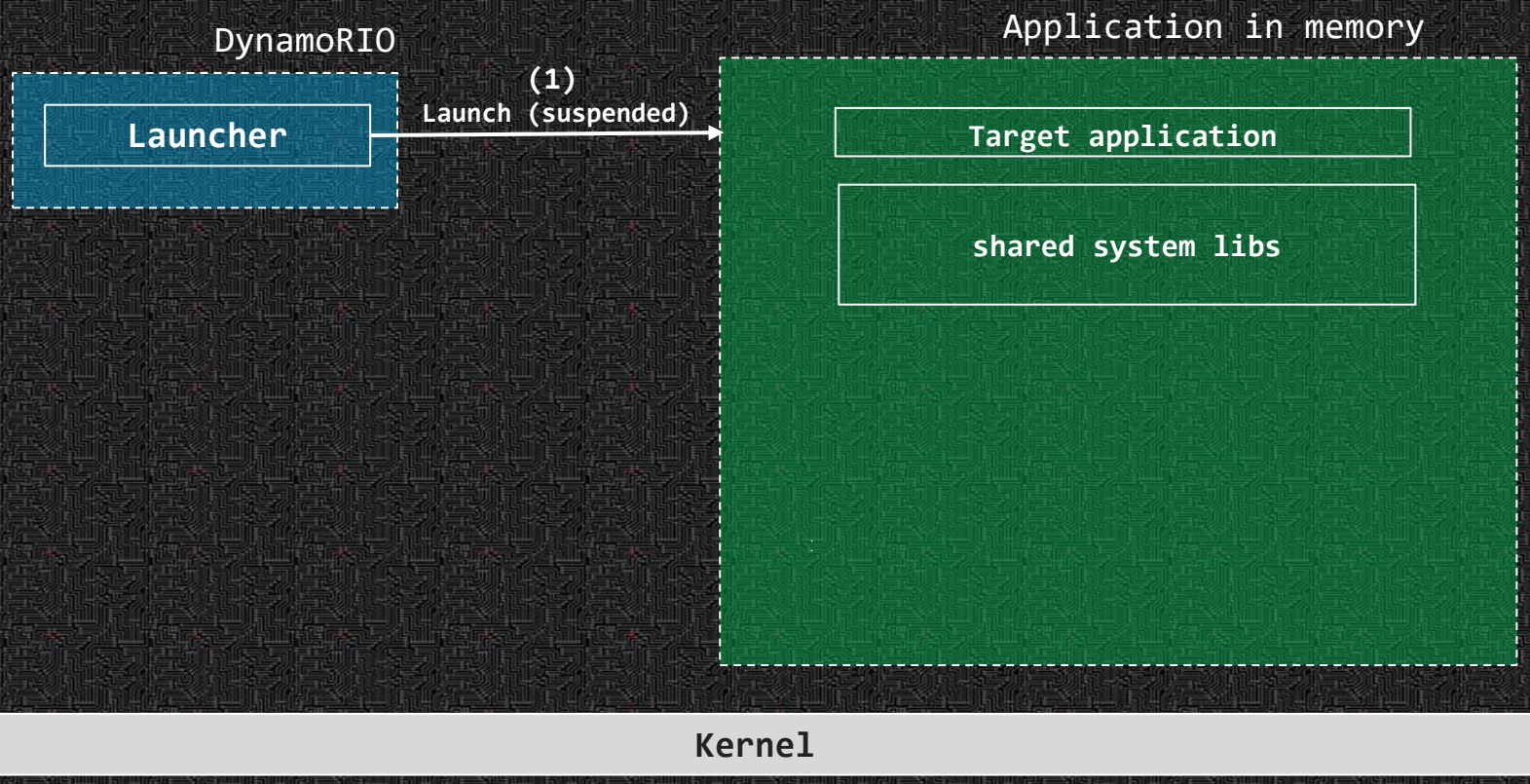
Modern DBI Frameworks

	DynamoRIO	Intel PIN
Redistribution model	Open-source, BSD – license	Proprietary
Supported architectures	x86, x86-64, ARM, AArch64	x86, x86-64
Supported Platforms	Linux, Windows, MacOS, Android	Linux, Windows, MacOS, Android
Average runtime overhead	108% (no tool) 139% (BBs counter)	130% (no tool) 162% (BBs counter)
Language	C/C++	C/C++ (some Python wrappers available)
Technology	Binary code transformation	callout/trampolines

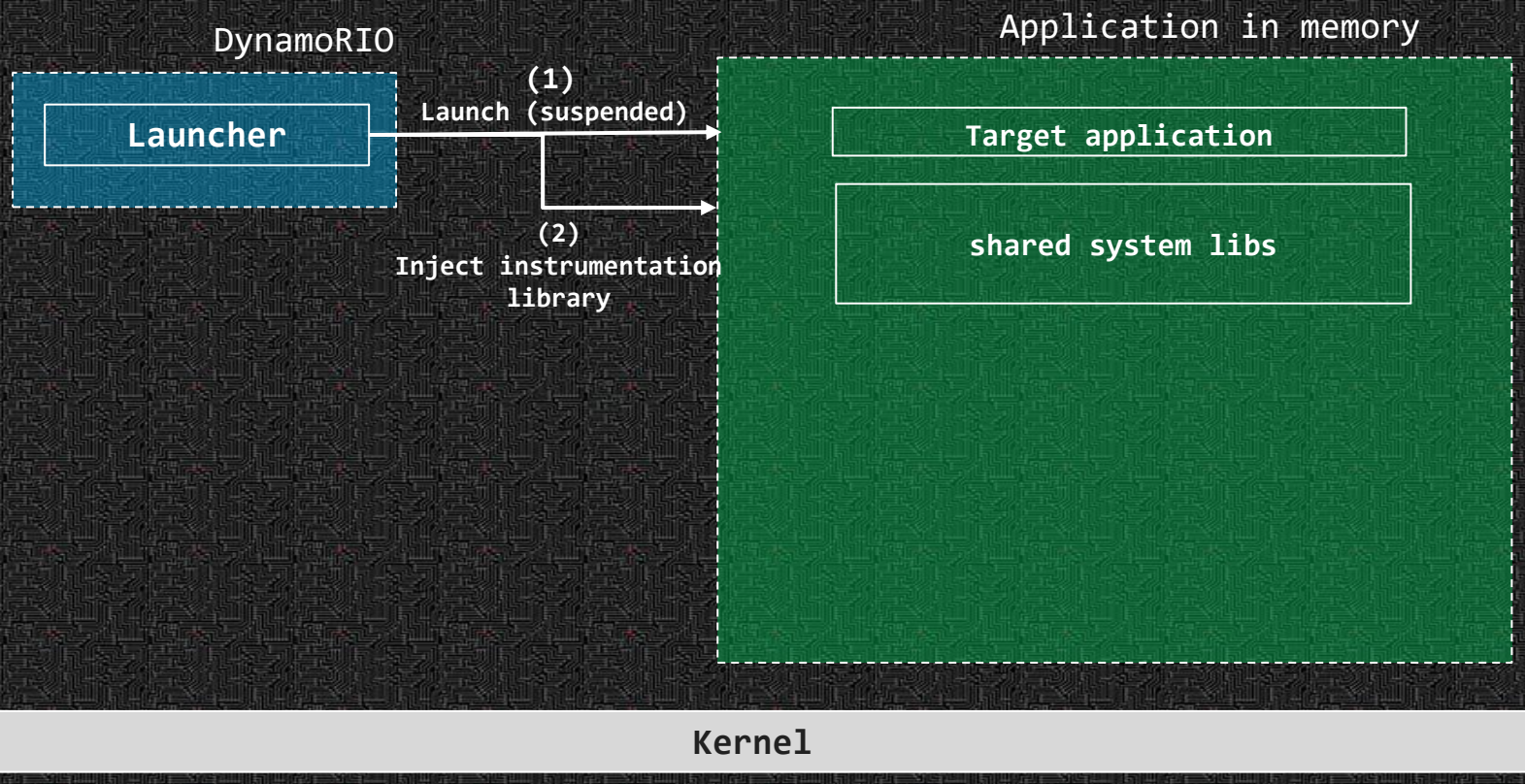
How Does DynamoRIO Work ? (10000 foot view)



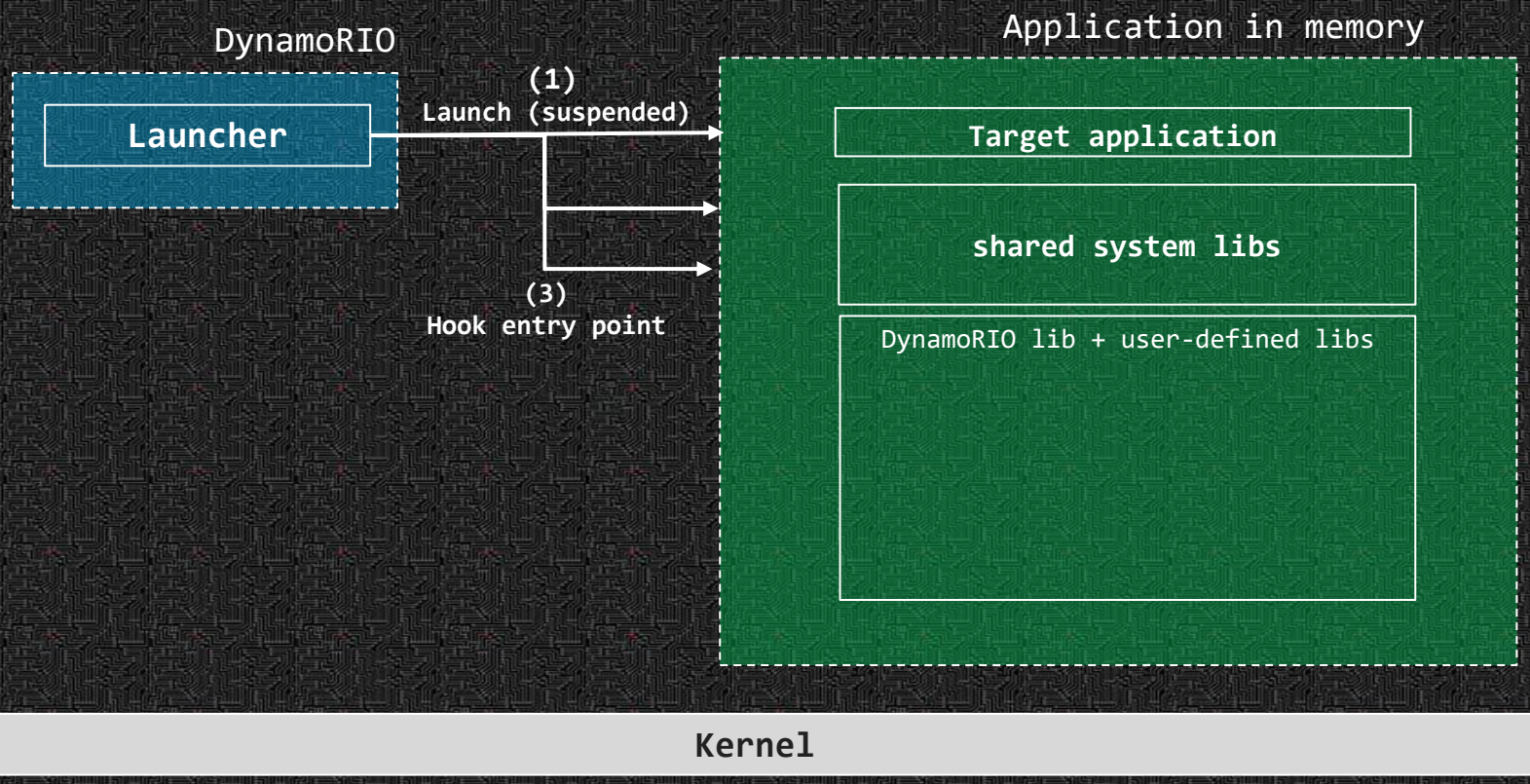
How Does DynamoRIO Work ? (10000 foot view)



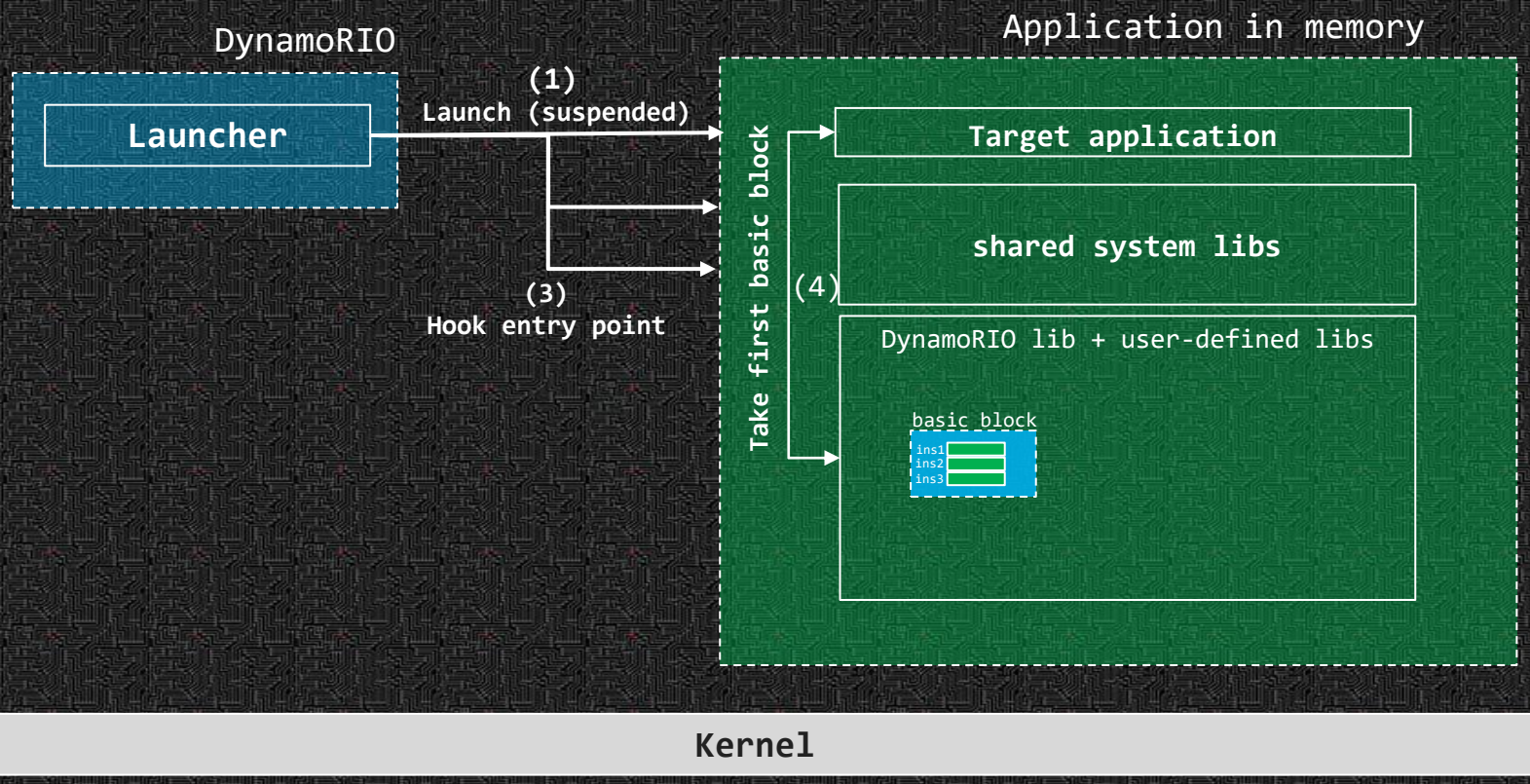
How Does DynamoRIO Work ? (10000 foot view)



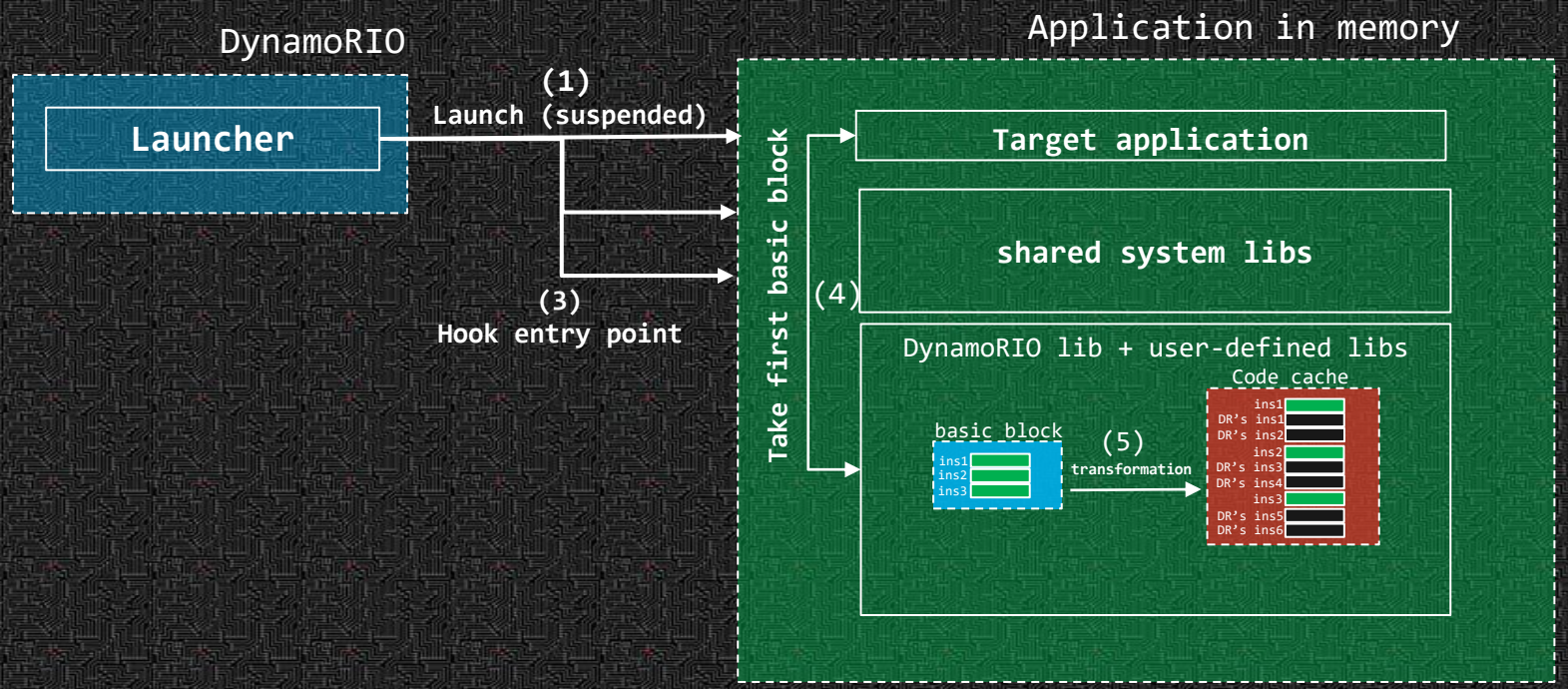
How Does DynamoRIO Work ? (10000 foot view)



How Does DynamoRIO Work ? (10000 foot view)

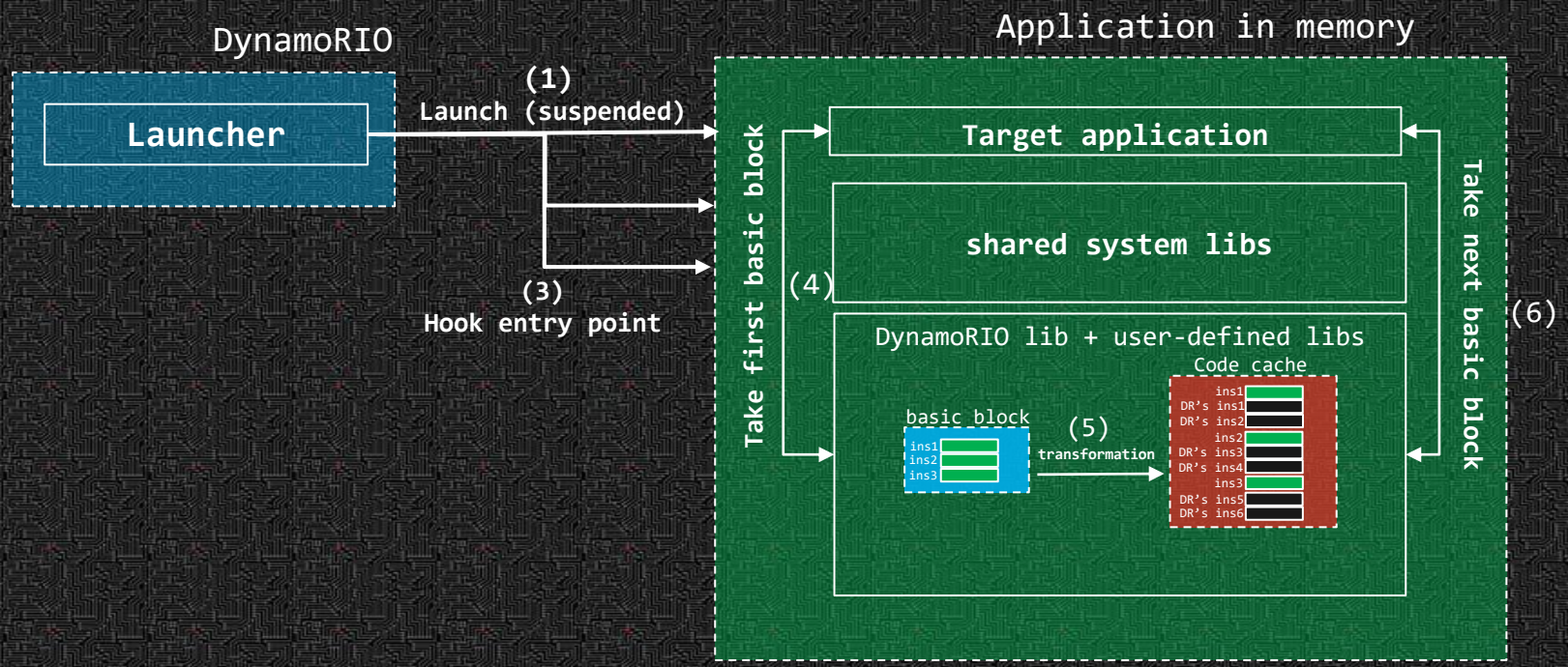


How Does DynamoRIO Work ? (10000 foot view)



Kernel

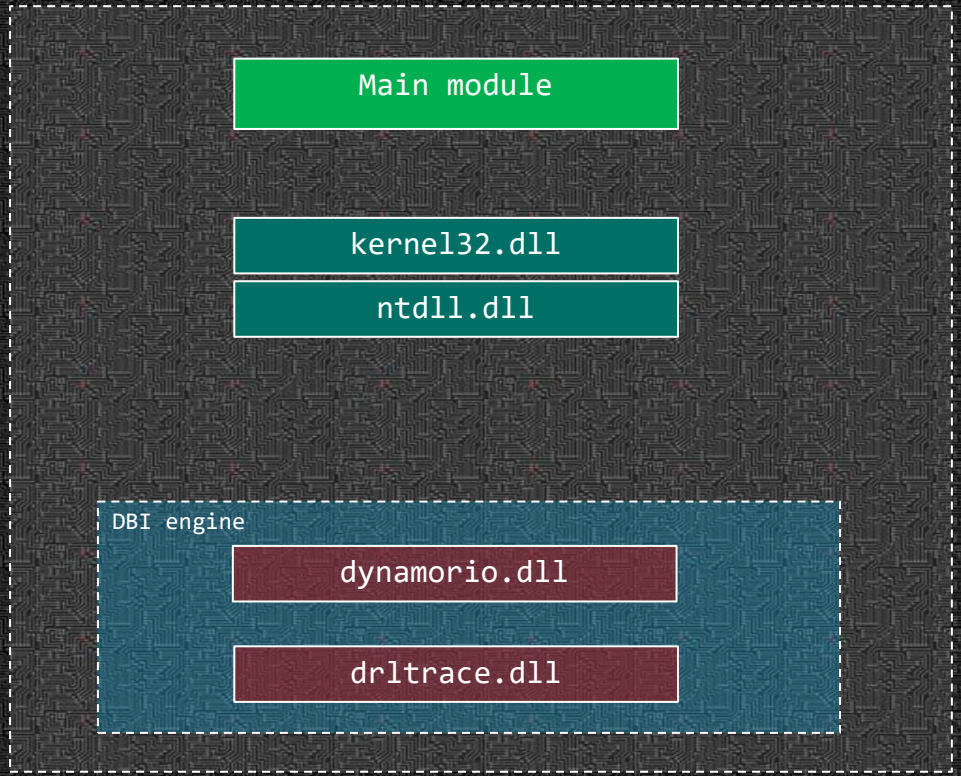
How Does DynamoRIO Work ? (10000 foot view)



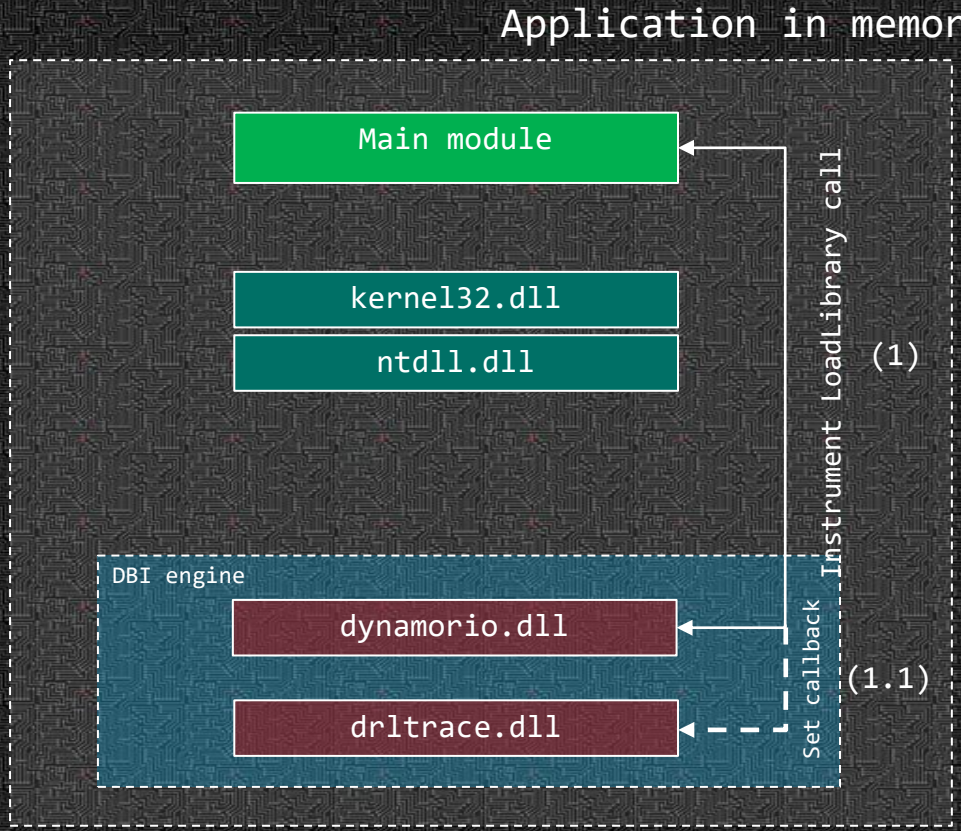
Kernel

How Does Drltrace Work ?

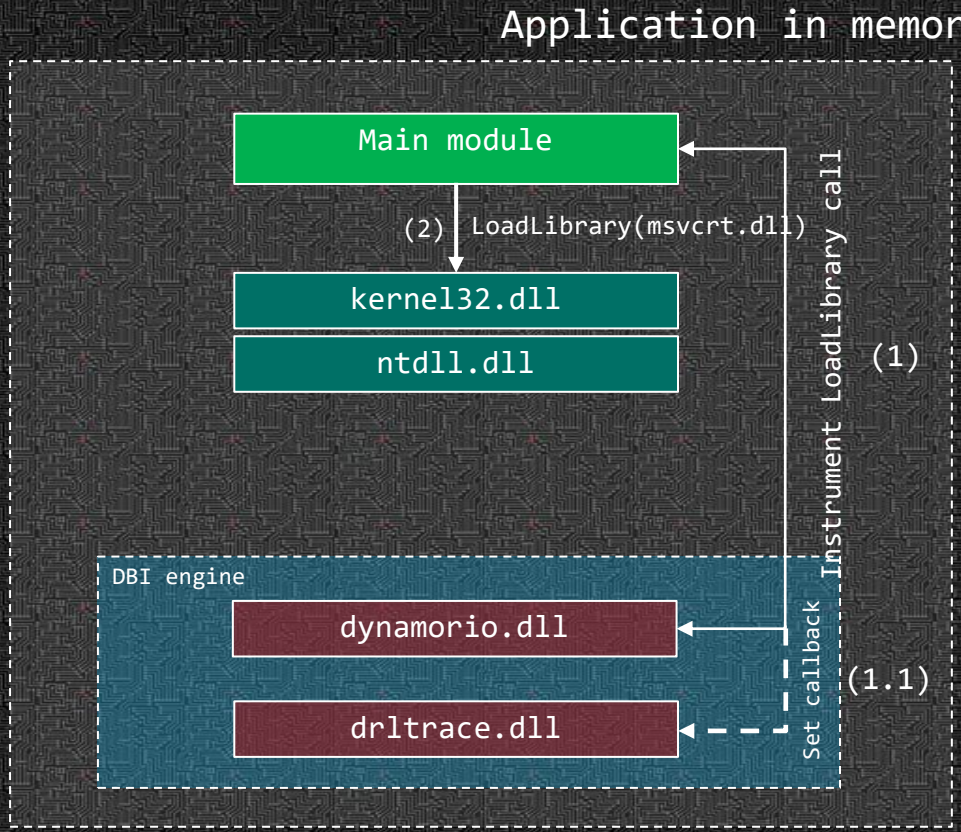
Application in memory



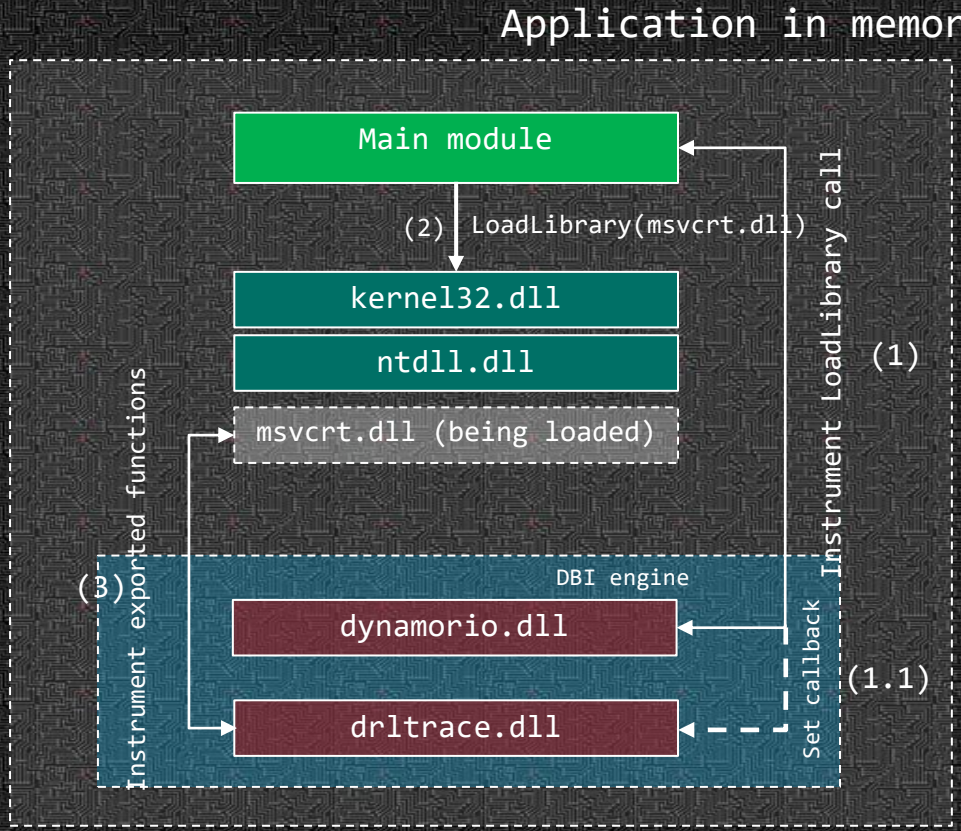
How Does Drltrace Work ?



How Does Drltrace Work ?



How Does Drltrace Work ?



DynamoRIO and Drltracelib in The Memory of Calculator

cmd.exe	2864		1,676 K	2,456 K	Windows Command Processor	Microsoft Corporation
drltrace.exe	3272		1,256 K	2,632 K	Library call tracing tool	Dr. Memory developers
calc.exe	3400		17,768 K	32,924 K	Windows Calculator	Microsoft Corporation
notepad++.exe	2236	0.02	906,464 K	905,132 K	Notepad++ : a free (GNU) so...	Don HO don.h@free.fr
procexp.exe	2064	1.68	11,948 K	18,516 K	Sysinternals Process Explorer	Sysinternals - www.sysinter...

Name	Description	Company Name	Path
drltracelib.dll	Library call tracer library	Dr. Memory developers	C:\Users\secuser\Desktop\drltrace\bin\release\drltracelib.dll
dynamorio.dll	DynamoRIO core library	DynamoRIO developers	C:\Users\secuser\Desktop\drltrace\dynamorio\lib32\releas...

Usage

```
drltrace.exe -logdir . - malware.exe
```

```
234369  ~~2840~~ WINHTTP.dll!WinHttpConnect
234370      arg 0: 0x003ca440 (type=<unknown>, size=0x0)
234371      arg 1: susiku.info (type=wchar_t*, size=0x0)
234372      arg 2: 0x00000050 (type=<unknown>, size=0x0)
234373      arg 3: 0x0 (type=DWORD, size=0x4)
234553  ~~2840~~ WINHTTP.dll!WinHttpOpenRequest
234554      arg 0: 0x004173a0 (type=<unknown>, size=0x0)
234555      arg 1: GET (type=wchar_t*, size=0x0)
234556      arg 2: /rbody320 (type=wchar_t*, size=0x0)
234557      arg 3: <null> (type=wchar_t*, size=0x0)
234558      arg 4: <null> (type=wchar_t*, size=0x0)
234559      arg 5: <null> (type=wchar_t*, size=0x0)
```

Why DrLtrace Rock

- Support x86 and x64 (ARM in future).

Why DrLtrace Rock

- Support x86 and x64 (ARM in future).
- Support Windows and Linux (macOS in future).

Why DrLtrace Rock

- Support x86 and x64 (ARM in future).
- Support Windows and Linux (macOS in future).
- Support self-modifying code.

Why DrLtrace Rock

- Support x86 and x64 (ARM in future).
- Support Windows and Linux (macOS in future).
- Support self-modifying code.
- Support all types of library API calls (static and dynamic).

Why DrLtrace Rock

- Support x86 and x64 (ARM in future).
- Support Windows and Linux (macOS in future).
- Support self-modifying code.
- Support all types of library API calls (static and dynamic).
- Not-detectable by standard malware anti-research techniques (anti-hooking, anti-debugging and anti-emulation).

Why DrLtrace Rock

- Support x86 and x64 (ARM in future).
- Support Windows and Linux (macOS in future).
- Support self-modifying code.
- Support all types of library API calls (static and dynamic).
- Not-detectable by standard malware anti-research techniques (anti-hooking, anti-debugging and anti-emulation).
- External configuration file to add new API calls.

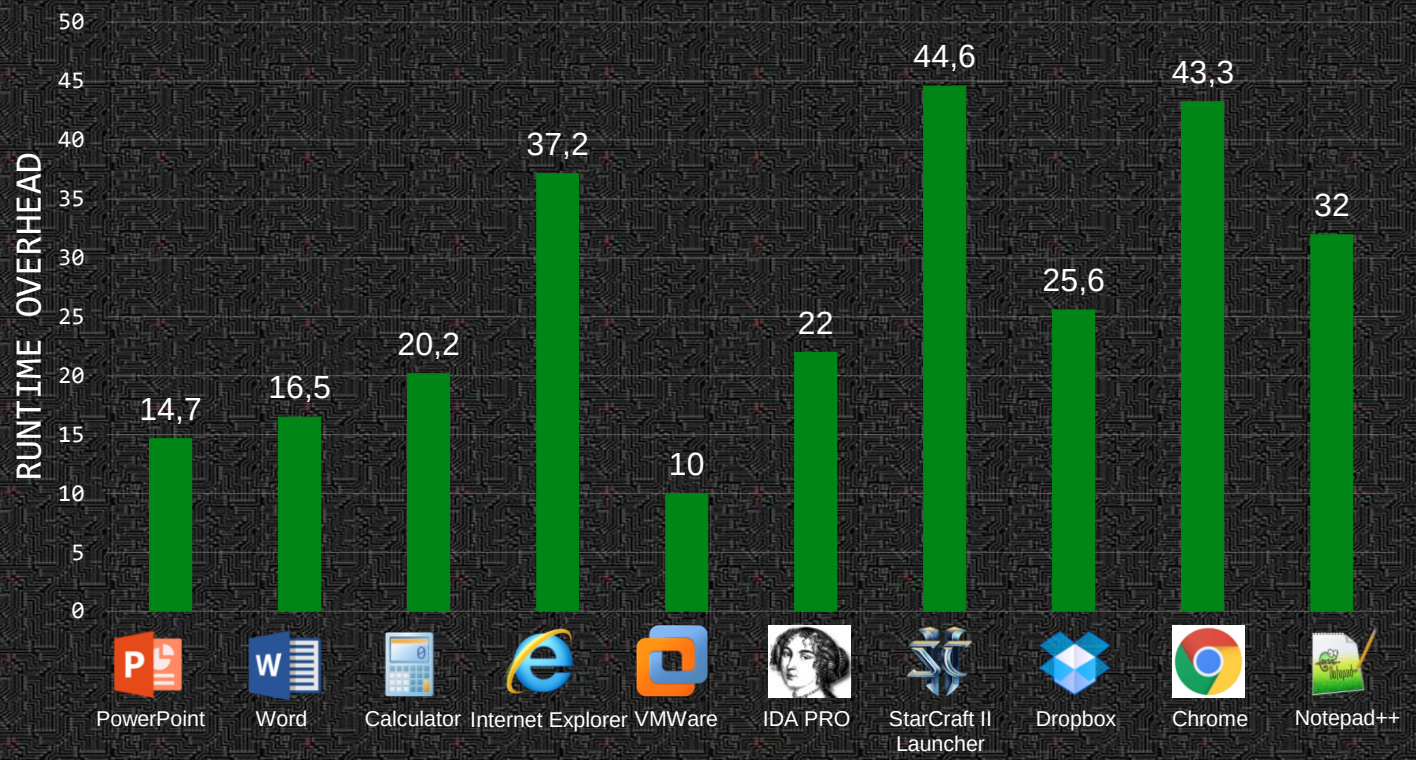
Why DrLtrace Rock

- Support x86 and x64 (ARM in future).
- Support Windows and Linux (macOS in future).
- Support self-modifying code.
- Support all types of library API calls (static and dynamic).
- Not-detectable by standard malware anti-research techniques (anti-hooking, anti-debugging and anti-emulation).
- External configuration file to add new API calls.
- Easy-to-use (no additional dependencies, no heavy-weight GUI).

Why DrLtrace Rock

- Support x86 and x64 (ARM in future).
- Support Windows and Linux (macOS in future).
- Support self-modifying code.
- Support all types of library API calls (static and dynamic).
- Not-detectable by standard malware anti-research techniques (anti-hooking, anti-debugging and anti-emulation).
- External configuration file to add new API calls.
- Easy-to-use (no additional dependencies, no heavy-weight GUI).
- Open-source (BSD-license).

Runtime Overhead



Example 1. EmbusteBot

- Type - Brazilian Banking Trojan
- Language - Delphi
- Main Functionality – Keylogger, Screenshots capturing
- Obfuscation – time-based anti-research checks, encryption of sensitive strings, no code packing
- Operation period – (2017 – present)

Report- https://securityintelligence.com/brazilian-malware-never-sleeps-meet-embustebot/

More details - https://github.com/mxmssh/drltrace/wiki/Malware-Analysis-Examples

Example 1. EmbusteBot

```
drltrace.exe -logdir . -print_ret_addr – vdeis.exe
```

```
167448 ~~2556~~ USER32.dll!GetForegroundWindow
167449     arg 0: 0x0022fbbc (type=void, size=0x0)
167450     and return to module id:32, offset:0x291b60
167451 ~~2556~~ ntdll.dll!KiFastSystemCall
167452     arg 0: 0x01741b60
167453     arg 1: 0x0022fbbc
167454     and return to module id:10, offset:0x13369
167455 ~~2556~~ ntdll.dll!KiFastSystemCallRet
167456     arg 0: 0x01741b60
167457     arg 1: 0x0022fbbc
167458     and return to module id:10, offset:0x13369
167459 ~~2556~~ USER32.dll!GetClassNameW
167460     arg 0: 0x0009014a (type=<unknown>, size=0x0)
167461     arg 2: 0x400 (type=int, size=0x4)
167462     and return to module id:32, offset:0x28641b
167463 ~~2556~~ ntdll.dll!KiFastSystemCall
167464     arg 0: 0x777b2a4d
167465     arg 1: 0x0009014a
167466     and return to module id:10, offset:0x11b6c
167467 ~~2556~~ ntdll.dll!KiFastSystemCallRet
167468     arg 0: 0x777b2a4d
167469     arg 1: 0x0009014a
167470     and return to module id:10, offset:0x11b6c
167471 ~~2556~~ USER32.dll!CharUpperBuffW
167472     arg 0: PROCEXPL (type=wchar_t*, size=0x0)
167473     arg 1: 0x8 (type=DWORD, size=0x4)
167474     and return to module id:32, offset:0x219d4
```

```
385907 ~~2556~~ USER32.dll!GetForegroundWindow
385908     arg 0: 0x0022fbbc (type=void, size=0x0)
385909     and return to module id:32, offset:0x291b60
385910 ~~2556~~ ntdll.dll!KiFastSystemCall
385911     arg 0: 0x01741b60
385912     arg 1: 0x0022fbbc
385913     and return to module id:10, offset:0x13369
385914 ~~2556~~ ntdll.dll!KiFastSystemCallRet
385915     arg 0: 0x01741b60
385916     arg 1: 0x0022fbbc
385917     and return to module id:10, offset:0x13369
385918 ~~2556~~ USER32.dll!GetClassNameW
385919     arg 0: 0x000c0318 (type=<unknown>, size=0x0)
385920     arg 2: 0x400 (type=int, size=0x4)
385921     and return to module id:32, offset:0x28641b
385922 ~~2556~~ ntdll.dll!KiFastSystemCall
385923     arg 0: 0x777b2a4d
385924     arg 1: 0x000c0318
385925     and return to module id:10, offset:0x11b6c
385926 ~~2556~~ ntdll.dll!KiFastSystemCallRet
385927     arg 0: 0x777b2a4d
385928     arg 1: 0x000c0318
385929     and return to module id:10, offset:0x11b6c
385930 ~~2556~~ USER32.dll!CharUpperBuffW
385931     arg 0: IEFrame (type=wchar_t*, size=0x0)
385932     arg 1: 0x7 (type=DWORD, size=0x4)
385933     and return to module id:32, offset:0x219d4
```

Example I. EmbusteBot. Searching for Tab with Bank Name

```
385980  ~~2556~~ USER32.dll!GetWindowTextW
385981      arg 0: 0x000c0318 (type=<unknown>, size=0x0)
385982      arg 2: 0x200 (type=int, size=0x4)
385983      and return to module id:32, offset:0x291ccb
```

```
386033  ~~2556~~ USER32.dll!CharUpperBuffW
386034      arg 0: Bank name - P - Microsoft Internet Explorer - Windows Internet Explorer
386035      arg 1: 0x46 (type=DWORD, size=0x4)
386036      and return to module id:32, offset:0x219d4
```

Example I. EmbusteBot. Screenshots Capturing and Keylogging API Calls

```
3539007 ~~2556~~ USER32.dll!GetDC
3539008     arg 0: <null> (type=<unknown>, size=0x0)
3539009     and return to module id:32, offset:0xfb266
3539018 ~~2556~~ GDI32.dll!CreateCompatibleDC
3539019     arg 0: 0x66010d48 (type=<unknown>, size=0x0)
3539020     and return to module id:32, offset:0xfb277
3539029 ~~2556~~ GDI32.dll!CreateCompatibleBitmap
3539030     arg 0: 0x66010d48 (type=<unknown>, size=0x
3539031     arg 1: 0x690 (type=int, size=0x4)
3539032     arg 2: 0x41a (type=int, size=0x4)
3539033     and return to module id:32, offset:0xfb2e7
3539077 ~~2556~~ GDI32.dll!SelectObject
3539078     arg 0: 0x27010f88 (type=<unknown>, size=0x
3539079     arg 1: 0x32050963 (type=<unknown>, size=0x0)
3539080     and return to module id:32, offset:0xfb6b2
3539089 ~~2556~~ GDI32.dll!BitBlt
3539090     arg 0: 0x0e010f38 (type=<unknown>, size=0x0)
3539091     arg 1: 0x0 (type=int, size=0x4)
3539092     arg 2: 0x0 (type=int, size=0x4)
3539093     arg 3: 0x690 (type=int, size=0x4)
3539094     arg 4: 0x41a (type=int, size=0x4)
3539095     arg 5: 0x27010f88 (type=<unknown>, size=0x0)
3539096     and return to module id:32, offset:0xfb73a
```

```
666323 ~~2556~~ USER32.dll!SetWindowsHookExW
666324     arg 0: 0xd (type=int, size=0x4)
666325     arg 1: 0x017332a8 (type=<unknown>, size=0x0)
666326     arg 2: 0x00400000 (type=<unknown>, size=0x0)
666327     arg 3: 0x0 (type=DWORD, size=0x4)
666328     and return to module id:32, offset:0x283262
```

Example I. EmbusteBot. Trigger

```
6432  ~~1464~~ KERNEL32.dll!LoadLibraryA
6433      arg 0: C:\Users\Public\Media\vdeis2.dll (type=char*, size=0x0)
```

```
386582  ~~2556~~ KERNEL32.dll!GetFileAttributesW
386583      arg 0: C:\Users\Public\Media\171703.reg (type=wchar_t*,
386584      and return to module id:32, offset:0x22c02
```

Example II. Gootkit Loader

- Type : Banking Trojan
- Language : C
- Main Functionality: Unpack actual payload loader, deliver Gootkit malware on victim's machine
- Obfuscation: anti-VM, anti-debugging, anti-emulation, time-based anti-research, packed payload, machine-code obfuscation, anti-sandboxing.
- Operation period: (2014 – present)

Technical report - https://drive.google.com/file/d/0BzFSoGMCVlTORUExdF9RTklpX3c/view

More details - https://github.com/mxmssh/drltrace/wiki/Malware-Analysis-Examples

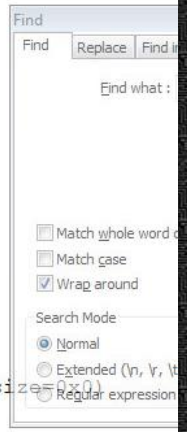
Example II. Gootkit Loader

```
drltrace.exe -logdir . -print_ret_addr -- 477c305~f01.exe
```

Name	Date modified	Type	Size
drltrace.attrib.exe.03500.0000.log	11/7/2017 10:05 AM	Text Document	31 KB
drltrace.cmd.exe.02868.0000.log	11/7/2017 10:05 AM	Text Document	583 KB
drltrace.mstsc.exe.04076.0000.log	11/7/2017 10:05 AM	Text Document	3,513 KB
drltrace.477c3057411648154852 18f165256...	11/7/2017 10:05 AM	Text Document	137 KB

Example II. Gootkit Loader. Unpacking

```
65  ~~2272~~ KERNEL32.dll!ReadProcessMemory
66      arg 0: 0xffffffff (type=HANDLE, size=0x4)
67      arg 1: 0x737e0188 => 0x00000000 (type=void*, size=0x0)
68      arg 3: 0x8 (type=size_t, size=0x4)
69      and return to module id:0, offset:0x4f4c
70  ~~2272~~ KERNELBASE.dll!ReadProcessMemory
71      arg 0: 0xffffffff (type=HANDLE, size=0x4)
72      arg 1: 0x737e0188 => 0x00000000 (type=void*, size=0x0)
73      arg 3: 0x8 (type=size_t, size=0x4)
74      and return to module id:0, offset:0x4f4c
75  ~~2272~~ ntdll.dll!RtlEncodePointer
76      arg 0: 0x00000000
77      arg 1: 0x772d0000
78      and return to module id:0, offset:0x4f59
79  ~~2272~~ KERNEL32.dll!GetProcAddress
80      arg 0: 0x772d0000
81      arg 1: 0x00409972
82      and return to module id:0, offset:0x4f7e
83  ~~2272~~ KERNELBASE.dll!GetProcAddress
84      arg 0: 0x772d0000
85      arg 1: 0x00409972
86      and return to module id:0, offset:0x4f7e
87  ~~2272~~ KERNEL32.dll!VirtualAlloc
88      arg 0: 0x00000000 => 0x00000000 (type=void*, size=0x0)
89      arg 1: 0x688 (type=size_t, size=0x4)
90      arg 2: 0x1000 (type=DWORD, size=0x4)
91      arg 3: 0x40 (type=DWORD, size=0x4)
92      and return to module id:0, offset:0x277f
```



Example II. Gootkit Loader. Process Hollowing. New Process Creation

```
2754 ~~2272~~ KERNEL32.dll!lstrcatW
2755     arg 0: C:\Windows\System32\mstsc.exe "C:\Users\secuser\Desktop\477c305741164815485
2756     arg 1: " (type=wchar_t*, size=0x0)
2757     and return to module id:0, offset:0x9233
2758 ~~2272~~ KERNEL32.dll!CreateProcessW
2759     arg 0: <null> (type=wchar_t*, size=0x0)
2760     arg 1: C:\Windows\System32\mstsc.exe "C:\Users\secuser\Desktop\477c305741164815485
2761     arg 2: <null> (type=<unknown>*, size=0x0)
2762     arg 3: <null> (type=<unknown>*, size=0x0)
2763     arg 4: 0x0 (type=BOOL, size=0x4)
2764     arg 5: 0x800000c (type=DWORD, size=0x4)
2765     and return to module id:0, offset:0x9263
```

```
2777 ~~2272~~ KERNEL32.dll!GetThreadContext
2778     arg 0: 0x108 (type=HANDLE, size=0x4)
2779     arg 1: 0x0012fc30 (type=<unknown>*, size=0x0)
2780     and return to module id:0, offset:0x92a1
```

Example II. Gootkit Loader. Process Hollowing. New Section

```
2893 ~~2272~~ ntdll.dll!ZwCreateSection
2894     arg 1: 0xf001f (type=unsigned int, size=0x4)
2895     arg 2: 0x0012fae8 (type=OBJECT_ATTRIBUTES*, size=0x4)
2896     arg 3: 0x0012fb10 (type=LARGE_INTEGER*, size=0x4)
2897     arg 4: 0x40 (type=unsigned int, size=0x4)
2898     arg 5: 0x8000000 (type=unsigned int, size=0x4)
2899     and return to module id:0, offset:0xa013
```

```
3463 ~~2272~~ ntdll.dll!ZwMapViewOfSection
3464     arg 0: 0x114 (type=HANDLE, size=0x4)
3465     arg 1: 0x10c (type=HANDLE, size=0x4)
3466     arg 2: 0x0012fbc0 => 0x00000000 (type=void **, size=0x4)
3467     arg 3: 0x0 (type=unsigned int, size=0x4)
3468     arg 4: 0x0 (type=unsigned int, size=0x4)
3469     arg 5: 0x0012fb24 (type=LARGE_INTEGER*, size=0x4)
3470     and return to module id:0, offset:0x9d42
```

Example II. Gootkit Loader. Process Hollowing. Write & Resume

```
3721 ~~2272~~ KERNEL32.dll!WriteProcessMemory
3722     arg 0: 0x10c (type=HANDLE, size=0x4)
3723     arg 1: 0x00095ae2 => 0x00000000 (type=void*, size=0x0)
3724     arg 2: 0x0012fb80 => 0x00000000 (type=void*, size=0x0)
3725     arg 3: 0x1 (type=size_t, size=0x4)
3726     and return to module id:0, offset:0x94fc
3727 ~~2272~~ KERNELBASE.dll!WriteProcessMemory
3728     arg 0: 0x10c (type=HANDLE, size=0x4)
3729     arg 1: 0x00095ae2 => 0x00000000 (type=void*, size=0x0)
3730     arg 2: 0x0012fb80 => 0x00000000 (type=void*, size=0x0)
3731     arg 3: 0x1 (type=size_t, size=0x4)
3732     and return to module id:0, offset:0x94fc
3733 ~~2272~~ KERNEL32.dll!ResumeThread
3734     arg 0: 0x108 (type=HANDLE, size=0x4)
3735     and return to module id:0, offset:0x9512
```

Example II. Gootkit Loader

```
3306 ~~3852~~ KERNEL32.dll!GetEnvironmentVariableA
3307     arg 0: crackmeololo (type=char*, size=0x0)
3308     arg 2: 0x104 (type=DWORD, size=0x4)
```

Example II. Gootkit Loader

```
87798 ~~1756~~ ADVAPI32.dll!RegQueryValueExW
87799     arg 0: 0x00000118 (type=<unknown>, size=0x0)
87800     arg 1: ProcessorNameString (type=wchar_t*, size=0x0)
87801     arg 2: 0x00000000 (type=DWORD*, size=0x4)
87802     arg 5: 0x01b2f6d4 => 0x200 (type=DWORD*, size=0x4)
```

```
87628 ~~1756~~ ADVAPI32.dll!RegOpenKeyW
87629     arg 0: 0x80000002 (type=<unknown>, size=0x0)
87630     arg 1: Hardware\DESCRIPTION\System\CentralProcessor\0
```

```
87855 ~~1756~~ SHLWAPI.dll!StrStrIW
87856     arg 0: Intel(R) CPU E5-2650 v3 @ 2.30GHz (type=wchar_t*,
87857     arg 1: Xeon (type=wchar_t*, size=0x0)
```

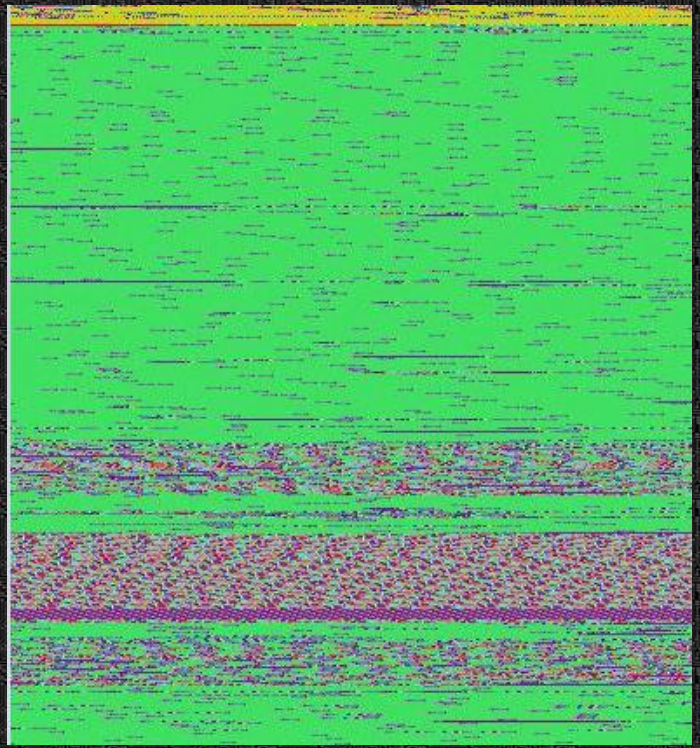
Example II. Gootkit Loader

```
234368
234369  ~~2840~~ WINHTTP.dll!WinHttpConnect
234370      arg 0: 0x003ca440 (type=<unknown>, size=0x0)
234371      arg 1: susiku.info (type=wchar_t*, size=0x0)
234372      arg 2: 0x00000050 (type=<unknown>, size=0x0)
234373      arg 3: 0x0 (type=DWORD, size=0x4)
234553  ~~2840~~ WINHTTP.dll!WinHttpOpenRequest
234554      arg 0: 0x004173a0 (type=<unknown>, size=0x0)
234555      arg 1: GET (type=wchar_t*, size=0x0)
234556      arg 2: /rbody320 (type=wchar_t*, size=0x0)
234557      arg 3: <null> (type=wchar_t*, size=0x0)
234558      arg 4: <null> (type=wchar_t*, size=0x0)
234559      arg 5: <null> (type=wchar_t*, size=0x0)
```

DEMO. NotPetya/PetrWrap

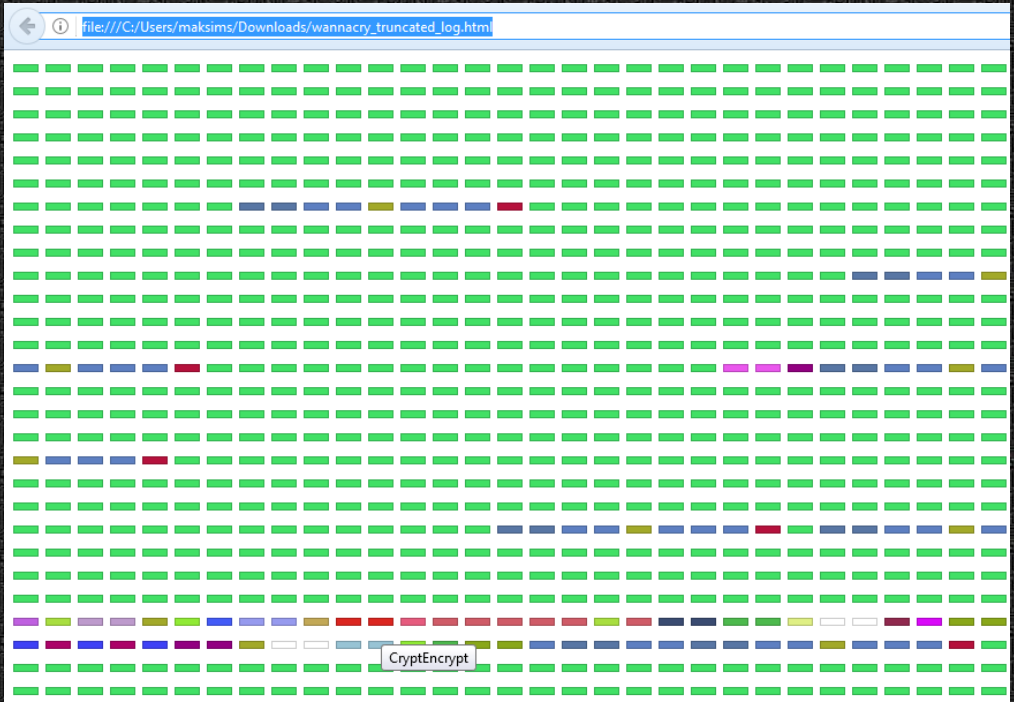
Drltrace. API calls visualization script

```
python api_calls_vis.py -i wannacry.jpeg -gr -t drltrace_log_wannacry.log
```



Drltrace. API calls visualization script

```
python api_calls_vis.py -ht wannacry.html -gr -t drltrace_log_wannacry.log
```



Future Work

- Make DynamoRIO more resistant against anti-DBI tricks.
- Add heuristics to search for certain (YARA rules?) malicious patterns in logs.
- ARM and macOS.
- Attach drltrace into running process

Conclusion

- Dynamic binary instrumentation is a reasonable trade-off for dynamic malware analysis.
- Drltrace is the first efficient and light-weight solution for API calls tracing in modern sophisticated malicious samples based on DBI technique.
- The solution allowed to revel in several minutes a lot of internal technical details about malicious sample without even starting IDA or debugger.

Thank you!

https://github.com/mxmssh/drltrace

https://www.linkedin.com/in/mshudrak
https://twitter.com/MShudrak