A new open-source hypervisor-level malware monitoring and extraction system – current state and further challenges

Michał Leszczyński, Krzysztof Stopczański
Virus Bulletin 2020
@127.0.0.1

$ whoami

Michał Leszczyński

- IT Security Engineer @ cert.pl

- malware analysis tools/infrastructure development
- breaking/auditing/fixing web applications

- monk@cert.pl
- https://icedev.pl/
- @icedevml

$ whoami

Krzysztof Stopczański

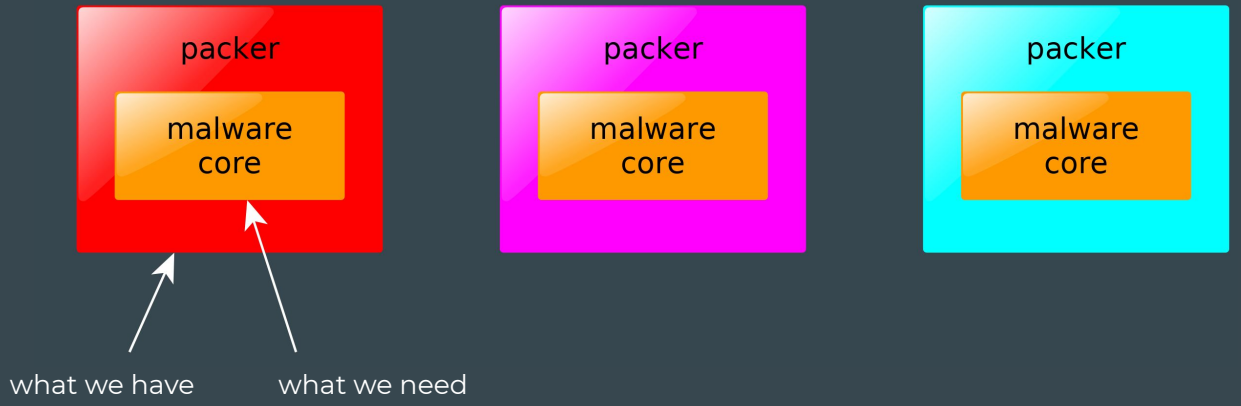
- (Formerly) IT Security Engineer @ cert.pl

- screws up computers
- playing CTFs with p4 team

- krzysztof@stopczanski.pl

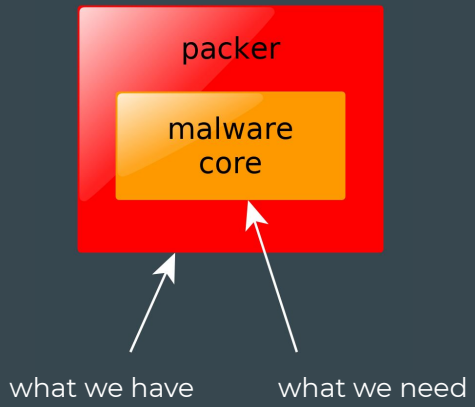
Introduction

Malware 101


packer
malware core
packer
malware core
packer
malware core
what we have
what we need

Malware 101

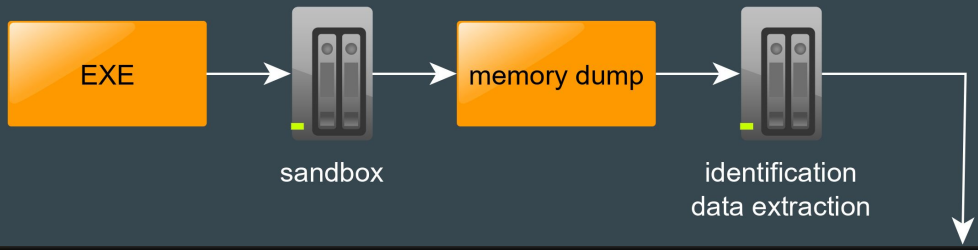
- different packers
- the same/similiar malware core
- malware core => easy identification
- ... also easy data extraction



Malware processing at CERT.PL

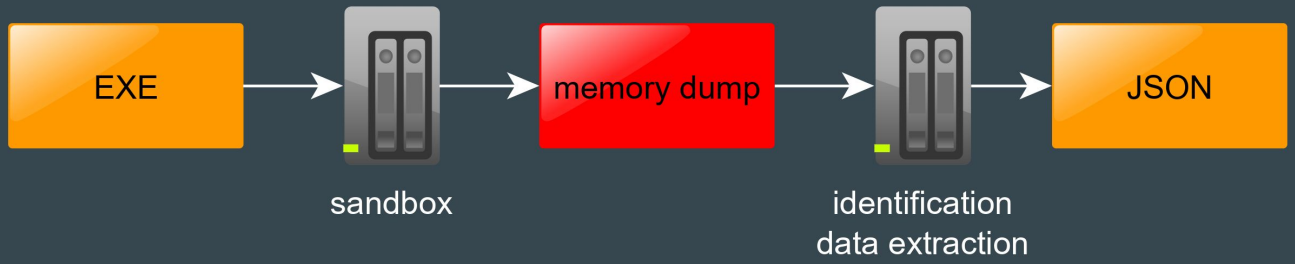
- malware unpacking
- extraction of some interesting stuff

Example:



Family	emotet
Config type	static
+ exe_words	["engine", "finish", "magnify", "resapi", "query", "skip", "wubi", "svcs", "router", "crypto", "backup", "ha…
+ public_key	-----BEGIN PUBLIC KEY----- MHwwDQYJKoZIhvcNAQEBBQADawAwaAJhALk+KlHgOKXm9eDkWu2yN9lanjwOm6W2 PV0tgr4msNVby2pOJ…
+ type	emotet
+ url_words	["teapot", "pnp", "tpt", "splash", "site", "codec", "health", "balloon", "cab", "odbc", "badge", "dma", "pse…
+ urls	[{ "cnc": "186.176.138.171", "port": 7080 }, { "cnc": "200.51.94.251", "port": 80 }, { "cnc": "46.105.131.87…
Upload time	Wed, 16 Oct 2019 11:40:07 GMT

Malware processing at CERT.PL



What is a memory dump?

- logical dump of the memory at given point of time
- metadata:
  - base address at which dump was made,
  - reason of the dump (e.g. malware made some interesting API call)
- profit? unpacked malware (at least sometimes)

Dynamic unpacking - theory

- in order to have good memory dumps, you need good heuristics
- good heuristics need good behavioral monitoring
- why can’t you just use an ordinary sandbox?
  - we do, but...

Malware monitoring problems

Example #1 - trickbot (1c81272ffc)

Example #1 - trickbot (1c81272ffc)

- Well known trojan / stealer
- Packed x86/x64 binaries
- Process hollowing using direct system calls

Sample:

[1] https://mwdb.cert.pl/sample/1c81272ffc28b29a82d8313bd74d1c6030c2af1ba4b165c44dc8ea6376679d9f

References:

[2] https://www.cyberbit.com/blog/endpoint-security/latest-trickbot-variant-has-new-tricks-up-its-sleeve/
[3] https://www.cert.pl/en/news/single/detricking-trickbot-loader/

Example #1 - trickbot (1c81272ffc)

Directly making syscalls - not visible on conventional sandboxes

```
10002600    8B D4        mov edx, esp
10002602    0F 34        sysenter
10002604    C3           ret
```

References:

[2] https://www.cyberbit.com/blog/endpoint-security/latest-trickbot-variant-has-new-tricks-up-its-sleeve/

Example #2 - remcos (60c07bac07)

Example #2 - remcos (60c07bac07)

- Remote Access Trojan
- Packed x86/x64 binaries
- Hollowing **svchost.exe** using WriteProcessMemory()

Sample:
[4] https://mwdb.cert.pl/sample/60c07bac07c7e2f2f3e03817addb88b38b8fbcd893d4b41b5007d984e8ba1fc5

Example #2 - remcos (60c07bac07)

This is how **Cuckoo** hooks **ntdll.dll** (for Windows 7 x86):

```
static int hook_api_jmp_direct(hook_t *h, unsigned char *from,
    unsigned char *to)
{
    // unconditional jump opcode
    *from = 0xe9;

    // store the relative address from this opcode to our hook function
    *(unsigned long *)(from + 1) = (unsigned char *) to - from - 5;
    return 0;
}
```

TLDR: replace first 5 bytes of the hooked function with a 0xE9 jump

Example #2 - remcos (60c07bac07)

This is how **Cuckoo** hooks **ntdll.dll** (for Windows 7 x86):

```
static int hook_api_jmp_direct(hook_t *h, unsigned char *from,
    unsigned char *to)
{
    // unconditional jump opcode
    *from = 0xe9;

    // store the relative address from this opcode to our hook function
    *(unsigned long *)(from + 1) = (unsigned char *) to - from - 5;
    return 0;
}
```

TLDR: replace first 5 bytes of the hooked function with a 0xE9 jump

Example #2 - remcos (60c07bac07)

This is how **Cuckoo** hooks **ntdll.dll** (for Windows 7 x86):

```
static int hook_api_jmp_direct(hook_t *h, unsigned char *from,
    unsigned char *to)
{
    // unconditional jump opcode
    *from = 0xe9;

    // store the relative address from this opcode to our hook function
    *(unsigned long *)(from + 1) = (unsigned char *) to - from - 5;
    return 0;
}
```

TLDR: replace first 5 bytes of the hooked function with a 0xE9 jump

Example #2 - remcos (60c07bac07)

… and this is how remcos unhooks:

```
94    v16 = v26;
95    while ( ++v16 != v27 + v26 )
96    {
97      if ( *(_BYTE *)v16 == 0xB8 && !*(_DWORD *)(v16 + 1) && *(_BYTE *)(v16 + 5) == 0xB9 )
98      {
99        v17 = v16 + 0xA;
100       v18 = 0;
101       v19 = 1;
102       do
103       {
104         ++v18;
105         if ( *(_DWORD *)++v17 == 0x424548D )
106         {
107           if ( *(_WORD *)(v17 - 2) != 0x33C9 && *(_BYTE *)(v17 - 5) == 0xB9 )
108           {
109             *(_BYTE *)(v17 - 10) = 0xB8;
110             *(_DWORD *)(v17 - 9) = v19++;
111           }
112           else
113           {
114             *(_BYTE *)(v17 - 7) = 0xB8;
115             *(_DWORD *)(v17 - 6) = v19++;
116           }
117         }
118       }
119       while ( v18 != 0x3000 );
120       goto LABEL_47;
121     }
122   }
```

Disassembly - C:\Users\janusz\Desktop\mlwr.exe - WinDbg:10.0.19041.1 AMD64

Offset: 77aef890

```
77aef875 6f              outs    dx,dword ptr [esi]
77aef876 6c              ins     byte ptr es:[edi],dx
77aef877 007763          add     byte ptr [edi+63h],dh
77aef87a 7374            jae     ntdll32!NtReadFile+0x10 (77aef8f0)
77aef87c 6f              outs    dx,dword ptr [esi]
77aef87d 6d              ins     dword ptr es:[edi],dx
77aef87e 627300          bound   esi,qword ptr [ebx]
77aef881 7763            ja      ntdll32!NtReadFile+0x6 (77aef8e6)
77aef883 7374            jae     ntdll32!NtReadFile+0x19 (77aef8f9)
77aef885 6f              outs    dx,dword ptr [esi]
77aef886 756c            jne     ntdll32!NtReadFile+0x14 (77aef8f4)
77aef888 009090909090    add     byte ptr [eax-6F6F6F70h],dl
77aef88e 90              nop
77aef88f 90              nop
ntdll32!ZwMapUserPhysicalPagesScatter:
77aef890 b800000000      mov     eax,0
77aef895 b90a000000      mov     ecx,0Ah
77aef89a 8d542404        lea     edx,[esp+4]
77aef89e 64ff15c0000000  call    dword ptr fs:[0C0h]
77aef8a5 83c404          add     esp,4
77aef8a8 c20c00          ret     0Ch
77aef8ab 90              nop
ntdll32!NtWaitForSingleObject:
77aef8ac b801000000      mov     eax,1
77aef8b1 b90d000000      mov     ecx,0Dh
77aef8b6 8d542404        lea     edx,[esp+4]
77aef8ba 64ff15c0000000  call    dword ptr fs:[0C0h]
```

Example #2 - remcos (60c07bac07)

… and this is how remcos unhooks:

binary-match first export

```
94    v16 = v26;
95    while ( ++v16 != v27 + v26 )
96    {
97      if ( *(_BYTE *)v16 == 0xB8 && !*(_DWORD *)(v16 + 1) && *(_BYTE *)(v16 + 5) == 0xB9 )
98      {
99        v17 = v16 + 0xA;
100       v18 = 0;
101       v19 = 1;
102       do
103       {
104         ++v18;
105         if ( *(_DWORD *)++v17 == 0x424548D )
106         {
107           if ( *(_WORD *)(v17 - 2) != 0x33C9 && *(_BYTE *)(v17 - 5) == 0xB9 )
108           {
109             *(_BYTE *)(v17 - 10) = 0xB8;
110             *(_DWORD *)(v17 - 9) = v19++;
111           }
112           else
113           {
114             *(_BYTE *)(v17 - 7) = 0xB8;
115             *(_DWORD *)(v17 - 6) = v19++;
116           }
117         }
118       }
119       while ( v18 != 0x3000 );
120       goto LABEL_47;
121     }
122   }
```

Disassembly - C:\Users\janusz\Desktop\mlwr.exe - WinDbg:10.0.19041.1 AMD64

Offset: 77aef890

```
77aef875 6f              outs    dx,dword ptr [esi]
77aef876 6c              ins     byte ptr es:[edi],dx
77aef877 007763          add     byte ptr [edi+63h],dh
77aef87a 7374            jae     ntdll32!NtReadFile+0x10 (77aef8f0)
77aef87c 6f              outs    dx,dword ptr [esi]
77aef87d 6d              ins     dword ptr es:[edi],dx
77aef87e 627300          bound   esi,qword ptr [ebx]
77aef881 7763            ja      ntdll32!NtReadFile+0x6 (77aef8e6)
77aef883 7374            jae     ntdll32!NtReadFile+0x19 (77aef8f9)
77aef885 6f              outs    dx,dword ptr [esi]
77aef886 756c            jne     ntdll32!NtReadFile+0x14 (77aef8f4)
77aef888 009090909090    add     byte ptr [eax-6F6F6F70h],dl
77aef88e 90              nop
77aef88f 90              nop
ntdll32!ZwMapUserPhysicalPagesScatter:
77aef890 b800000000      mov     eax,0
77aef895 b90a000000      mov     ecx,0Ah
77aef89a 8d542404        lea     edx,[esp+4]
77aef89e 64ff15c0000000  call    dword ptr fs:[0C0h]
77aef8a5 83c404          add     esp,4
77aef8a8 c20c00          ret     0Ch
77aef8ab 90              nop
ntdll32!NtWaitForSingleObject:
77aef8ac b801000000      mov     eax,1
77aef8b1 b90d000000      mov     ecx,0Dh
77aef8b6 8d542404        lea     edx,[esp+4]
77aef8ba 64ff15c0000000  call    dword ptr fs:[0C0h]
```

Example #2 - remcos (60c07bac07)

… and this is how remcos unhooks:

```
94    v16 = v26;
95    while ( ++v16 != v27 + v26 )
96    {
97      if ( *(_BYTE *)v16 == 0xB8 && !*(_DWORD *)(v16 + 1) && *(_BYTE *)(v16 + 5) == 0xB9 )
98      {
99        v17 = v16 + 0xA;
100       v18 = 0;
101       v19 = 1;
102       do
103       {
104         ++v18;
105         if ( *(_DWORD *)++v17 == 0x424548D )
106         {
107           if ( *(_WORD *)(v17 - 2) != 0x33C9 && *(_BYTE *)(v17 - 5) == 0xB9 )
108           {
109             *(_BYTE *)(v17 - 10) = 0xB8;
110             *(_DWORD *)(v17 - 9) = v19++;
111           }
112           else
113           {
114             *(_BYTE *)(v17 - 7) = 0xB8;
115             *(_DWORD *)(v17 - 6) = v19++;
116           }
117         }
118       }
119       while ( v18 != 0x3000 );
120       goto LABEL_47;
121     }
122   }
```

“for each export”

```
ntdll32!ZwMapUserPhysicalPagesScatter:
77aef890 b800000000      mov     eax,0
77aef895 b90a000000      mov     ecx,0Ah
77aef89a 8d542404        lea     edx,[esp+4]
77aef89e 64ff15c0000000  call    dword ptr fs:[0C0h]
77aef8a5 83c404          add     esp,4
77aef8a8 c20c00          ret     0Ch
77aef8ab 90              nop
ntdll32!NtWaitForSingleObject:
77aef8ac b801000000      mov     eax,1
77aef8b1 b90d000000      mov     ecx,0Dh
77aef8b6 8d542404        lea     edx,[esp+4]
77aef8ba 64ff15c0000000  call    dword ptr fs:[0C0h]
```

Example #2 - remcos (60c07bac07)

… and this is how remcos unhooks:

```
94    v16 = v26;
95    while ( ++v16 != v27 + v26 )
96    {
97      if ( *(_BYTE *)v16 == 0xB8 && !*(_DWORD *)(v16 + 1) && *(_BYTE *)(v16 + 5) == 0xB9 )
98      {
99        v17 = v16 + 0xA;
100       v18 = 0;
101       v19 = 1;
102       do
103       {
104         ++v18;
105         if ( *(_DWORD *)++v17 == 0x424548D )
106         {
107           if ( *(_WORD *)(v17 - 2) != 0x33C9 && *(_BYTE *)(v17 - 5) == 0xB9 )
108           {
109             *(_BYTE *)(v17 - 10) = 0xB8;
110             *(_DWORD *)(v17 - 9) = v19++;
111           }
112           else
113           {
114             *(_BYTE *)(v17 - 7) = 0xB8;
115             *(_DWORD *)(v17 - 6) = v19++;
116           }
117         }
118       }
119       while ( v18 != 0x3000 );
120       goto LABEL_47;
121     }
122   }
```

override first 5 bytes to ensure we're unhooked

```
ntdll32!ZwMapUserPhysicalPagesScatter:
77aef890 b800000000      mov     eax,0
77aef895 b90a000000      mov     ecx,0Ah
77aef89a 8d542404        lea     edx,[esp+4]
77aef89e 64ff15c0000000  call    dword ptr fs:[0C0h]
77aef8a5 83c404          add     esp,4
77aef8a8 c20c00          ret     0Ch
77aef8ab 90              nop
ntdll32!NtWaitForSingleObject:
77aef8ac b801000000      mov     eax,1
77aef8b1 b90d000000      mov     ecx,0Dh
77aef8b6 8d542404        lea     edx,[esp+4]
77aef8ba 64ff15c0000000  call    dword ptr fs:[0C0h]
```

Example #2 - remcos (60c07bac07)

… and this is how remcos unhooks:

```
 94    v16 = v26;
 95    while ( ++v16 != v27 + v26 )
 96    {
 97      if ( *(_BYTE *)v16 == 0xB8 && !*(_DWORD *)(v16 + 1) && *(_BYTE *)(v16 + 5) == 0xB9 )
 98      {
 99        v17 = v16 + 0xA;
100        v18 = 0;
101        v19 = 1;
102        do
103        {
104          ++v18;
105          if ( *(_DWORD *)++v17 == 0x424548D )
106          {
107            if ( *(_WORD *)(v17 - 2) != 0x33C9 && *(_BYTE *)(v17 - 5) == 0xB9 )
108            {
109              *(_BYTE *)(v17 - 10) = 0xB8;
110              *(_DWORD *)(v17 - 9) = v19++;
111            }
112            else
113            {
114              *(_BYTE *)(v17 - 7) = 0xB8;
115              *(_DWORD *)(v17 - 6) = v19++;
116            }
117          }
118        }
119        while ( v18 != 0x3000 );
120        goto LABEL_47;
121      }
122    }
```

override first 5 bytes to ensure we're unhooked

```
ntdll32!ZwMapUserPhysicalPagesScatter:
77aef890 b800000000      mov     eax,0
77aef895 b90a000000      mov     ecx,0Ah
77aef89a 8d542404        lea     edx,[esp+4]
77aef89e 64ff15c0000000  call    dword ptr fs:[0C0h]
77aef8a5 83c404          add     esp,4
77aef8a8 c20c00          ret     0Ch
77aef8ab 90              nop
ntdll32!NtWaitForSingleObject:
77aef8ac b801000000      mov     eax,1
77aef8b1 b90d000000      mov     ecx,0Dh
77aef8b6 8d542404        lea     edx,[esp+4]
77aef8ba 64ff15c0000000  call    dword ptr fs:[0C0h]
```

Unhooking

Of course you can implement anti$^{(2n - 1)}$-unhooking...

... and they would implement anti$^{(2n)}$-unhooking ...

(Valid for $n \in Z^{+}$)

Example #3 - kronos (6a8419d81f)

Example #3 - kronos (6a8419d81f)

- Banking malware
- Packed x86/x64 binaries
- API hammering

Sample:
[5] https://mwdb.cert.pl/sample/6a8419d81fb645c073439e284a988ab540cd514a933ce2b6ee4b776aa50b50ac

Example #3 - kronos (6a8419d81f)

API hammering, pretty long sequence of operations:

- manipulating registry keys
  `\\REGISTRY\\MACHINE\\Software\\Wow6432Node\\Microsoft\\Windows\\CurrentVersion\\Uninstall\\occidentalconvertors`
- creating directories
- etc.

Example #3 - kronos (6a8419d81f)

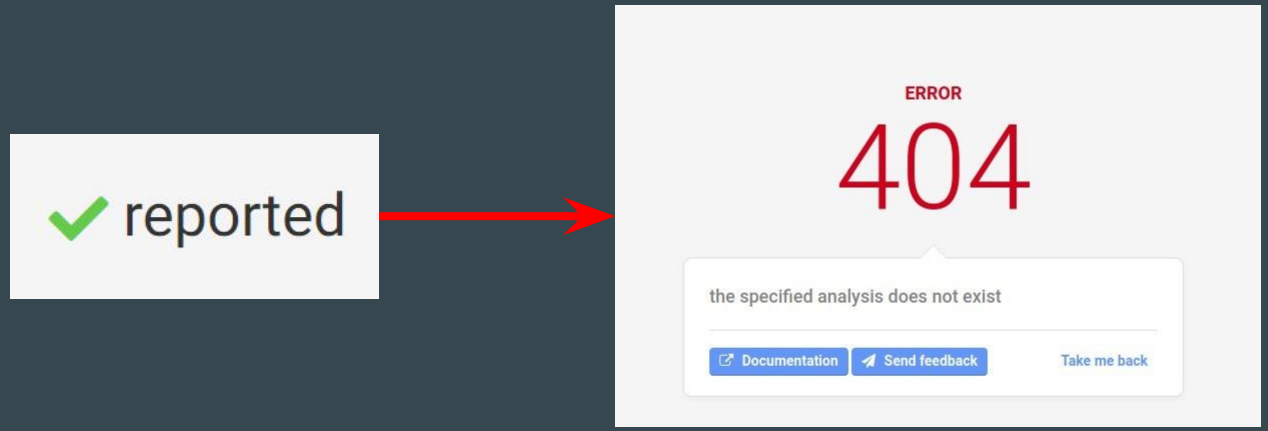
API hammering:

```
$ cat drakmon.log | grep NtCreateKey | grep occidentalconvertors | wc -l
40484

$ cat drakmon.log | grep NtCreateKey | grep occidentalconvertors | head -n1
{
  "Plugin": "regmon",
  "TimeStamp": "1596380139.796501",
  "ProcessName": "\\Device\\HarddiskVolume2\\Users\\janusz\\Desktop\\MALWAR.EXE",
  "UserName": "SessionID",
  "UserId": 1,
  "PID": 1584,
  "PPID": 804,
  "Method": "NtCreateKey",
  "Key":
"\\REGISTRY\\MACHINE\\Software\\Wow6432Node\\Microsoft\\Windows\\CurrentVersion\\Uninstall\\occidentalconvertors"
}
```

Example #3 - kronos (6a8419d81f)

After uploading to **cuckoo.cert.ee**:



Example #3 - kronos (6a8419d81f)

Our old&rusty Cuckoo 1:

```
sie 02 17:08:08 rex python[9179]: 2020-08-02 17:08:08,536 [lib.cuckoo.core.guest]
INFO: Starting analysis on guest (id=m, ip=192.168.122.31)
sie 02 17:10:33 rex python[9179]: 2020-08-02 17:10:33,621 [lib.cuckoo.core.scheduler]
ERROR: Analysis failed: [Errno 10054] An existing connection was forcibly closed by
the remote host
sie 02 17:10:35 rex python[9179]: 2020-08-02 17:10:35,608 [lib.cuckoo.core.scheduler]
INFO: Task #132707: analysis procedure completed
```

(exact reason not known)

Let’s do it on our own

Let’s do it on our own

- ~~user mode~~ (problems already mentioned)
- kernel mode
- hypervisor

Let’s do it on our own

- ~~user mode~~ (problems already mentioned)
- ~~kernel mode~~
- hypervisor

Let’s do it on our own

- ~~user mode~~ (problems already mentioned)
- ~~kernel mode~~
- **hypervisor**

New dynamic unpacking system

- we need something open source to extend it
- must be a pretty decent malware monitor
- … and we will add the memory dump thing

… hypervisor-level monitor? VMI? DRAKVUF?

Virtual Machine Introspection

What is VMI?

- Virtual Machine Introspection

- inspecting VM state using magic programs running purely on host

```
$ vmi-process-list windows7-sp1
Process listing for VM windows7-sp1-x86 (id=7)
[    4] System (struct addr:84aba980)
[  220] smss.exe (struct addr:85a44020)
[  300] csrss.exe (struct addr:85f67a68)
[  336] wininit.exe (struct addr:8601e030)
```

DRAKVUF

What is DRAKVUF?

- blackbox binary analysis system
- “strace” for Virtual Machines

What is DRAKVUF?

- blackbox binary analysis system
- “strace” for Virtual Machines

```
$ drakvuf -d windows7-sp1 ...

[SYSCALL] TIME:1571248115.605033 VCPU:1
CR3:0x56ca5000,"\Device\HarddiskVolume2\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" SessionID:1
ntoskrnl.exe!NtProtectVirtualMemory Arguments: 5
    IN HANDLE ProcessHandle: 0xffffffffffffffff
    INOUT PVOID *BaseAddress: 0x13cd08
    INOUT PSIZE_T RegionSize: 0x13cd10
    IN WIN32_PROTECTION_MASK NewProtectWin32: 0x4
    OUT PULONG OldProtect: 0x13cfb0

[SYSCALL] TIME:1571248171.517430 VCPU:0 ...
```

What next?

DRAKVUF

What we have:

- event tracing on the level of Windows kernel
- stealth - the VM doesn’t see hooks that are applied on the hypervisor level (in a big simplification)

What we need:

- memory dumps
- WinAPI behavioral analysis

Memory dumps

Memory dumps

- we need to dump interesting memory regions
- … at interesting points of run time

- LibVMI features: memory inspection at low level (read/write/interpret guest kernel’s structures)
- DRAKVUF features: hooks on syscalls (and a little bit more)

Heuristics

Hook `NtProtectVirtualMemory(process_handle, base_addr, ...)`:

```
if (process_handle == ~0ULL) {
    char buf[2];
    __read_vm_memory(base_addr, buf, 2);

    if (buf[0] == 'M' && buf[1] == 'Z') {
        __dump_memory(base_addr, "possible binary detected");
    }
}
```

Heuristics

Hook `NtFreeVirtualMemory(process_handle, base_addr, ...)`:

```
if (process_handle == ~0ULL) {
    if (__lookup_pagetable(base_addr, &pte_value) == VMI_SUCCESS) {
        bool pte_valid = !(pte_value & (1UL << 0));
        bool page_writeable = !(pte_value & (1UL << 1));
        bool page_executable = !!(pte_value & (1UL << 63));

        if (pte_valid && page_writeable && page_executable) {
            __dump_memory(base_addr, "free called on RWX memory");
        }
    }
}
```

Memory dumps

How to map a single pointer into a corresponding memory region?

```
__dump_memory(mem_base_address, "possible binary detected");
```

Memory dumps

How to map a single pointer into a corresponding memory region?

```
__dump_memory(mem_base_address, "possible binary detected");
```

→ Look inside Virtual Address Descriptors.

Memory dumps

VAD - Virtual Address Descriptor

```
[1] dump.mem 18:15:32> vad(eprocess=0xfa8002992060)
  VAD            lev start          end            com    type    exe    protect              filename
  -------------- --- -------------- -------------- ------ ------- ------ -------------------- --------
  0xfa8003228310   7         0x10000        0x1ffff       0 Mapped       READWRITE
  0xfa8002ad9440   8         0x20000        0x21fff       0 Mapped       READONLY
  0xfa8002063c80   6         0x30000        0x33fff       0 Mapped       READONLY
  0xfa800149bc70   7         0x40000        0x42fff       0 Mapped       READONLY
  0xfa8002459b10   5         0x50000        0xcffff       7 Private      READWRITE
  0xfa80030088b0   8         0xd0000        0xd0fff       1 Private      READWRITE
  0xfa8001310850   7         0xe0000        0x146fff      0 Mapped       READONLY             C:\Windows\System32\locale.nls
  0xfa8001308290   8         0x150000       0x155fff      0 Mapped       READONLY
  0xfa8003022430   6         0x160000       0x160fff      0 Mapped       READWRITE
  0xfa8002165870   8         0x170000       0x170fff      1 Private      READWRITE
  ...
  0xfa80020076d0   8  0x7fef4020000  0x7fef405ffff        3 Mapped  Exe  EXECUTE_WRITECOPY    C:\Windows\System32\tapi32.dll
  0xfa80016d6d80   6  0x7fef4060000  0x7fef4097fff        2 Mapped  Exe  EXECUTE_WRITECOPY    C:\Windows\System32\WinSCard.dll
  ...
  0xfa80016cbc40   6  0x7fefd020000  0x7fefd036fff        2 Mapped  Exe  EXECUTE_WRITECOPY    C:\Windows\System32\cryptsp.dll
  0xfa8003022a00   7  0x7fefd680000  0x7fefd68efff        2 Mapped  Exe  EXECUTE_WRITECOPY    C:\Windows\System32\cryptbase.dll
```

Memory dumps

What if we don’t have any pointer provided as an argument?

E.g. `NtTerminateProcess` is not memory-related but it’s still interesting to know the caller.

Memory dumps

What if we don’t have any pointer provided as an argument?

E.g. `NtTerminateProcess` is not memory-related but it’s still interesting to know the caller.

→ Perform stack walk.

Memory dumps

Known: current CPU context inside syscall
Unknown: 64 bit stack, 32 bit stack (SYSWOW64)

64 bit: `_KTHREAD->TrapFrame->Rsp`
32 bit: `(WOW_CONTEXT*)(_KTHREAD->Teb->TlsSlots[1] + 4)->Esp/Ebp`

Memory dumps

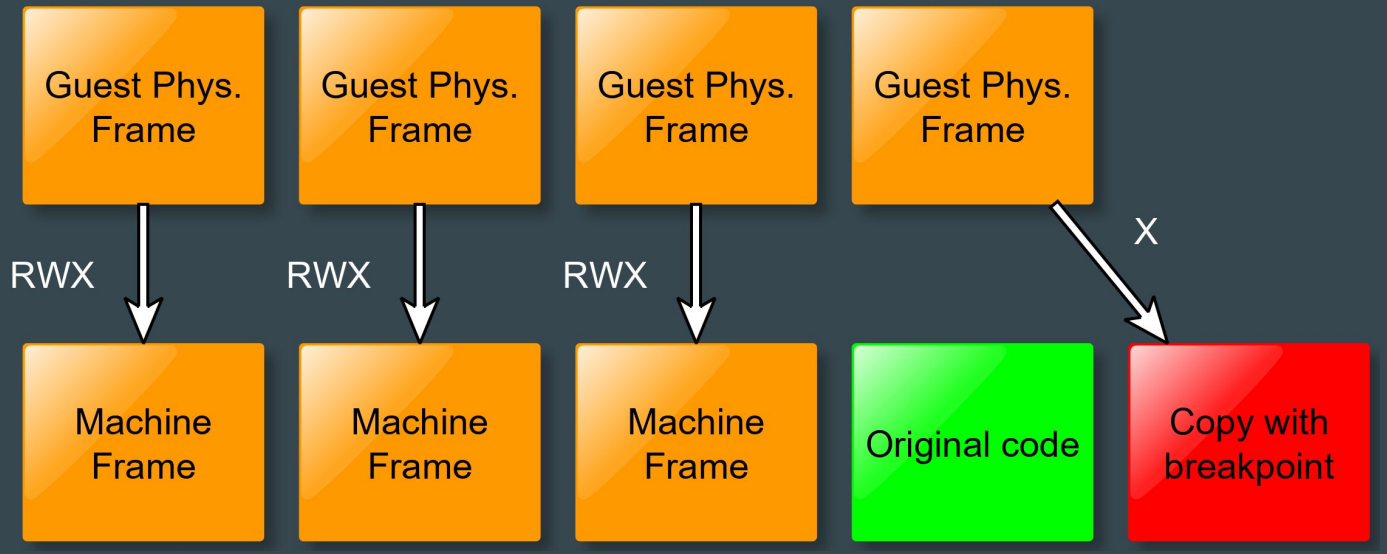
Stack unwinding?

```
for (int i = 0; i < 500; i++) {
    addr_t ptr = *(rsp+i);
    if (__has_mmvad(ptr) && __is_executable_page(ptr))
       __add_stack_entry(ptr);
}
```

Usermode hooking

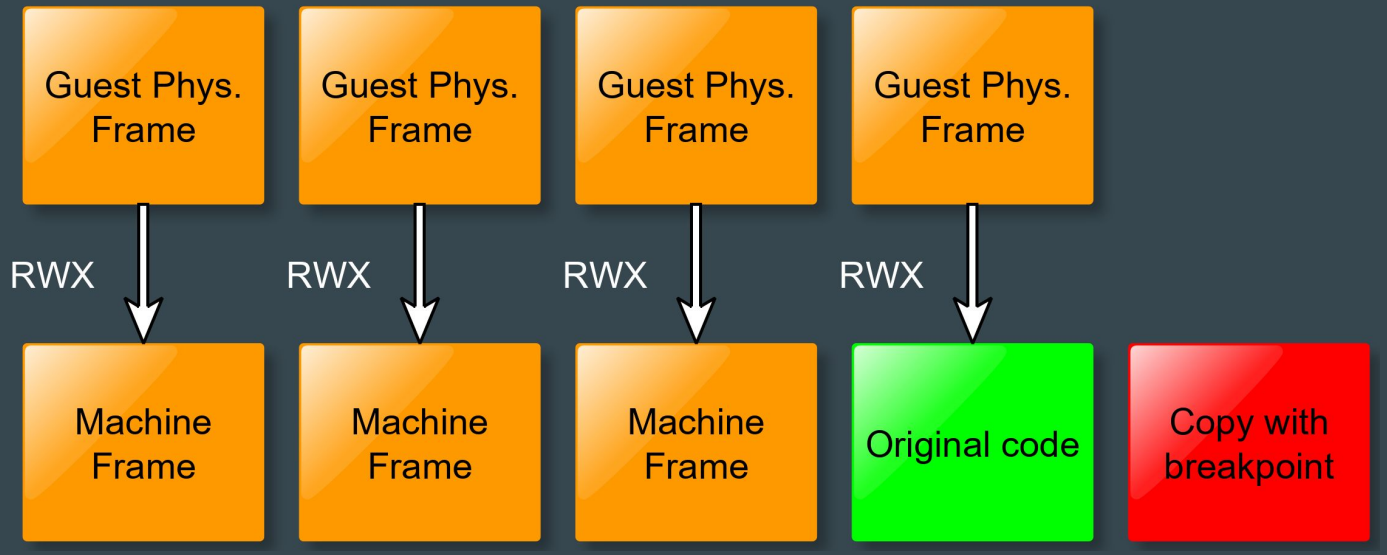
DRAKVUF's hooks (simplified)

Default altp2m view during execution



DRAKVUF's hooks (simplified)

"Normal view" - used only during single-step

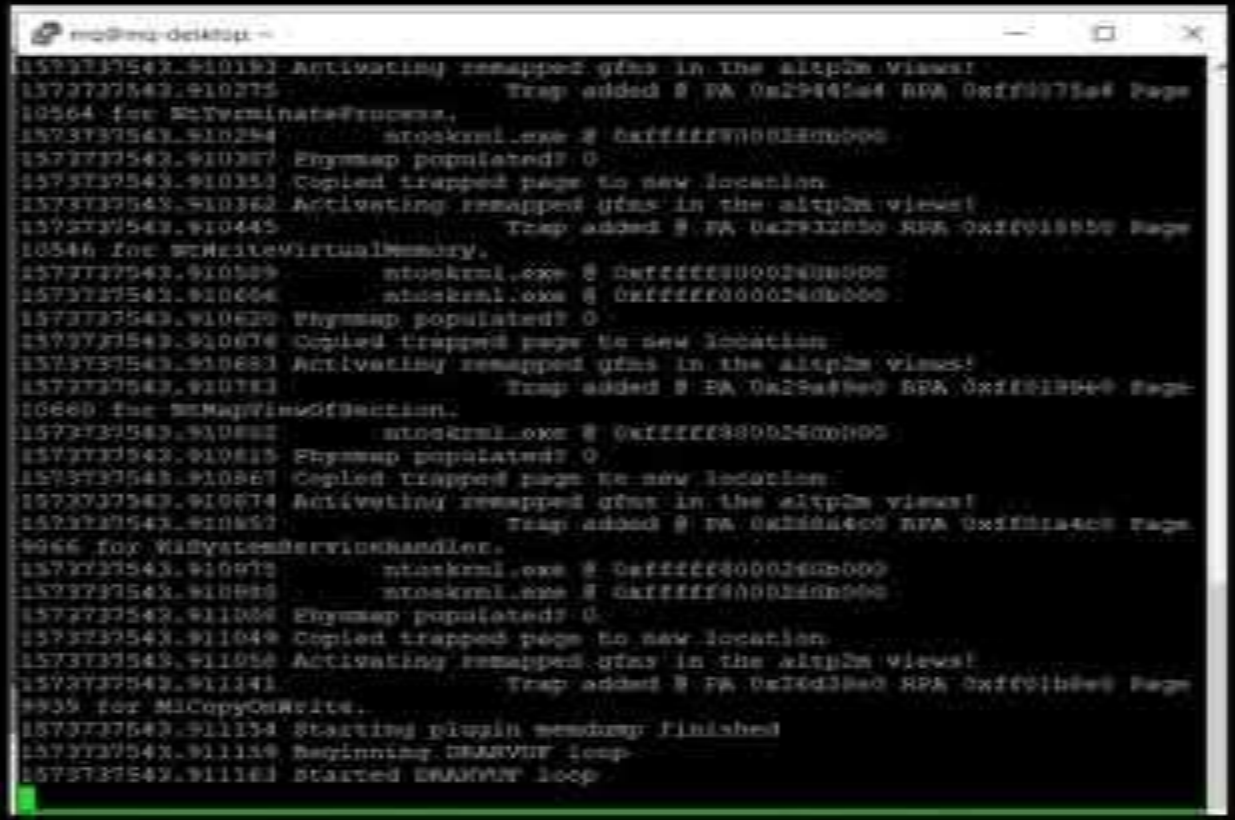


Usermode hooking

But why?

- hooks on syscalls are too low-level for us
- sometimes it is possible to extract some information using some tricks...
- ... but we want to have an universal way
- there are WinAPI functions that are not doing any syscalls at all
- full behavioral analysis!

DRAKVUF Demo #2: Crypto API


1573737543.910193 Activating remapped gfns in the altp2m views!
1573737543.910275 Trap added @ PA 0x29465e4 RPA 0xfff975e4 Page
10564 for NtTerminateProcess.
1573737543.910294 ntoskrnl.exe @ 0xfffff80003800000
1573737543.910307 Physmap populated? 0
1573737543.910353 Copied trapped page to new location
1573737543.910362 Activating remapped gfns in the altp2m views!
1573737543.910445 Trap added @ PA 0x2932050 RPA 0xfff91050 Page
10546 for NtWriteVirtualMemory.
1573737543.910509 ntoskrnl.exe @ 0xfffff80002400000
1573737543.910604 ntoskrnl.exe @ 0xfffff80002400000
1573737543.910620 Physmap populated? 0
1573737543.910674 Copied trapped page to new location
1573737543.910683 Activating remapped gfns in the altp2m views!
1573737543.910783 Trap added @ PA 0x29a89e0 RPA 0xfff91be0 Page
10660 for NtMapViewOfSection.
1573737543.910802 ntoskrnl.exe @ 0xfffff80002400000
1573737543.910815 Physmap populated? 0
1573737543.910867 Copied trapped page to new location
1573737543.910874 Activating remapped gfns in the altp2m views!
1573737543.910957 Trap added @ PA 0x2966a4c0 RPA 0xfff91a4c0 Page
9966 for KiSystemServiceHandler.
1573737543.910973 ntoskrnl.exe @ 0xfffff80002400000
1573737543.910988 ntoskrnl.exe @ 0xfffff80002400000
1573737543.911008 Physmap populated? 0
1573737543.911049 Copied trapped page to new location
1573737543.911058 Activating remapped gfns in the altp2m views!
1573737543.911141 Trap added @ PA 0x2640d3e0 RPA 0xfff91b0e0 Page
9959 for MiCopyOnWrite.
1573737543.911154 Starting plugin memdump finished
1573737543.911159 Beginning DRAKVUF loop
1573737543.911163 Started DRAKVUF loop

Usermode hooking

Which syscalls are issued when a new DLL is loaded?

Usermode hooking

Which syscalls are issued when a new DLL is loaded?

Closest call: `NtMapViewOfSection` / `NtProtectVirtualMemory`

Usermode hooking

Which syscalls are issued when a new DLL is loaded?

Closest call: `NtMapViewOfSection` / `NtProtectVirtualMemory`

DLLs are loaded...

Usermode hooking

Which syscalls are issued when a new DLL is loaded?

Closest call: `NtMapViewOfSection` / `NtProtectVirtualMemory`

DLLs are loaded…
But they don’t exist in the physical memory (yet).

Usermode hooking

DRAKVUF can’t add breakpoint on a memory which is not yet mapped :(
So…

Usermode hooking

Approach #1:

- let’s hook writes to the page tables (PTE)
- when the hook executes => check if our interesting address is now mapped
- if so => place a breakpoint on physical memory

Usermode hooking

Approach #1:

- let’s hook writes to the page tables (PTE)
- when the hook executes => check if our interesting address is now mapped
- if so => place a breakpoint on physical memory

- works!
- pretty complicated code
- very slow

Usermode hooking

Approach #2: cause page faults manually

- override the current RIP with code that would cause page fault, e.g: `mov eax, DWORD [0x12345678]`
- execute a single instruction
- revert everything to the original state (overriden code, CPU registers)

Usermode hooking

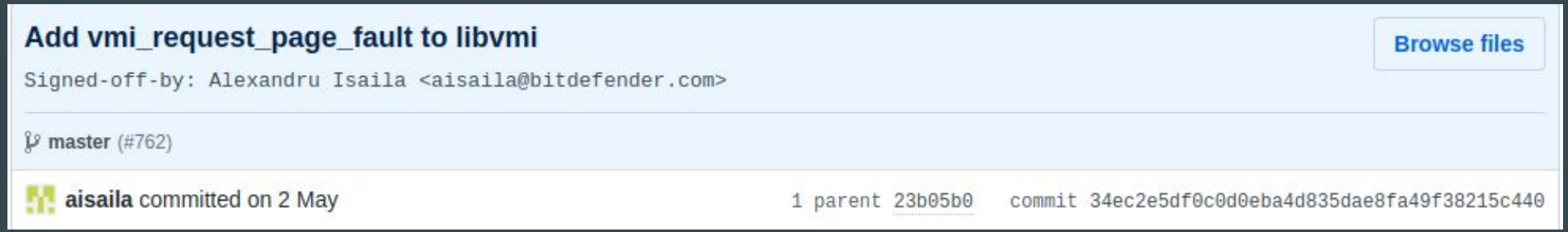
Approach #2: cause page faults manually

- override the current RIP with code that would cause page fault, e.g: `mov eax, DWORD [0x12345678]`
- execute a single instruction
- revert everything to the original state (overriden code, CPU registers)

- fast!
- unstable, invasive

Usermode hooking

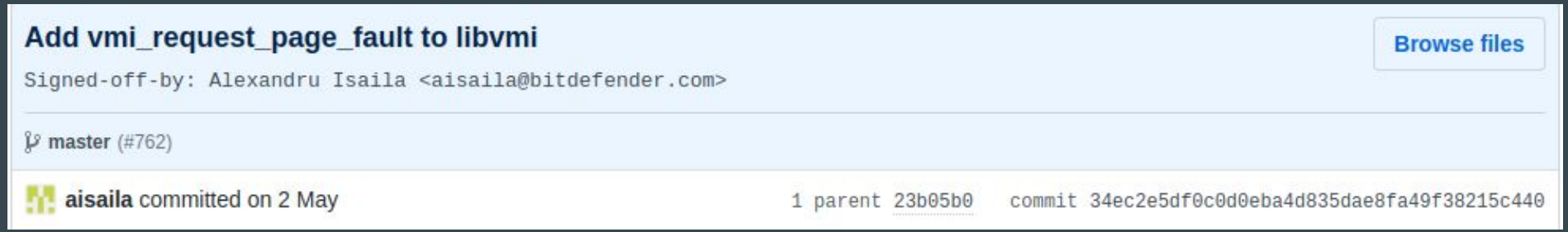
Approach #3:



i.e. inject page fault through VMX from the Xen hypervisor level

Usermode hooking

Approach #3:

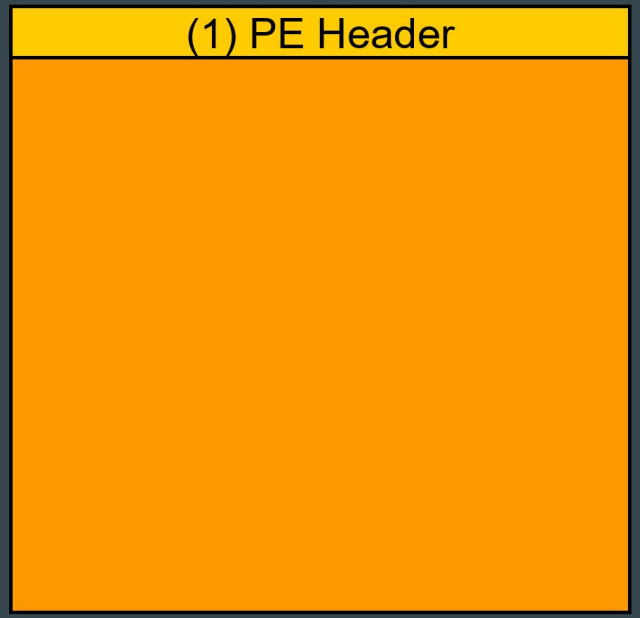


- stable
- fast
- easy (one line of code)
- somebody did job for us :)

Usermode hooking

How to reach the interesting DLL export?

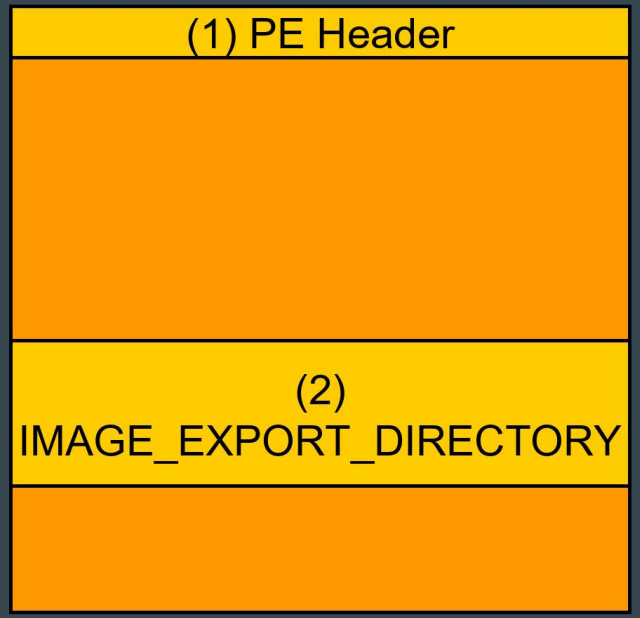
1. Parse the PE header



Usermode hooking

How to reach the interesting DLL export?

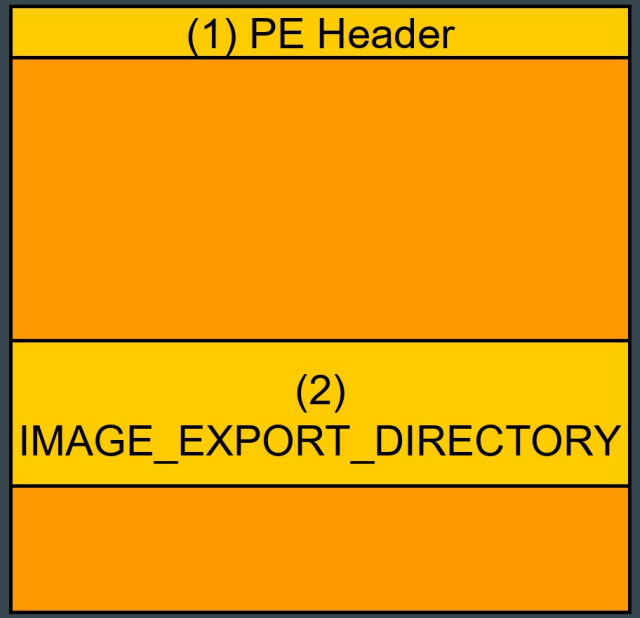
1. Parse the PE header
2. Find image export directory



Usermode hooking

How to reach the interesting DLL export?

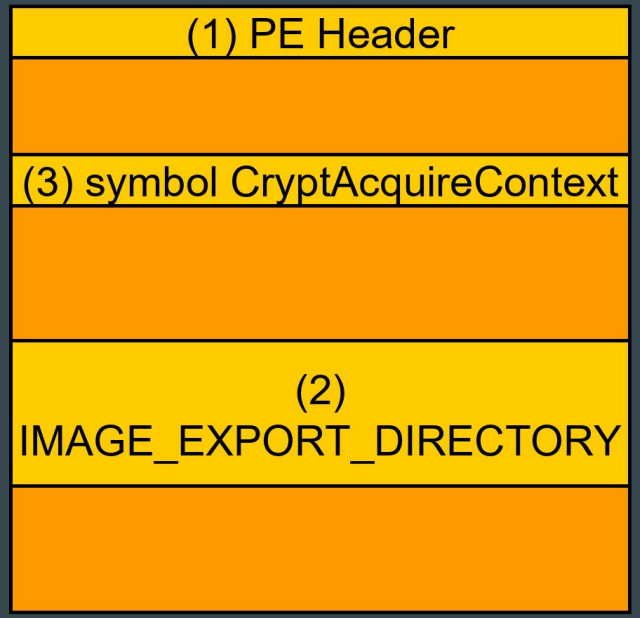
1. Parse the PE header
2. Find image export directory
3. Not readable? Page fault the export directory



Usermode hooking

How to reach the interesting DLL export?

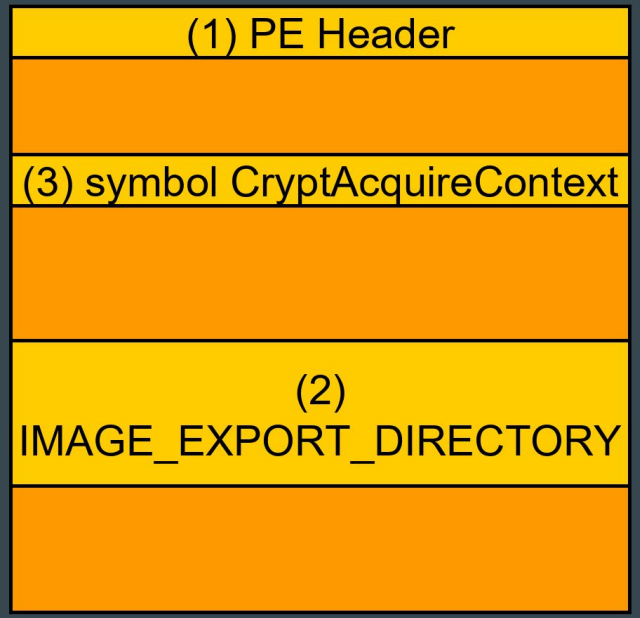
1. Parse the PE header
2. Find image export directory
3. Not readable? Page fault the export directory
4. Find out the RVA of export



Usermode hooking

How to reach the interesting DLL export?

1. Parse the PE header
2. Find image export directory
3. Not readable? Page fault the export directory
4. Find out the RVA of export
5. The first instruction of the exported function is not accessible? Page fault



Usermode hooking

What if the DLL would be (purposely?) corrupted and the pointer to `IMAGE_EXPORT_DIRECTORY` would be invalid?

Our injected page fault would crash the whole Windows system.

Usermode hooking

What if the DLL would be (purposely?) corrupted and the pointer to `IMAGE_EXPORT_DIRECTORY` would be invalid?

Our injected page fault would crash the whole Windows system.

Let’s hook `KiSystemServiceHandler` and pretend that nothing has happened.

Usermode hooking

Break on `KiSystemServiceHandler`:

1. Check if we were recently injecting a page fault into this vCPU. Not “our fault”? Resume the handler and let it cause BSOD.

2. “Our fault”? Emulate ret instruction:
   - Read saved_rip from stack
   - Adjust the CPU context:

```
info->regs->rip = saved_rip;
info->regs->rsp += sizeof(addr_t);
info->regs->rax = EXCEPTION_CONTINUE_EXECUTION;
```

Usermode hooking

Malware could attempt to override it’s own WinAPI function

- DLLs are shared between processes, Copy On Write occurs when they are overridden
- the virtual page is moved to another physical address
- our hooks would not be rewritten to the new page :(

Usermode hooking

Malware could attempt to override it’s own WinAPI function

- DLLs are shared between processes, Copy On Write occurs when they are overridden
- the virtual page is moved to another physical address
- our hooks would not be rewritten to the new page :(

- let’s hook the syscall responsible for CoW: `MiCopyOnWrite`
- let’s rewrite hooks to the new physical page

iexplore.exe - the best test program

iexplore.exe - overriding it’s own DLLs

```
.text:7666FBD1                 ; int __stdcall MessageBoxIndirectA(const MSGBOXPARAMSA *lpmbp)
.text:7666FBD1                 public _MessageBoxIndirectA@4
.text:7666FBD1                 _MessageBoxIndirectA@4 proc near        ; DATA XREF: .text:off_76610570↑o
.text:7666FBD1
.text:7666FBD1                 var_68          = byte ptr -68h
.text:7666FBD1                 MultiByteString = dword ptr -5Ch
.text:7666FBD1                 var_58          = dword ptr -58h
.text:7666FBD1                 var_8           = dword ptr -8
.text:7666FBD1                 P               = dword ptr -4
.text:7666FBD1                 lpmbp           = dword ptr  8
.text:7666FBD1
.text:7666FBD1 8B FF                           mov     edi, edi
.text:7666FBD3 55                              push    ebp
.text:7666FBD4 8B EC                           mov     ebp, esp
.text:7666FBD6 83 EC 68                        sub     esp, 68h
.text:7666FBD9 53                              push    ebx
.text:7666FBDA 56                              push    esi
.text:7666FBDB 57                              push    edi
.text:7666FBDC 33 DB                           xor     ebx, ebx
.text:7666FBDE 6A 60                           push    60h ; '`'       ; Size
.text:7666FBE0 8D 45 98                        lea     eax, [ebp+var_68]
.text:7666FBE3 53                              push    ebx             ; Val
```

iexplore.exe - overriding it’s own DLLs

```
.text:7666FBD1                 ; int __stdcall MessageBoxIndirectA(const MSGBOXPARAMSA *lpmbp)
.text:7666FBD1                 public _MessageBoxIndirectA@4
.text:7666FBD1                 _MessageBoxIndirectA@4 proc near        ; DATA XREF: .text:off_76610570↑o
.text:7666FBD1
.text:7666FBD1                 var_68          = byte ptr -68h
.text:7666FBD1                 MultiByteString = dword ptr -5Ch
.text:7666FBD1                 var_58          = dword ptr -58h
.text:7666FBD1                 var_8           = dword ptr -8
.text:7666FBD1                 P               = dword ptr -4
.text:7666FBD1                 lpmbp           = dword ptr  8
.text:7666FBD1
.text:7666FBD1 8B FF                           mov     edi, edi
.text:7666FBD3 55                              push    ebp
.text:7666FBD4 8B EC                           mov     ebp, esp
.text:7666FBD6 83 EC 68                        sub     esp, 68h
.text:7666FBD9 53                              push    ebx
.text:7666FBDA 56                              push    esi
.text:7666FBDB 57                              push    edi
.text:7666FBDC 33 DB                           xor     ebx, ebx
.text:7666FBDE 6A 60
.text:7666FBE0 8D 45 98
.text:7666FBE3 53
```

```
7666FBCC    90              nop
7666FBCD    90              nop
7666FBCE    90              nop
7666FBCF    90              nop
7666FBD0    90              nop
7666FBD1  ^ E9 34E08AFA     jmp ieframe.70F1DC0A
7666FBD6    83EC 68         sub esp,68
7666FBD9    53              push ebx
7666FBDA    56              push esi
7666FBDB    57              push edi
7666FBDC    33DB            xor ebx,ebx
```

iexplore.exe - overriding it’s own DLLs

user32.dll vs ieframe.dll

```
int __stdcall MessageBoxIndirectA(const MSGBOXPARAMSA *lpmbp)
{
  int v2; // edi
  char v3; // [esp+Ch] [ebp-68h]
  CHAR *MultiByteString; // [esp+18h] [ebp-5Ch]
  CHAR *v5; // [esp+1Ch] [ebp-58h]
  PVOID v6; // [esp+6Ch] [ebp-8h]
  PVOID P; // [esp+70h] [ebp-4h]

  P = 0;
  v6 = 0;
  memset(&v3, 0, 0x60u);
  qmemcpy(&v3, lpmbp, 0x28u);
  if ( (unsigned int)MultiByteString & 0xFFFF0000 )
  {
    if ( !MBToWCSEx(0, MultiByteString, 0xFFFFFFFF, (int)&P, -1, 1) )
      return 0;
    MultiByteString = (CHAR *)P;
  }
  if ( (unsigned int)v5 & 0xFFFF0000 )
  {
    if ( !MBToWCSEx(0, v5, 0xFFFFFFFF, (int)&v6, -1, 1) )
    {
      RtlFreeHeap(pUserHeap, 0, P);
      return 0;
    }
    v5 = (CHAR *)v6;
  }
  v2 = MessageBoxWorker(&v3);
  if ( P )
    RtlFreeHeap(pUserHeap, 0, P);
  if ( v6 )
    RtlFreeHeap(pUserHeap, 0, v6);
  return v2;
}
```

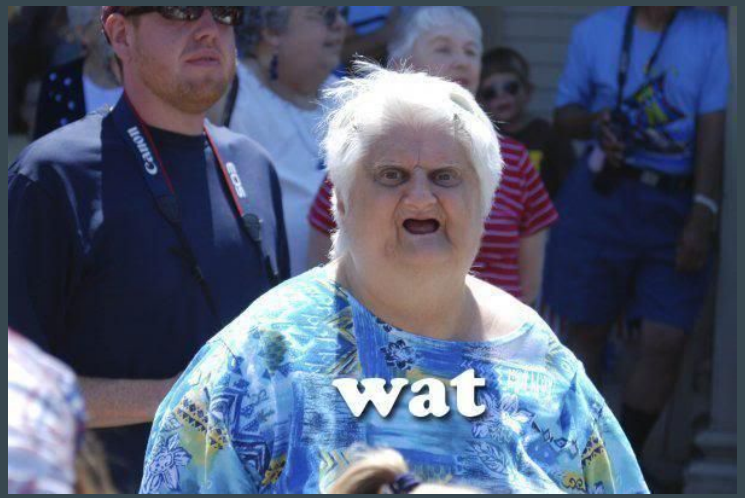
```
int __stdcall Detour_MessageBoxIndirectA(const struct tagMSGBOXPARAMSA *a1)
{
  int v1; // ebx
  char v3; // [esp+8h] [ebp-2Ch]
  HWND hWnd; // [esp+Ch] [ebp-28h]
  struct IEUserBroker *v5; // [esp+30h] [ebp-4h]

  v1 = 0;
  if ( a1 )
  {
    qmemcpy(&v3, a1, 0x28u);
    if ( SuppressDialog(&hWnd, 1u) >= 0
      && UnifiedFrameAware_AcquireModalDialogLockAndParent(hWnd, (int)&v5, (int)&hWnd, (int)&a1) >= 0 )
    {
      v1 = dword_71066B40(&v3);
      UnifiedFrameAware_ReleaseModalDialogLockAndParent(v5, hWnd, (char)a1);
    }
  }
  return v1;
}
```

iexplore.exe - overriding it’s own DLLs

DLLs overridden by IE:

- `comdlg32.dll`
- `ole32.dll`
- `oleaut32.dll`
- `user32.dll`
- `comctl32.dll`



DRAKVUF Demo


[HERBST] Current money 20000
[HERBST] Current money 15000
[HERBST] Current money 17000
[HERBST] Current money 19000

DRAKVUF Sandbox

DRAKVUF Sandbox

Wrapper for DRAKVUF Engine with:

- web interface
- easy installation
- sample queueing
- … much more coming soon!

DRAKVUF
Sandbox

SANDBOX

Upload sample

Analyses

ANALYSIS

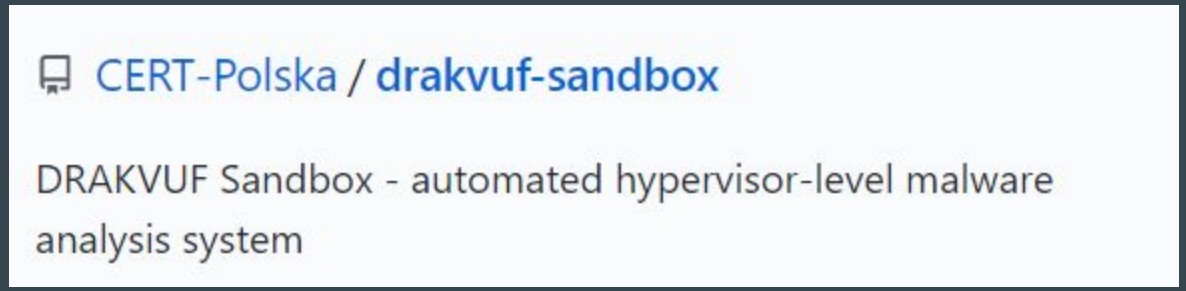
Report

API calls

1594640303.923439	LdrGetProcedureAddress	0x77670000 0x1af220 0x0 0x1af248
1594640303.923658	LdrGetProcedureAddress	0x77670000 0x1af220 0x0 0x1af248
1594640303.953502	WriteConsoleW	0x7 0x2823f0 0x25 0x1af650 0x0
1594640303.953871	LdrLoadDll	0x27f4e0 0x1af670 0x1af628:"dhcpcsvc.dll" 0x1af688
1594640303.954152	LdrGetProcedureAddress	0x7fefbc00000 0x1af650 0x0 0x1af678
1594640303.965090	LdrLoadDll	0x27f4e0 0x1af050 0x1af008:"dhcpcsvc6.DLL" 0x1af068
1594640303.965328	LdrGetProcedureAddress	0x7fefbc80000 0x1af080 0x0 0x1af0a8
1594640303.965545	LdrLoadDll	0x27f4e0 0x1aed70 0x1aed28:"IPHLPAPI.DLL" 0x1aed88
1594640303.965811	LdrGetProcedureAddress	0x7fefbe00000 0x1aeda0 0x0 0x1aedc8
1594640303.966066	LdrLoadDll	0x27f4e0 0x1ae6f0 0x1ae6a8:"rpcrt4.dll" 0x1ae708

GitHub project

Fully open-source and free ;)



Intel Processor Trace (coming soon)

#xen-devel

15:22 <andyhhp__> oh wow - we've got Cert.pl implementing a VM feature which we couldn't even perusade Intel to do

15:22 <andyhhp__> this is going to be interesting

Intel Processor Trace

Summary

GitHub

- DRAKVUF
  [6] https://github.com/tklengyel/drakvuf

- DRAKVUF Sandbox
  [7] https://github.com/CERT-Polska/drakvuf-sandbox

- LibVMI
  [8] https://github.com/libvmi/libvmi

Kudos

- **CERT.PL Reverse Engineers - nazywam, psrok1, msm**
  → for many important remarks and hints about malware monitoring

- **CERT.PL - BonusPlay, chivay, konstantyc**
  → further development of DRAKVUF/DRAKVUF Sandbox

Kudos

- **Maciej “mak” Kotowicz**
  → for providing many good heurstics for memory dumping (and some hints about them)

- **Tamas K. Lengyel**
  → a lots of helpful remarks during our research
  → creator/maintainer of DRAKVUF project na GitHub

Self-advertisement

We share a lot of data about malware.
White-hat external researchers could apply at:

mwdb.cert.pl


CERT.PL