

MALWARE: ANTI-FORENSICS

BHACK 2018



ALEXANDRE BORGES



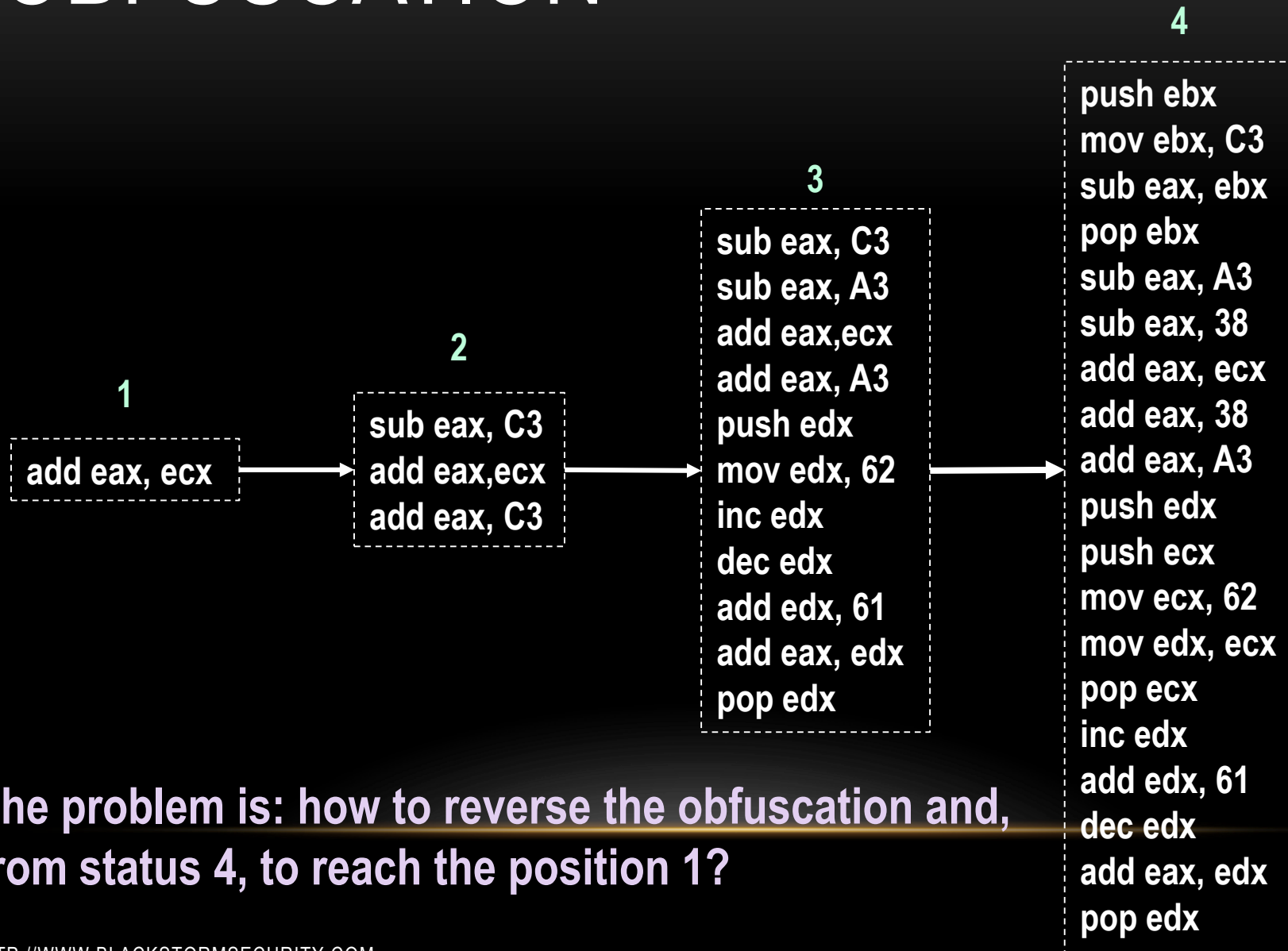
TOC:

- ❖ Obfuscation
- ❖ Emulation
- ❖ Anti-Reversing
- ❖ Anti-VM

- ✓ Malware and Security Researcher.
- ✓ Speaker at DEFCON USA 2018
- ✓ Consultant, Instructor and Speaker on Malware Analysis, Memory Analysis, Digital Forensics and Rookits.
- ✓ Reviewer member of the The Journal of Digital Forensics, Security and Law.
- ✓ Referee on Digital Investigation: The International Journal of Digital Forensics & Incident Response
- ✓ Member of Digital Law and Compliance Committee at OAB/SP
- ✓ Instructor at Oracle and (ISC)2. Ex-instructor at Symantec.

OBFUSCATION

OBFUSCATION



The problem is: how to reverse the obfuscation and, from status 4, to reach the position 1?

OBFUSCATION

- ✓ There are many ways to handle obfuscation and one of them is by using **METASM**, which:
 - ✓ It is written in Ruby.
 - ✓ Works as:
 - ✓ **a disassembler**
 - ✓ **assembler**
 - ✓ **debugger**
 - ✓ **linker and compiler.**
 - ✓ However, it does not work with intermediate language.
 - ✓ It is able to **separate control and data flows.**

OBFUSCATION

✓ Installing METASM is simple:

✓ `root@kali:~/programs# git clone
https://github.com/jjyg/metasm.git`

✓ `root@kali:~/programs# cd metasm/`

✓ `root@kali:~/programs/metasm# make`

✓ `root@kali:~/programs/metasm# make all`

✓ Include the following line into **.bashrc file** to indicate the Metasm directory installation:

✓ `export RUBYLIB=$RUBYLIB:~/programs/metasm`

OBFUSCATION

```
#!/usr/bin/env ruby
#

require "metasm"
include Metasm

ourcode = Metasm::Shellcode.assemble(Metasm::Ia32.new, <<EOB)
entry:
  push ebx
  mov ebx, 0xc3
  sub eax, ebx
  pop ebx
  sub eax, 0xa3
  sub eax, 0x38
  add eax, ecx
  add eax, 0x38
  add eax, 0xa3
  push edx
  push ecx
  mov ecx, 0x62
  mov edx, ecx
  pop ecx
  inc edx
  add edx, 0x61
  dec edx
  add eax, edx
  pop edx
  jmp eax
```

EOB

✓ Possible values:

- ✓ PowerPC
- ✓ Ia32
- ✓ x86_64
- ✓ MachO

This instruction was inserted to make the eax register evaluation easier. ☺

OBFUSCATION

```
asmcode = mycode.init_disassembler
asmcode.disassemble(0)
bhack = asmcode.di_at(0).block
puts "\n[$$$] Our BHACK 2018 test code:\n "
puts bhack.list

bhack.list.each{|aborges|
  puts "\n[###] #{aborges.instruction}"
  back = aborges.backtrace_binding()
  puts "BHACK 2018 data flow follows below:\n"
  back.each{|key, value| puts " * #{key} => #{value}"}

  if aborges.opcode.props[:setip]
    puts "\nBHACK 2018 control flow follows below:\n"
    puts " * #{asmcode.get_xrefs_x(aborges)}"
  end
}

asmcode2 = mycode.init_disassembler
asmcode2.disassemble(0)

dd = asmcode2.block_at(0)
final = asmcode2.get_xrefs_x(dd.list.last).first
puts "\n[+] final output: #{final}"
```

OBFUSCATION

```
values = asmcode2.backtrace(final, dd.list.last.address, {:log => bt_log = [] , :include_start => true})
bt_log.each{|entry|
  case type = entry.first
  when :start
    entry, expr, addr = entry
    puts "[start] Here is the sequence of expression evaluations #{expr} from 0x#{addr.to_s(16)}\n"

  when :di
    entry, to, from, instr = entry
    puts "[new update] instr #{instr},\n --> updating expression once a gain from #{from} to #{to}\n"

  when :found
    entry, final = entry
    puts "[found] possible value:#{final.first}\n"

  when :up
    entry, to, from, addr_down, addr_up = entry
    puts "[up] addr 0x#{addr_down.to_s(16)} -> 0x#{addr_up.to_s(16)}\n"

  end
}
```

```
effective = bt_log.select{|y| y.first==:di}.map{|y| y[3]}.reverse
puts "\nThe effective instructions are:\n\n"
puts effective
```

ALEXANDRE BORGES – IT IS NOT ALLOWED TO COPY OR REPRODUCE THIS SLIDE.

OBFUSCATION

```
root@kali:~/programs/metasm# ./bhack2018.rb
```

```
[$$$] Our BHACK 2018 test code:
```

```
0 push ebx
1 mov ebx, 0c3h
6 sub eax, ebx
8 pop ebx
9 sub eax, 0a3h
0eh sub eax, 38h
11h add eax, ecx
13h add eax, 38h
16h add eax, 0a3h
1bh push edx
1ch push ecx
1dh mov ecx, 62h
22h mov edx, ecx
24h pop ecx
25h inc edx
26h add edx, 61h
29h dec edx
2ah add eax, edx
2ch pop edx
2dh jmp eax
```

OBFUSCATION

```
[###] push ebx
```

```
BHACK 2018 data flow follows below:
```

```
* esp => esp-4  
* dword ptr [esp] => ebx
```

```
[###] mov ebx, 0c3h
```

```
BHACK 2018 data flow follows below:
```

```
* ebx => 0c3h
```

```
[###] sub eax, ebx
```

```
BHACK 2018 data flow follows below:
```

```
* eax => eax-ebx  
* eflag_z => (((eax&0xffffffff)-(ebx&0xffffffff))&0xffffffff)==0  
* eflag_s => (((eax&0xffffffff)-(ebx&0xffffffff))&0xffffffff)>>1fh)!=0  
* eflag_c => (eax&0xffffffff)<(ebx&0xffffffff)  
* eflag_o => (((eax&0xffffffff)>>1fh)!=0)==(!(((ebx&0xffffffff)>>1fh)!=0))&&(((eax&0xffffffff)>>1fh)!=0)!=((((eax&0xffffffff)-(ebx&0xffffffff))&0xffffffff)>>1fh)!=0))
```

Metasm is an excellent tool because the entire context is kept, even when we are working on instruction virtualizers such as Virtual Protect (VMP) 😊

OFUSCATION

```
[+] final output: eax
[start] Here is the sequence of expression evaluations  eax from 0x2d
[new update] instr 2ah add eax, edx,
--> updating expression once again from eax to eax+edx
[new update] instr 29h dec edx,
--> updating expression once again from eax+edx to eax+edx-1
[new update] instr 26h add edx, 61h,
--> updating expression once again from eax+edx-1 to eax+edx+60h
[new update] instr 25h inc edx,
--> updating expression once again from eax+edx+60h to eax+edx+61h
[new update] instr 22h mov edx, ecx,
--> updating expression once again from eax+edx+61h to eax+ecx+61h
[new update] instr 1dh mov ecx, 62h,
--> updating expression once again from eax+ecx+61h to eax+0c3h
[new update] instr 16h add eax, 0a3h,
--> updating expression once again from eax+0c3h to eax+166h
[new update] instr 13h add eax, 38h,
--> updating expression once again from eax+166h to eax+19eh
[new update] instr 11h add eax, ecx,
--> updating expression once again from eax+19eh to eax+ecx+19eh
[new update] instr 0eh sub eax, 38h,
--> updating expression once again from eax+ecx+19eh to eax+ecx+166h
[new update] instr 9 sub eax, 0a3h,
--> updating expression once again from eax+ecx+166h to eax+ecx+0c3h
[new update] instr 6 sub eax, ebx,
--> updating expression once again from eax+ecx+0c3h to eax-ebx+ecx+0c3h
[new update] instr 1 mov ebx, 0c3h,
--> updating expression once again from eax-ebx+ecx+0c3h to eax+ecx
```

OFUSCATION

The **effective instructions** are:

```
1 mov ebx, 0c3h
6 sub eax, ebx
9 sub eax, 0a3h
0eh sub eax, 38h
11h add eax, ecx
13h add eax, 38h
16h add eax, 0a3h
1dh mov ecx, 62h
22h mov edx, ecx
25h inc edx
26h add edx, 61h
29h dec edx
2ah add eax, edx
```

We can realize that the number of instructions is smaller than the initial set because many of them were not useful to compose the final result. Great! 😊

EMULATION

EMULATION

- ✓ **Emulation** is another excellent method to solve many practical reverse engineering problems.
- ✓ Focusing on a easy approach to handle our small case, it is possible to use **uEmu + Keystone Engine**. 😊
- ✓ First, install the fantastic **Keystone Engine** as shown below:
 - ✓ `root@kali:~/Desktop# wget https://github.com/keystone-engine/keystone/archive/0.9.1.tar.gz`
 - ✓ `root@kali:~/programs# cp /root/Desktop/keystone-0.9.1.tar.gz .`
 - ✓ `root@kali:~/programs# tar -zxvf keystone-0.9.1.tar.gz`
 - ✓ `root@kali:~/programs/keystone-0.9.1# apt-get install cmake`

EMULATION

- ✓ `root@kali:~/programs/keystone-0.9.1# mkdir build`
- ✓ `root@kali:~/programs/keystone-0.9.1# cd build/`
- ✓ `root@kali:~/programs/keystone-0.9.1/build# apt-get install time`
- ✓ `root@kali:~/programs/keystone-0.9.1/build# ../make-share.sh`
- ✓ `root@kali:~/programs/keystone-0.9.1/build# make install`
- ✓ `root@kali:~/programs/keystone-0.9.1/build# ldconfig`
- ✓ `root@kali:~/programs/keystone-0.9.1/build# tail -3 /root/.bashrc`
- ✓ `export PATH=$PATH:/root/programs/phantomjs-2.1.1-linux-x86_64/bin:/usr/local/bin/kstool`
- ✓ `export RUBYLIB=$RUBYLIB:~/programs/metasm`
- ✓ `export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib`

EMULATION

```
root@kali:~/programs/bhack_2018# more bhack2018.c
```

```
#include <stdio.h>
```

```
#include <keystone/keystone.h>
```

```
#define BHACK "push ebx; mov ebx, 0xc3; sub eax, ebx; pop ebx; sub eax, 0xa3; sub e  
ax, 0x38; add eax, ecx; add eax, 0x38; add eax, 0xa3; push edx; push ecx; mov ecx,  
0x62; mov edx, ecx; pop ecx; inc edx; add edx, 0x61; dec edx; add eax, edx; pop edx  
"
```

```
int main(int argc, char **argv)  
{
```

```
    ks_engine *keyeng;
```

```
    ks_err keyerr = KS_ERR_ARCH;
```

```
    size_t count;
```

```
    unsigned char *encode;
```

```
    size_t size;
```

```
    keyerr = ks_open(KS_ARCH_X86, KS_MODE_32, &keyeng);
```

```
    if (keyerr != KS_ERR_OK) {
```

```
        printf("ERROR: A fail occurred while calling ks_open(), quit\n");
```

```
        return -1;
```

```
    }
```

All instructions from our original problem.

Creating a Keystone engine for x86.

EMULATION

```
if (ks_asm(keyeng, BHACK, 0, &encode, &size, &count)) {  
    printf("ERROR: A fail has occured while calling ks_asm() with count = %lu,  
error code = %u\n", count, ks_errno(keyeng));  
} else {  
    size_t i;  
  
    for (i = 0; i < size; i++) {  
        printf("%02x ", encode[i]);  
    }  
  
    // printf ("\n\n");  
  
    // for (i = 0; i < size; i++) {  
    //     printf("\x%02x", encode[i]);  
    // }  
  
    //printf("\n\n");  
}  
  
ks_free(encode);  
ks_close(keyeng);  
  
return 0;  
}
```

Assembling our instructions using Keystone.

EMULATION

```
root@kali:~/programs/bhack_2018# more Makefile
```

```
.PHONY: all clean
```

```
KEYSTONE_LDFLAGS = -lkeystone -lstdc++ -lm
```

```
all:
```

```
    ${CC} -o bhack2018 bhack2018.c
```

```
    ${KEYSTONE_LDFLAGS}
```

```
clean:
```

```
    rm -rf *.o bhack2018
```

EMULATION

```
root@kali:~/programs/bhack_2018# make
```

```
cc -o bhack2018 bhack2018.c -lkeystone -lstdc++ -lm
```

```
root@kali:~/programs/bhack_2018# ./bhack2018
```

```
53 bb c3 00 00 00 29 d8 5b 2d a3 00 00 00 83 e8 38 01 c8 83 c0 38 05 a3 00 00 00 52
51 b9 62 00 00 00 89 ca 59 42 83 c2 61 4a 01 d0 5a
```

```
root@kali:~/programs/bhack_2018# ./bhack2018 | xxd -r -p - > output.bin
```

```
root@kali:~/programs/bhack_2018# hexdump -C output.bin
```

```
00000000 53 bb c3 00 00 00 29 d8 5b 2d a3 00 00 00 83 e8 |S.....).[-.....|
00000010 38 01 c8 83 c0 38 05 a3 00 00 00 52 51 b9 62 00 |8....8.....RQ.b.|
00000020 00 00 89 ca 59 42 83 c2 61 4a 01 d0 5a          |....YB..aJ..Z|
0000002d
```

EMULATION

`apt-get install libcapstone3 libcapstone-dev`

```
#include <inttypes.h>
#include <capstone/capstone.h>

#define CODE "\x53\xbb\xc3\x00\x00\x00\x29\xd8\x5b\x2d\xa3\x00\x00\x00\x83\x
\xe8\x38\x01\xc8\x83\xc0\x38\x05\xa3\x00\x00\x00\x52\x51\xb9\x62\x00\x00\x
00\x89\xca\x59\x42\x83\xc2\x61\x4a\x01\xd0\x5a"

int main(void)
{
    csh cs_handle;
    cs_insn *instruction;
    size_t count;

    if (cs_open(CS_ARCH_X86, CS_MODE_32, &cs_handle) != CS_ERR_OK)
        return -1;

    count = cs_disasm(cs_handle, CODE, sizeof(CODE)-1, 0x0001, 0, &ins
truction);
    if (count > 0) {
        size_t j;
        for (j = 0; j < count; j++) {
            printf("0x%"PRIx32":\t%s\t\t%s\n", instruction[j].
address, instruction[j].mnemonic, instruction[j].op_str);
        }

        cs_free(instruction, count);
    } else
        printf("Error: It's happened an error during the disassemb
ling!\n");

    cs_close(&cs_handle);

    return 0;
}
```



Checking whether our hexcode can be reverted to initial the assembly 21

EMULATION

```
root@kali:~/programs/bhack_2018/capstone# more Makefile
.PHONY: all clean

CAPSTONE_LDFLAGS = -lcapstone -lstdc++ -lm

all:
    ${CC} -o bhack2018_rev bhack2018_rev.c ${CAPSTONE_LDFLAGS}

clean:
    rm -rf *.o bhack2018_rev
root@kali:~/programs/bhack_2018/capstone#
root@kali:~/programs/bhack_2018/capstone# make
cc -o bhack2018_rev bhack2018_rev.c -lcapstone -lstdc++ -lm
root@kali:~/programs/bhack_2018/capstone#
root@kali:~/programs/bhack_2018/capstone# ./bhack2018_rev
0x1:  push    ebx
0x2:  mov     ebx, 0xc3
0x7:  sub     eax, ebx
0x9:  pop     ebx
0xa:  sub     eax, 0xa3
0xf:  sub     eax, 0x38
0x12: add     eax, ecx
0x14: add     eax, 0x38
0x17: add     eax, 0xa3
0x1c: push    edx
0x1d: push    ecx
0x1e: mov     ecx, 0x62
0x23: mov     edx, ecx
0x25: pop     ecx
0x26: inc     edx
0x27: add     edx, 0x61
0x2a: dec     edx
0x2b: add     eax, edx
0x2d: pop     edx
```

It is perfect!

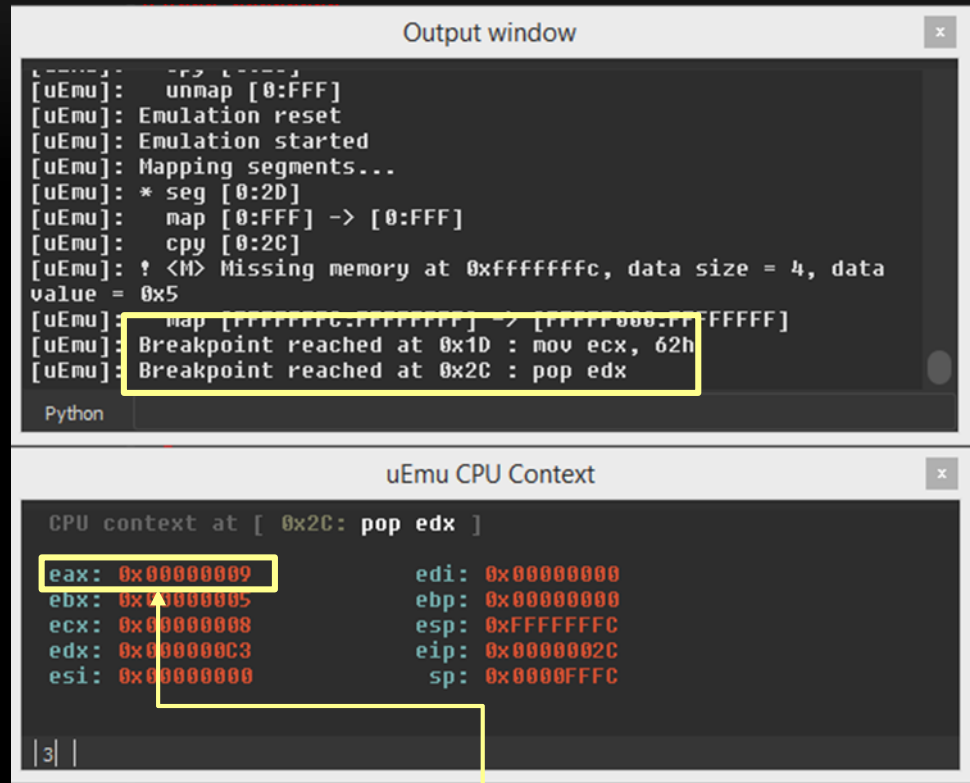
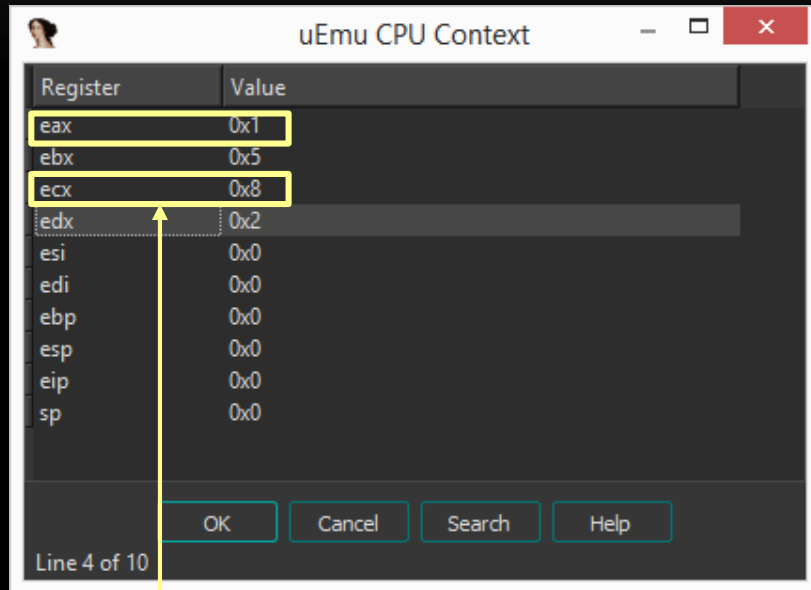
EMULATION

```
seg000:00000000 ; -----
seg000:00000000 ; File Name   : C:\Users\Administrador\Downloads\output.bin
seg000:00000000 ; Format      : Binary file
seg000:00000000 ; Base Address: 0000h Range: 0000h - 0020h Loaded length: 0020h
seg000:00000000
seg000:00000000 .686p
seg000:00000000 .mmx
seg000:00000000 .model flat
seg000:00000000 ; =====
seg000:00000000 ; Segment type: Pure code
seg000:00000000 seg000      segment byte public 'CODE' use32
seg000:00000000                assume cs:seg000
seg000:00000000                assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
seg000:00000000                push     ebx
seg000:00000000                mov      ebx, 0C3h
seg000:00000001                sub      eax, ebx
seg000:00000006                pop      ebx
seg000:00000008                sub      eax, 0A3h
seg000:00000009                sub      eax, 38h
seg000:0000000E                add      eax, ecx
seg000:00000011                add      eax, 38h
seg000:00000013                add      eax, 0A3h
seg000:00000016                push     edx
seg000:0000001B                push     ecx
seg000:0000001C                mov      ecx, 62h
seg000:0000001D                mov      edx, ecx
seg000:00000022                pop      ecx
seg000:00000024                inc      edx
seg000:00000025                add      edx, 61h
seg000:00000026                dec      edx
seg000:00000029                add      eax, edx
seg000:0000002A                pop      edx
seg000:0000002C                ends
seg000:0000002C                end
```

<https://github.com/alexhude/uEmu>

Our instructions perfectly assembled by the Keystone. 😊

EMULATION



According to METASM, but emulating the code, it is the expected result, isn't it?! 😊

EMULATION

✓ INSTALLING MIASM:

- ✓ `git clone https://github.com/serpilliere/elfesteem.git elfesteem`
- ✓ `cd elfesteem/`
- ✓ `python setup.py build`
- ✓ `python setup.py install`
- ✓ `apt-get install clang`
- ✓ `apt-get remove libtcc-dev`
- ✓ `apt-get install llvm`
- ✓ `cd ..`
- ✓ `git clone http://repo.or.cz/tinycc.git`
- ✓ `cd tinycc/`
- ✓ `git checkout release_0_9_26`

EMULATION

- ✓ `./configure --disable-static`
- ✓ `make`
- ✓ `make install`
- ✓ `pip install llvmlite`
- ✓ `apt-get install z3`
- ✓ `apt-get install python-pycparser`
- ✓ `git clone https://github.com/cea-sec/miasm.git`
- ✓ `root@kali:~/programs/miasm# python setup.py build`
- ✓ `root@kali:~/programs/miasm# python setup.py install`
- ✓ `root@kali:~/programs/miasm/test# python test_all.py`
- ✓ `apt-get install graphviz`
- ✓ `apt-get install xdot`
- ✓ `python /root/programs/miasm/example/disasm/full.py -m x86_32 shellcode.bin`

EMULATION

It is only a simple test with a random shellcode to make sure that MIASM is working (it is not related to our initial obfuscation problem). 😊

```
loc_0000000000000000
00000000 XOR      BYTE PTR [EAX], DH
00000002 AND      BYTE PTR [EAX], DH
00000004 XOR      AL, 0x20
00000006 XOR      AL, BYTE PTR [EBX + 0x20]
00000009 XOR      BYTE PTR [EAX], DH
0000000B AND      BYTE PTR [ECX + EAX * 0x2], DH
0000000E AND      BYTE PTR [EAX], DH
00000010 INC      EBX
00000011 AND      BYTE PTR [EAX], DH
00000013 XOR      AH, BYTE PTR [EAX]
00000015 XOR      BYTE PTR [EBX + 0x20], AL
00000018 INC      ESI
00000019 INC      EDX
0000001A AND      BYTE PTR [EAX], BH
0000001C INC      ESP
0000001D AND      BYTE PTR [ESI + 0x38], AL
00000020 AND      BYTE PTR [EAX], DH
00000022 XOR      BYTE PTR [EAX], AH
00000024 XOR      BYTE PTR [EDX], DH
00000026 AND      BYTE PTR [EAX], DH
00000028 XOR      BYTE PTR [EAX], AH
0000002A XOR      BYTE PTR [EAX], DH
0000002C AND      BYTE PTR [EAX], DH
```

EMULATION

Again, we are using a x86 in the MASM engine.

- Here, we are using llvm, but there are other jitter engines.
- About the address, we are randomly using this one. 😊
- The initial eax and ecx values.

```
from miasm2.analysis.binary import Container
from miasm2.analysis.machine import Machine
from miasm2.jitter.csts import PAGE_READ, PAGE_WRITE

with open("output.bin") as fdesc:
    cont=Container.from_stream(fdesc)

machine=Machine('x86_32')
mdis=machine.dis_engine(cont.bin_stream)
ourblocks = mdis.dis_multiblock(0)
for block in ourblocks:
    print block

    jitter = machine.jitter("llvm")
    jitter.init_stack()
    s = open("output.bin").read()
    run_addr = 0x40000000
    #jitter.cpu.EAX=run_addr
    jitter.cpu.EAX=1
    jitter.cpu.ECX=6
    jitter.vm.add_memory_page(run_addr, PAGE_READ | PAGE_WRITE, s)

    def code_sentinelle(jitter):
        jitter.run = False
        jitter.pc = 0
        return True

    jitter.add_breakpoint(0x40000002c, code_sentinelle)
    jitter.push_uint32_t(0x40000002c)
    jitter.jit.log_regs = True
    jitter.jit.log_mn = True
    jitter.init_run(run_addr)
    jitter.continue_run()
```

Remember: during the Keystone phase, we have saved our binary using output.bin filename.

MIASM disassembling the code.

From slide 21, the last instruction is at 2c, so: 0x40000000 + 0x2c 😊

EMULATION

```
root@kali:/analysis# python miasm.py
WARNING: not enough bytes in str
WARNING: cannot disasm at 2D
WARNING: not enough bytes in str
WARNING: cannot disasm at 2D
loc_00000000:0x00000000
PUSH    EBX
MOV     EBX, 0xC3
SUB     EAX, EBX
POP     EBX
SUB     EAX, 0xA3
SUB     EAX, 0x38
ADD     EAX, ECX
ADD     EAX, 0x38
ADD     EAX, 0xA3
PUSH    EDX
PUSH    ECX
MOV     ECX, 0x62
MOV     EDX, ECX
POP     ECX
INC     EDX
ADD     EDX, 0x61
DEC     EDX
ADD     EAX, EDX
POP     EDX
```

MIASM
perfectly
disassembled
the binary.

It is not a problem,
of course. 😊

EMULATION

```
40000000 PUSH     EBX
EAX 00000001 EBX 00000000 ECX 00000006 EDX 00000000 ESI 00000000 EDI 00000000
ESP 0123FFE8 EBP 00000000 EIP 40000000 zf 0 nf 0 of 0 cf 0
40000001 MOV      EBX, 0xC3
EAX 00000001 EBX 000000C3 ECX 00000006 EDX 00000000 ESI 00000000 EDI 00000000
ESP 0123FFE8 EBP 00000000 EIP 40000000 zf 0 nf 0 of 0 cf 0
40000006 SUB      EAX, EBX
EAX FFFFFFF3E EBX 000000C3 ECX 00000006 EDX 00000000 ESI 00000000 EDI 00000000
ESP 0123FFF8 EBP 00000000 EIP 40000000 zf 0 nf 1 of 0 cf 1
40000008 POP      EBX
EAX FFFFFFF3E EBX 00000000 ECX 00000006 EDX 00000000 ESI 00000000 EDI 00000000
ESP 0123FFEC EBP 00000000 EIP 40000000 zf 0 nf 1 of 0 cf 1
40000009 SUB      EAX, 0xA3
EAX FFFFE9B EBX 00000000 ECX 00000006 EDX 00000000 ESI 00000000 EDI 00000000
ESP 0123FFEC EBP 00000000 EIP 40000000 zf 0 nf 1 of 0 cf 0
4000000E SUB      EAX, 0x38
EAX FFFFE63 EBX 00000000 ECX 00000006 EDX 00000000 ESI 00000000 EDI 00000000
ESP 0123FFEC EBP 00000000 EIP 40000000 zf 0 nf 1 of 0 cf 0
40000011 ADD      EAX, ECX
EAX FFFFE69 EBX 00000000 ECX 00000006 EDX 00000000 ESI 00000000 EDI 00000000
ESP 0123FFEC EBP 00000000 EIP 40000000 zf 0 nf 1 of 0 cf 0
40000013 ADD      EAX, 0x38
EAX FFFFEA1 EBX 00000000 ECX 00000006 EDX 00000000 ESI 00000000 EDI 00000000
ESP 0123FFEC EBP 00000000 EIP 40000000 zf 0 nf 1 of 0 cf 0
40000016 ADD      EAX, 0xA3
EAX FFFFFFF44 EBX 00000000 ECX 00000006 EDX 00000000 ESI 00000000 EDI 00000000
ESP 0123FFEC EBP 00000000 EIP 40000000 zf 0 nf 1 of 0 cf 0
4000001B PUSH     EDX
EAX FFFFFFF44 EBX 00000000 ECX 00000006 EDX 00000000 ESI 00000000 EDI 00000000
ESP 0123FFF8 EBP 00000000 EIP 40000000 zf 0 nf 1 of 0 cf 0
4000001C PUSH     ECX
EAX FFFFFFF44 EBX 00000000 ECX 00000006 EDX 00000000 ESI 00000000 EDI 00000000
ESP 0123FFF4 EBP 00000000 EIP 40000000 zf 0 nf 1 of 0 cf 0
```

EMULATION

```
4000001D MOV ECX, 0x62
EAX FFFFFFF4 EBX 00000000 ECX 00000062 EDX 00000000 ESI 00000000 EDI 00000000
ESP 0123FFF4 EBP 00000000 EIP 40000000 zf 0 nf 1 of 0 cf 0
40000022 MOV EDX, ECX
EAX FFFFFFF4 EBX 00000000 ECX 00000062 EDX 00000062 ESI 00000000 EDI 00000000
ESP 0123FFF4 EBP 00000000 EIP 40000000 zf 0 nf 1 of 0 cf 0
40000024 POP ECX
EAX FFFFFFF4 EBX 00000000 ECX 00000006 EDX 00000062 ESI 00000000 EDI 00000000
ESP 0123FFF8 EBP 00000000 EIP 40000000 zf 0 nf 1 of 0 cf 0
40000025 INC EDX
EAX FFFFFFF4 EBX 00000000 ECX 00000006 EDX 00000063 ESI 00000000 EDI 00000000
ESP 0123FFF8 EBP 00000000 EIP 40000000 zf 0 nf 0 of 0 cf 0
40000026 ADD EDX, 0x61
EAX FFFFFFF4 EBX 00000000 ECX 00000006 EDX 000000C4 ESI 00000000 EDI 00000000
ESP 0123FFF8 EBP 00000000 EIP 40000000 zf 0 nf 0 of 0 cf 0
40000029 DEC EDX
EAX FFFFFFF4 EBX 00000000 ECX 00000006 EDX 000000C3 ESI 00000000 EDI 00000000
ESP 0123FFF8 EBP 00000000 EIP 40000000 zf 0 nf 0 of 0 cf 0
4000002A ADD EAX, EDX
EAX 00000007 EBX 00000000 ECX 00000006 EDX 000000C3 ESI 00000000 EDI 00000000
ESP 0123FFF8 EBP 00000000 EIP 40000000 zf 0 nf 0 of 0 cf 1
```

The expected result according to initial values. ☺

EMULATION

- ✓ Let's solve the obfuscation problem again, but this time we will try a **symbolic solution** using **MIASM**. 😊

```
root@kali:~/programs/miasm# python
```

```
Python 2.7.15rc1 (default, Apr 15 2018, 21:51:34)
```

```
[GCC 7.3.0] on linux2
```

```
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> from miasm2.analysis.binary import Container
```

```
>>> from miasm2.analysis.machine import Machine
```

```
>>> from miasm2.jitter.csts import PAGE_READ, PAGE_WRITE
```

EMULATION

```
>>> with open("output.bin") as fdesc:
...     cont=Container.from_stream(fdesc)
...
>>> bhackmach=Machine('x86_32')
>>> aborgesdis=bjackmach.dis_engine(cont.bin_stream)
>>> ourblocks = aborgesdis.dis_multiblock(0)
```

WARNING: not enough bytes in str

WARNING: cannot disasm at 2D

WARNING: not enough bytes in str

WARNING: cannot disasm at 2D

EMULATION

```
>>> sym = bhackmach.ira( )  
>>> for block in ourblocks:  
...     sym.add_block(block)  
...
```

```
[<miasm2.ir.ir.IRBlock object at 0x7ff0974a0d70>]
```

```
[]
```

```
>>> from miasm2.ir.symbexec import SymbolicExecutionEngine  
>>> sb = SymbolicExecutionEngine(sym,bhackmach.mn.regs.regs_init)  
>>> symbolic_pc = sb.run_at(0, step=True)
```

EMULATION

Instr **PUSH** **EBX**

Assignblk:

ESP = ESP + -0x4

@32[ESP + -0x4] = EBX

ESP = ESP_init + 0xFFFFFFFF

@32[ESP_init + 0xFFFFFFFF] = EBX_init

Instr **MOV** **EBX, 0xC3**

Assignblk:

EBX = 0xC3

ESP = ESP_init + 0xFFFFFFFF

EBX = 0xC3

@32[ESP_init + 0xFFFFFFFF] = EBX_init

Instr **SUB** **EAX, EBX**

Assignblk:

zf = (EAX + -EBX) ? (0x0, 0x1)

nf = (EAX + -EBX)[31:32]

pf = parity((EAX + -EBX) & 0xFF)

of = ((EAX ^ (EAX + -EBX)) & (EAX ^ EBX))[31:32]

cf = (((EAX ^ EBX) ^ (EAX + -EBX)) ^ ((EAX ^ (EAX + -EBX)) & (EAX ^ EBX)))[31:32]

af = ((EAX ^ EBX) ^ (EAX + -EBX))[4:5]

EAX = EAX + -EBX

EMULATION

```
EAX          = EAX_init + 0xFFFFFFFF3D
cf           = (EAX_init ^ ((EAX_init ^ (EAX_init + 0xFFFFFFFF3D)) & (EAX_init ^
0xC3))) ^ (EAX_init + 0xFFFFFFFF3D) ^ 0xC3)[31:32]
pf           = parity((EAX_init + 0xFFFFFFFF3D) & 0xFF)
zf           = (EAX_init + 0xFFFFFFFF3D) ? (0x0, 0x1)
af           = (EAX_init ^ (EAX_init + 0xFFFFFFFF3D) ^ 0xC3)[4:5]
ESP          = ESP_init + 0xFFFFFFF0C
of           = ((EAX_init ^ (EAX_init + 0xFFFFFFFF3D)) & (EAX_init ^ 0xC3))[31:32]
]
EBX          = 0xC3
nf           = (EAX_init + 0xFFFFFFFF3D)[31:32]
@32[ESP_init + 0xFFFFFFF0C] = EBX_init
```

Instr POP EBX

Assignblk:

```
EBX = @32[ESP]
ESP = ESP + 0x4
```

```
EAX          = EAX_init + 0xFFFFFFFF3D
cf           = (EAX_init ^ ((EAX_init ^ (EAX_init + 0xFFFFFFFF3D)) & (EAX_init ^
0xC3))) ^ (EAX_init + 0xFFFFFFFF3D) ^ 0xC3)[31:32]
pf           = parity((EAX_init + 0xFFFFFFFF3D) & 0xFF)
zf           = (EAX_init + 0xFFFFFFFF3D) ? (0x0, 0x1)
af           = (EAX_init ^ (EAX_init + 0xFFFFFFFF3D) ^ 0xC3)[4:5]
of           = ((EAX_init ^ (EAX_init + 0xFFFFFFFF3D)) & (EAX_init ^ 0xC3))[31:32]
]
nf           = (EAX_init + 0xFFFFFFFF3D)[31:32]
@32[ESP_init + 0xFFFFFFF0C] = EBX_init
```

EMULATION

I've skipped many lines to show the last ones....

EMULATION

```
Instr POP          EDX
Assignblk:
IRDst = loc_000000000000002D:0x0000002d
```

Fantastic! The same output as other solutions.
(eax = eax + ecx)

```
EAX                = EAX_init + ECX_init
cf                 = (((EAX_init + ECX_init) ^ (EAX_init + ECX_init + 0xFFFFFFFF3D)) &
  ((EAX_init + ECX_init + 0xFFFFFFFF3D) ^ 0xFFFFFFFF3C)) ^ (EAX_init + ECX_init) ^ (EAX_in
it + ECX_init + 0xFFFFFFFF3D) ^ 0xC3)[31:32]
pf                 = parity((EAX_init + ECX_init) & 0xFF)
zf                 = (EAX_init + ECX_init) ? (0x0, 0x1)
af                 = ((EAX_init + ECX_init) ^ (EAX_init + ECX_init + 0xFFFFFFFF3D) ^ 0x
C3)[4:5]
IRDst              = 0x2D
of                 = (((EAX_init + ECX_init) ^ (EAX_init + ECX_init + 0xFFFFFFFF3D)) &
  ((EAX_init + ECX_init + 0xFFFFFFFF3D) ^ 0xFFFFFFFF3C))[31:32]
nf                 = (EAX_init + ECX_init)[31:32]
@32[ESP_init + 0xFFFFFFFF8] = ECX_init
@32[ESP_init + 0xFFFFFFFFC] = EDX_init
```

ANTI-REVERSING TECHNIQUES

ANTI-REVERSING

- ✓ **IDA Pro** faces few problems with obfuscation, though exist interesting solutions for helping us. Additionally, we always can write **IDA processor modules** to make the analysis less painful and repeatable.
- ✓ Unfortunately, protectors such as **VMProtect** and **Themida** usually deploy heavy layers of obfuscation techniques. For example, **VMProtect** holds the following characteristics:
 - ✓ Most of the time, code protected with **VMProtect** is seen in **64-bit malwares**.
 - ✓ Any function from the original malware is removed of the **IAT**. This means that **IAT** shown by **pestudio** and **PE Bear** tools is associated to the packer itself.
 - ✓ **VMProtect** checks the file memory integrity. Therefore, any attempt to change the malware on memory is easily detected.
 - ✓ Instructions (**CPU code**) are virtualized and transformed into virtual machine instructions (**RISC instruction**).

ANTI-REVERSING

- ✓ The obfuscation is **stack based**.
- ✓ The **virtualized code is polymorphic**, so there are many representations referring the same CPU instruction.
- ✓ The **original code is never entirely decrypted on the memory**.
- ✓ There are many **dead and useless codes**. Additionally, there is some **code reordering using unconditional jumps**.
- ✓ There are **many hooks** on calls such as **LoadString()** and **LdrAccessResource() functions** (resources are usually encrypted).
- ✓ It has **few anti-debugger and anti-vm tricks**.
- ✓ **Calls to IAT functions are replaced by calls at VMProtect section (VMProtect's IAT)**.
- ✓ There are also **fake push instructions**.

ANTI-REVERSING

pestudio 8.73 - Malware Initial Assessment - www.winitor.com

File Help

indicators (wait..)

virustotal (13/68 - 30.04.2018)

dos-stub (192 bytes)

file-header (Apr.2018)

optional-header (Console)

directories (2)

sections (entry-point)

libraries (kernel32)

imports (TlsSetValue)

exports (0)

tls-callbacks (n/a)

resources (12)

strings (wait..)

debug (n/a)

manifest (administrator)

version (n/a)

certificate (n/a)

overlay (wait..)

Non-disclosure Agreement

Packed with Themida

name (1)	group (1)
TlsSetValue	2

There is only one imported function

Non-disclosure Agreement

cpu: 32-bit file-type: executable

ALEXANDRE BORGES – IT IS NOT ALLOWED TO COPY OR REPRODUCE THIS SLIDE.

HTTP://WWW.BLACKSTORMSECURITY.COM

42

ANTI-REVERSING

pestudio 8.73 - Malware Initial Assessment - www.winitor.com

File Help

c:\extrato

indicators (wait..)

virustotal (n/a)

dos-stub (192 bytes)

file-header (Apr.2018)

optional-header (Console)

directories (2)

sections (entry-point)

libraries (2/9)

imports (9/6/0/1/5)

exports (0)

tls-callbacks (n/a)

resources (12)

strings (wait..)

debug (n/a)

manifest (administrator)

version (n/a)

certificate (n/a)

overlay (wait..)

Non-disclosure Agreement

After Unpacking – Without Themida

name (9)	group (6)	ar
ExitWindowsEx	17	
PathFileExistsW	6	
URLDownloadToFileW	6	
InternetOpenW	3	
ShellExecuteW	2	
RegSetValueExW	1	
GetUserDefaultLCID	-	
SysAllocString	-	
CoUninitialize	-	

The Import list is longer when viewed in IDA Pro ☺

cpu: 32-bit file-type:

Non-disclosure Agreement

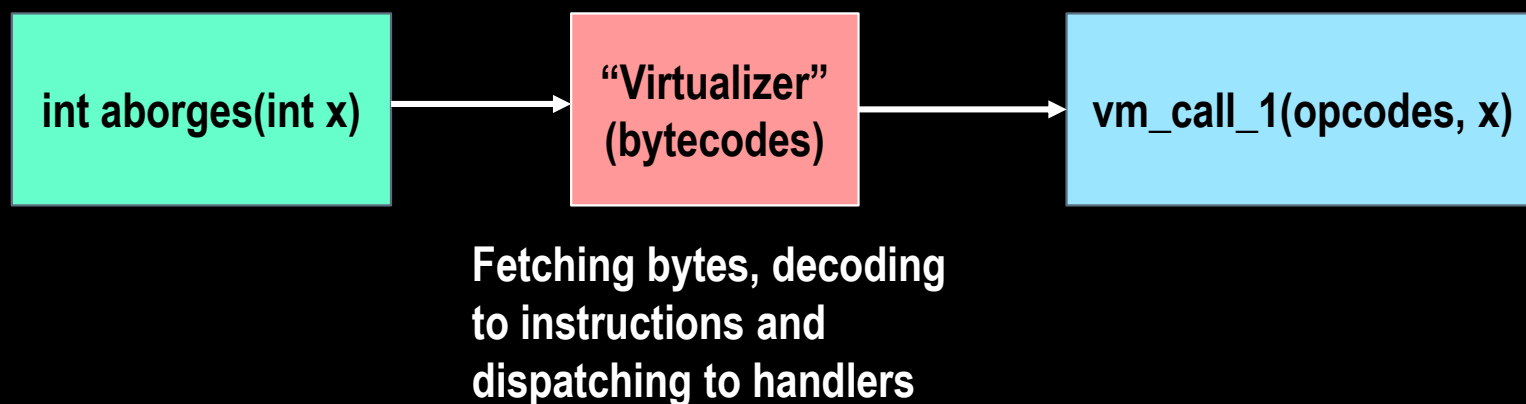
ALEXANDRE BORGES – IT IS NOT ALLOWED TO COPY OR REPRODUCE THIS SLIDE.

HTTP://WWW.BLACKSTORMSECURITY.COM

43

ANTI-REVERSING

- Roughly, x86 / x64 instructions are submitted to the following “transformation”:



- ✓ **Handlers**, which are independent of one each other, usually **configure registers, a encryption key and memory**.
- ✓ Additionally, there is **one handler by instruction type**.
- ✓ **The challenge is to revert “virtualized instructions” back to original ones.**

ANTI-REVERSING

- ✓ Furthermore, there are many “obstacles” to circumvent:
 - ✓ **Instructions’ operands are encrypted** using keys (initializing code) provided by handlers. Furthermore, **handlers are “started” by the VM dispatcher.**
 - ✓ **Code obfuscation: VM instructions are obfuscated**
 - ✓ **Dead (garbage) code:** this technique is implemented by inserting codes whose results will be overwritten in next lines of code or, worse, they won’t be used anymore.
 - ✓ **Code-flow graph flattening:**
 - ✓ This technique is used to **hide the real control flow of a program.**
 - ✓ In a general way, the idea is to **break the control-flow by removing if-statements and loops, transforming the flow in a series of switch-case statements.**

ANTI-REVERSING

- ✓ Thus, there is a **dispatcher** handing over the control flow to handlers, which **each handler updates the instruction pointer to the value of the next handler to be executed (virtualize the flow control).**
- ✓ Usually there is an **invocation stub**, which makes the **transition to from native instructions to the virtualized instruction.**
- ✓ This approach presents two reversing problems: the **mapping can be from CISC to RISC instruction and the original registers can be turned into special registers from VM.**
- ✓ Because trade-offs, **CFG is only applied to specific functions.**

ANTI-REVERSING

```
#include <stdio.h>
```

```
int main (void)
```

```
{
```

```
    int aborges = 0;
```

```
    while (aborges < 30)
```

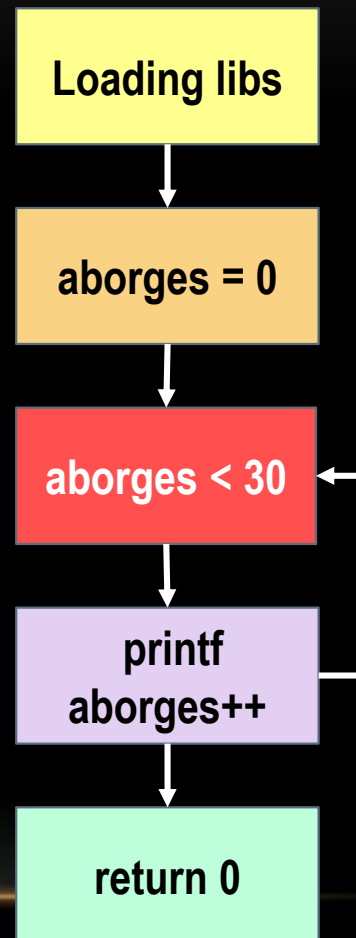
```
    {
```

```
        printf ("%d\n", aborges);  
        aborges++;
```

```
    }
```

```
    return 0;
```

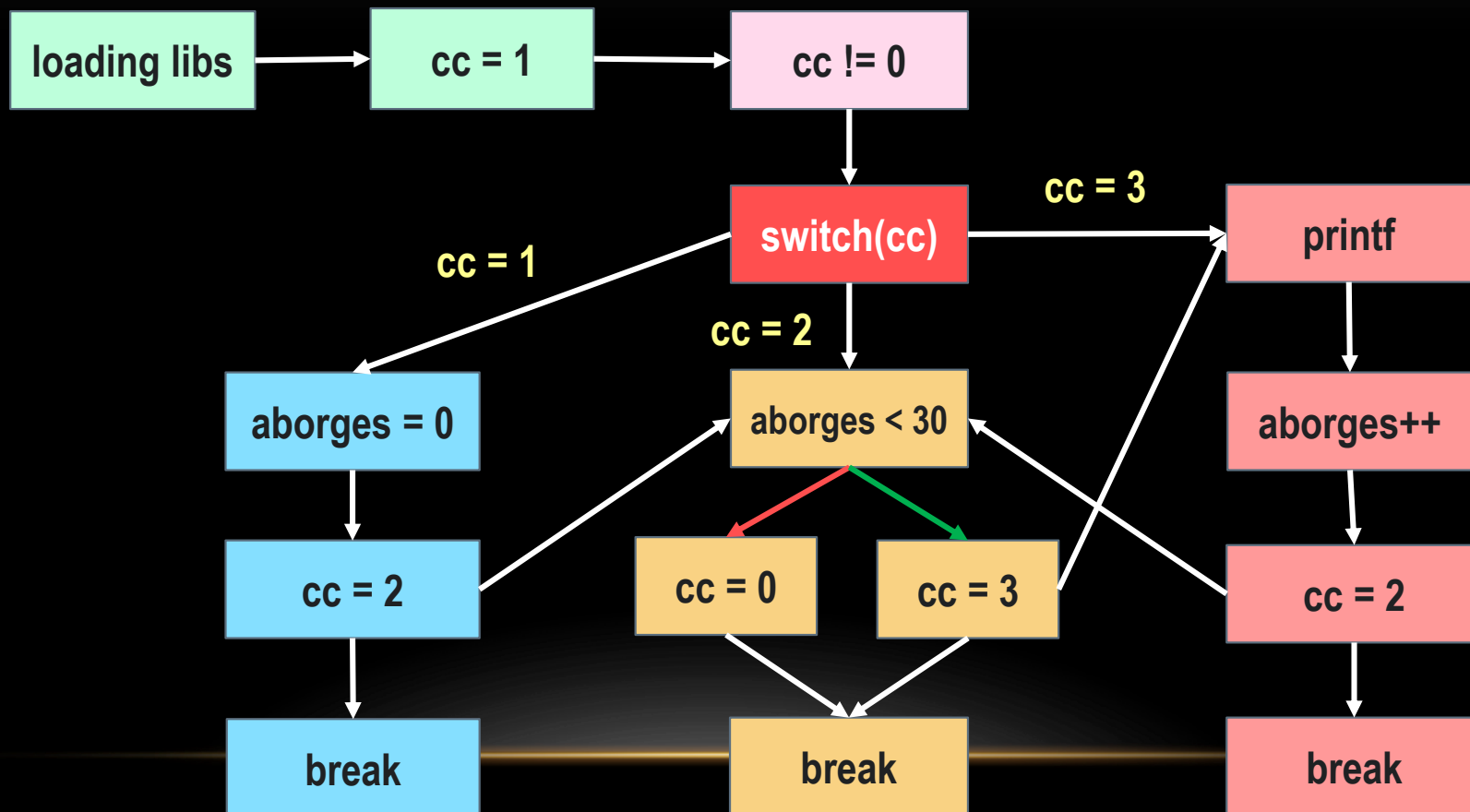
```
}
```



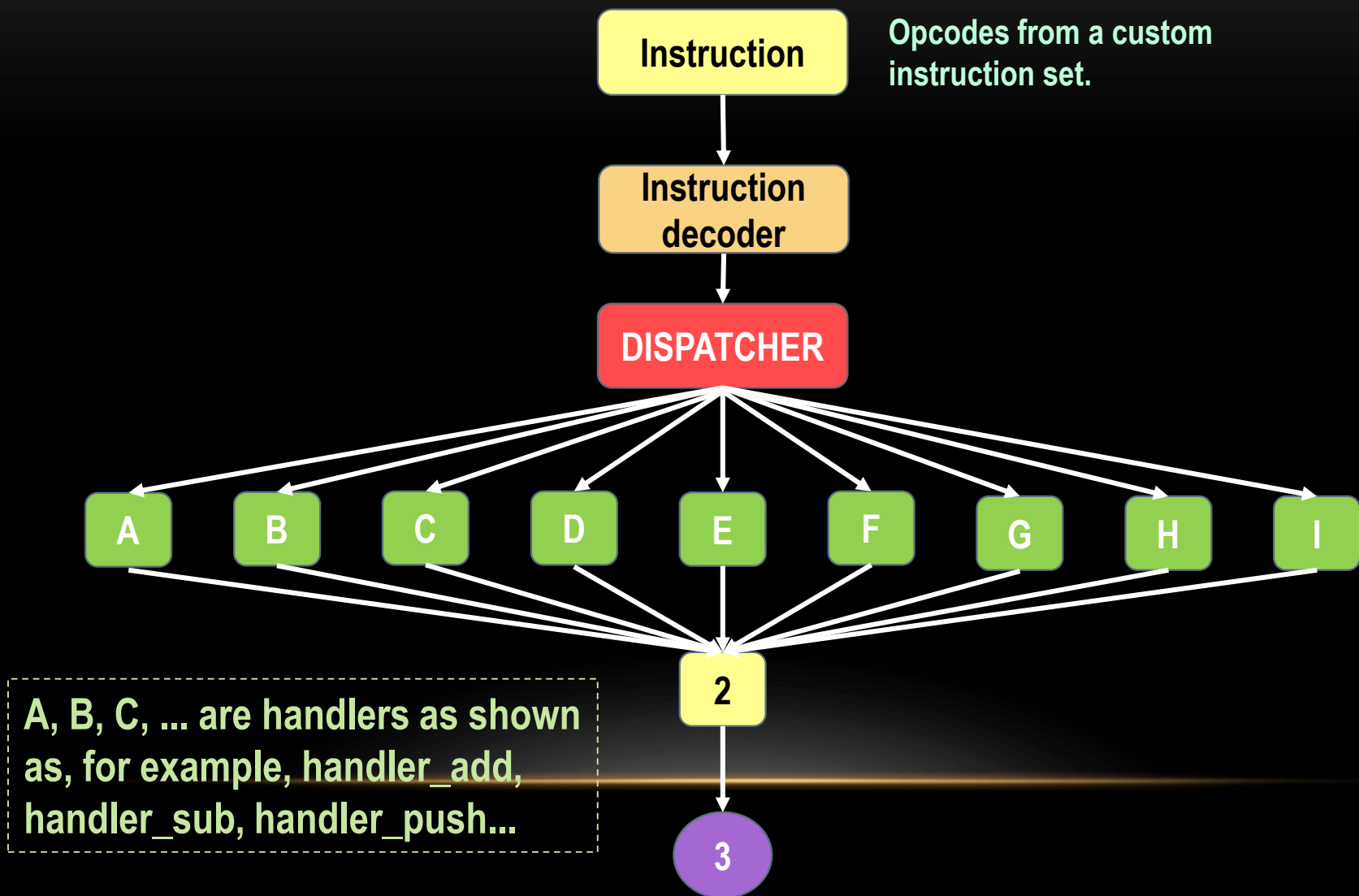
ANTI-REVERSING

❖ Disadvantages:

- ✓ Loss of performance
- ✓ Easy to identify the CFG flattening



ANTI-REVERSING



ANTI-REVERSING

- ✓ Remember: obfuscate is transforming a code from X to X' using any trick, including virtualization.
- ✓ Identify if the program is virtualized or not.
- ✓ Identify the input points.
- ✓ Understand the virtual program counter (known as vpc).
- ✓ Try to understand each component of the proposed VM.
- ✓ Different approaches such as Dynamic Symbolic Execution (DSE) and Dynamic Taint Analysis could help us.

ANTI-REVERSING

- ✓ **Constant unfolding:** technique used by obfuscators to replace a constant by a bunch of code that produces the same resulting constant's value.
- ✓ **Pattern-based obfuscation:** exchange of one instruction by a set of equivalent instructions.
- ✓ **Abusing inline functions.**
- ✓ **Code duplication:** different paths coming into the same destination (used by virtualization obfuscators).
- ✓ **Control indirection 1:** call instruction → stack pointer update → return skipping some junk code after the call instruction (**RET x**).
- ✓ **Control indirection 2:** malware trigger an **exception** → **registered exception is called** → **new branch of instructions**

ANTI-REVERSING

- ✓ **Opaque predicate:** Although apparently there is an evaluation, the result is always evaluated to true (or false), which means an **unconditional jump**. Thus, **there is a dead branch**. Usually, a **series of arithmetic / logic tricks** are used.
- ✓ **Polymorphism:** it is produced by **self-modification code** (like shellcodes) and by **encrypting resources** (similar most malware samples).
- ✓ **Anti-debugging:** used as irritating techniques to slow the process analysis.
- ✓ **Call stack manipulation:** Changes the stack flow **by using instruction tricks composed with the ret instruction**, making the **real ret hidden**.
- ✓ **Anti-VM techniques:** prevents the malware sample to run inside a VM.

ANTI-REVERSING

Opaque predicate and anti-disassembly techniques

.text:00401000 loc_401000: **; CODE XREF: _main+Fp**

.text:00401000	push	ebp
.text:00401001	mov	ebp, esp
.text:00401003	xor	eax, eax
.text:00401005	jz	short near ptr loc_40100D+1
.text:00401007	jnz	near ptr loc_40100D+4

.text:0040100D		
.text:0040100D loc_40100D:		; CODE XREF: .text:00401005j
.text:0040100D		; .text:00401007j

.text:0040100D	jmp	near ptr 0D0A8837h
-----------------------	------------	---------------------------

Decrypted
shellcode

```

seg000:0000020C loc_20C:                                ; CODE XREF: sub_208+14↓j
seg000:0000020C      lodsb
seg000:0000020D      mov     dl, al
seg000:0000020F      sub     dl, 41h ; 'A'
seg000:00000212      shl     dl, 4
seg000:00000215      lodsb
seg000:00000216      sub     al, 41h ; 'A'
seg000:00000218      add     al, dl
seg000:0000021A      stosb
seg000:0000021B      dec     ecx
seg000:0000021C      jnz     short loc_20C
seg000:0000021E      retn
seg000:0000021E      sub_208      endp
seg000:0000021E
seg000:0000021F ; -----
seg000:0000021F
seg000:0000021F loc_21F:                                ; CODE XREF: seg000:00000206↑j
seg000:0000021F      call    sub_208
seg000:00000224      mov     ebp, esp
seg000:00000226      sub     esp, 40h
seg000:0000022C      jmp     loc_364
seg000:00000231 ; ===== S U B R O U T I N E =====
seg000:00000231
seg000:00000231 sub_231      proc near                    ; CODE XREF: sub_252+1F↓j
seg000:00000231 arg_0      = dword ptr 4
seg000:00000231
seg000:00000231      push    esi
seg000:00000232      push    edi
seg000:00000233      mov     esi, [esp+8+arg_0]
seg000:00000237      xor     edi, edi
seg000:00000239      cld
seg000:0000023A
seg000:0000023A loc_23A:                                ; CODE XREF: sub_231+15↓j
seg000:0000023A      xor     eax, eax
seg000:0000023C      lodsb
seg000:0000023D      cmp     al, ah
seg000:0000023F      jz      short loc_248
seg000:00000241      ror     edi, 0Dh
seg000:00000244      add     edi, eax
seg000:00000246      jmp     loc_23A
seg000:0000024B ; -----
seg000:0000024B
seg000:0000024B loc_24B:                                ; CODE XREF: sub_231+E↑j
seg000:0000024B      mov     eax, edi
seg000:0000024D      pop     edi
seg000:0000024E      pop     esi

```

ANTI-REVERSING

00401040	call + \$5
00401045	pop ecx
00401046	inc ecx
00401047	inc ecx
00401048	add ecx, 4
00401049	add ecx, 4
0040104A	push ecx
0040104B	ret
0040104C	sub ecx, 6
0040104D	dec ecx
0040104E	dec ecx
0040104F	jmp 0x401320

❖ Call stack manipulation:

✓ Do you know what's happening here? 😊

ANTI-REVERSING

- ✓ Is it possible to **deobfuscate virtualized instructions**? Yes, it is possible using **reverse recursive substitution** (similar -- not equal -- to **backtracking feature from Metasm**).
- ✓ Additionally, **symbolic equation system** is another good approach (again....., **Metasm and Miasm**!).
- ✓ There are many good plugins such as **Code Unvirtualizer, VMAttack, VMSweeper**, and so on, which could be used to handle simple virtualization problems.
- ✓ Some evolution of the instruction virtualizers has risen using simple and efficient concepts of cryptography as **Substitution Boxes (S-Boxes)**.

ANTI-DEBUGGING TECHNIQUE

ANTI-VM TECHNIQUES

- ✓ There is a powerful **anti-debugging technique** named “**int 2d**”, which **trigger an exception**.
- ✓ However, the trick is: when a debugger is attached, the **exception triggered** by the **int 2d** instruction **is handled by debugger**.
- ✓ However **when a debugger is not attached**, is the malware that handle the **exception** (for example, using the combination try / catch from C++) 😊.
- ✓ Therefore, the **malware only will trigger + catch the exception whether there is not an attached debugger**.
- ✓ Additionally, there is **little side effect** here: when the exception happen, the exception is triggered, but the **next 1 byte following the int 2D is skipped**. **This side effect can be used to deceive the analyst**.

ANTI-VM TECHNIQUES

```
static BOOL DebuggerPresent = TRUE;

static LONG CALLBACK VectoredHandler(_In_ PEXCEPTION_POINTERS ExceptionInfo)
{
    DebuggerPresent = FALSE;
    if (ExceptionInfo->ExceptionRecord->ExceptionCode == EXCEPTION_BREAKPOINT)
    {
        return EXCEPTION_CONTINUE_EXECUTION;
    }
    return EXCEPTION_CONTINUE_SEARCH;
}

BOOL Debugger_int2d()
{
    /*
    PVOID WINAPI AddVectoredExceptionHandler(
        _In_ ULONG FirstHandler,
        _In_ PVECTORED_EXCEPTION_HANDLER VectoredHandler);
    */

    PVOID hndl = AddVectoredExceptionHandler(1, VectoredHandler);
    DebuggerPresent = TRUE; /* At this point is our indication that the exception was consumed! */
    abint2d(); /* Calling the INT 2D instruction from abint2d*/
    RemoveVectoredExceptionHandler(hndl); /* Remove the Vectored Exception Handler*/
    return DebuggerPresent;
}
```

```
1 | .code
2 | abint2d proc
3 |     int 2dh
4 |     nop
5 |     ret
6 | abint2d endp
7 |
8 | end
```

ALEXANDRE BORGES – IT IS NOT ALLOWED TO COPY OR REPRODUCE THIS SLIDE.

ANTI-VM TECHNIQUES

```
LONG CALLBACK VectoredHandler(  
    _In_ PEXCEPTION_POINTERS ExceptionInfo  
);
```

```
typedef struct _EXCEPTION_POINTERS {  
    PEXCEPTION_RECORD ExceptionRecord;  
    PCONTEXT ContextRecord;  
} EXCEPTION_POINTERS, *PEXCEPTION_POINTERS;
```

```
typedef struct _EXCEPTION_RECORD {  
    DWORD ExceptionCode;  
    DWORD ExceptionFlags;  
    struct _EXCEPTION_RECORD *ExceptionRecord;  
    PVOID ExceptionAddress;  
    DWORD NumberParameters;  
    ULONG_PTR ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS];  
} EXCEPTION_RECORD, *PEXCEPTION_RECORD;
```

✓ **Exception_Pointer** structure contains an **exception record** with a machine-independent **description of an exception** and a **context record** with a machine-dependent description of the **processor context** at the time of the exception.

ANTI-VM TECHNIQUES

```
C:\Users\Administrador\source\repos\Debug_INT_2D\x64\Release>Debug_INT_2D.exe
```

```
-----  
*****Testing the INT2D anti-debugging technique....*****  
-----
```

```
There is NOT a debugger attached. :)
```

```
It is really nice to be speaking in BSIDES SAO PAULO!  
Thank you for attending my talk!
```

```
-----  
***** End of the INT2D anti-debugging technique test....*****  
-----
```

```
-----  
*****Testing the INT2D anti-debugging technique....*****  
-----
```

```
There is a DEBUGGER attached. :(<
```

```
It is really nice to be speaking in BSIDES SAO PAULO!  
Thank you for attending my talk!
```

ANTI-VM TECHNIQUES

ANTI-VM TECHNIQUES

- ✓ **Anti-VM techniques** detecting VMWare (**checking I/O port communication**), VirtualBox, Parallels, SeaBIOS emulator, QEMU emulator, Bochs emulator, QEMU emulator, Hyper-V, Innotek VirtualBox, sandboxes (Cuckoo).
- ✓ There are dozens of techniques that could be used for detection Vmware sandboxes:
 - ✓ Examining the registry (**OpenSubKey()** function) to try to find entries related to tools installed in the guest (**HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VirtualMachine\Guest\Parameters**).
 - ✓ Using **WMI** to query the **Win32_BIOS management class** to interact with attributes from the physical machine.

ANTI-VM TECHNIQUES

```
using System;
using System.Management;

namespace Test_VM
{
    class Program
    {
        static void Main(string[] args)
        {
            ManagementClass bioscClass =
                new ManagementClass("Win32_BIOS");
            ManagementObjectCollection biosc =
                bioscClass.GetInstances();
            ManagementObjectCollection.ManagementObjectEnumerator
                bioscEnumerator =
                biosc.GetEnumerator();
            while (bioscEnumerator.MoveNext())
            {
                ManagementObject biosc1 =
                    (ManagementObject)bioscEnumerator.Current;
                Console.WriteLine(
                    "Attributes:\n\n" + "Version:\t " + biosc1["version"].ToString( ));
                Console.WriteLine(
                    "SerialNumber:\t " + biosc1["SerialNumber"].ToString());
                Console.WriteLine(
                    "OperatingSystem:\t " + biosc1["TargetOperatingSystem"].ToString());
                Console.WriteLine(
                    "Manufacturer:\t " + biosc1["Manufacturer"].ToString());
            }
            //return 0;
        }
    }
}
```

ALEXANDRE BORGES – IT IS NOT ALLOWED TO COPY OR REPRODUCE THIS SLIDE.

ANTI-VM TECHNIQUES

- ✓ As you could remember, we have:
 - ✓ The **ManagementClass** class represents a **Common Information Model (CIM)** management class.
 - ✓ **Win32_BIOS WMI class** represents the **attributes of BIOS** and members of this class enable you to access WMI data using a specific WMI class path.
 - ✓ **GetInstances()** acquires a collection of all instances of the class.
 - ✓ **GetEnumerator()** returns the enumerator (**IEnumerator**) to the collection.
 - ✓ **IEnumerator.Current()** returns the same object.
 - ✓ **IEnumerator.MoveNext()** advances the enumerator to the next element of the collection.

ANTI-VM TECHNIQUES

❖ Physical host:

c:\Users\Administrador\source\repos\Test_VM\Test_VM\bin\Debug>

Test_VM.exe

Attributes:

Version: DELL - 6222004

SerialNumber: **D5965S1**

OperatingSystem: 0

Manufacturer: Dell Inc.

❖ Guest virtual machine:

E:\> **Test_VM.exe**

Attributes:

Version: LENOVO - 6040000

SerialNumber: **VMware**-56 4d 8d c3 a7 c7 e5 2b-39 d6 cc 93 bf 90 28 2d

OperatingSystem: 0

Manufacturer: Phoenix Technologies LTD

ANTI-VM TECHNIQUES

```
1 using System;
2 using System.Management;
3
4 namespace TestVM_2
5 {
6     class Program
7     {
8         static void Main(string[] args)
9         {
10             ManagementClass cscolClass =
11             new ManagementClass("Win32_ComputerSystem");
12             ManagementObjectCollection csinstance =
13                 cscolClass.GetInstances();
14             ManagementObjectCollection.ManagementObjectEnumerator
15                 cscolEnumerator =
16                 csinstance.GetEnumerator();
17             while (cscolEnumerator.MoveNext())
18             {
19                 ManagementObject cscolobj =
20                     (ManagementObject)cscolEnumerator.Current;
21                 string buffer = cscolobj["Manufacturer"].ToString().ToLower();
22                 if (buffer.Contains("vmware"))
23                 {
24                     Console.WriteLine("This program is running in a VMware guest" + cscolobj["Model"].ToString());
25                     Console.WriteLine("Manufacturer:\t" + cscolobj["Manufacturer"].ToString());
26                 }
27                 else
28                 {
29                     Console.WriteLine("This program is NOT RUNNING in a VMware guest");
30                 }
31             }
32         }
33     }
34 }
35
```

ALEXANDRE BORGES – IT IS NOT ALLOWED TO COPY OR REPRODUCE THIS SLIDE.

ANTI-VM TECHNIQUES

❖ PhysicalHost:

```
C:> TestVM_2.exe
```

This program is NOT RUNNING in a VMware guest

❖ VirtualMachine:

```
E:\> TestVM_2.exe
```

This program is running in a VMware guestVMware Virtual Platform

Manufacturer: VMware, Inc.

ANTI-VM TECHNIQUES

```
namespace TestVM_3
{
    class Program
    {
        static void Main(string[] args)
        {
            ManagementClass tempClass =
            new ManagementClass("Win32_TemperatureProbe");
            ManagementObjectCollection tempinstance =
                tempClass.GetInstances() ;
            foreach (ManagementObject aborges in tempinstance)
            {
                string buffer = aborges.GetProperty("CurrentReading").ToString( );
                Console.WriteLine("Temperature:\t" + buffer);
            }
        }
    }
}
```

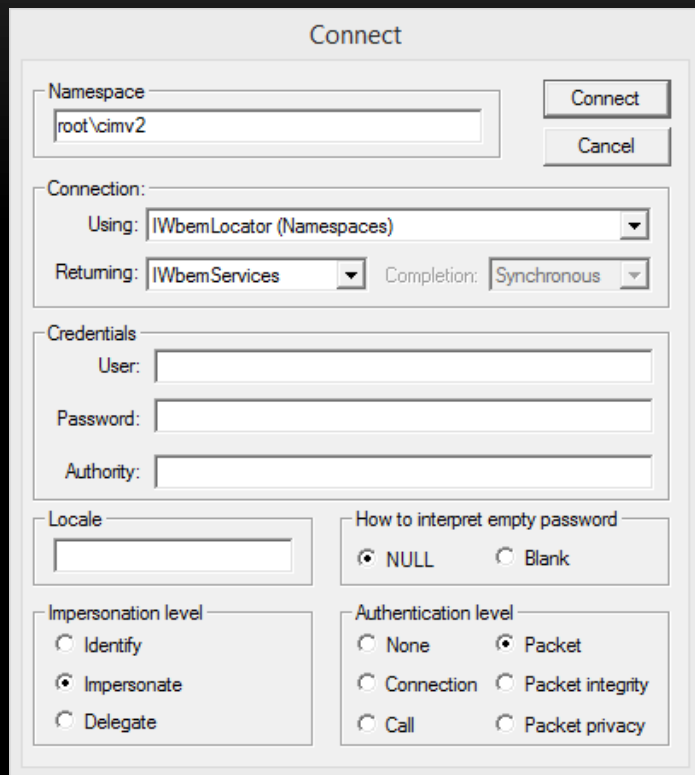
This code does NOT work. ☹️

```
c:\Users\Administrador\source\repos\TestVM_3\TestVM_3\bin\Debug>TestVM_3.exe
```

```
Unhandled Exception: System.NullReferenceException: Object reference not set to an instance of an object.
   at TestVM_3.Program.Main(String[] args) in c:\users\administrador\source\repos\testvm_3\testvm_3\Program.cs:line 16
```

ALEXANDRE BORGES – IT IS NOT ALLOWED TO COPY OR REPRODUCE THIS SLIDE.

ANTI-VM TECHNIQUES



Connect

Namespace:

Connection:

Using:

Returning: Completion:

Credentials:

User:

Password:

Authority:

Locale:

How to interpret empty password:

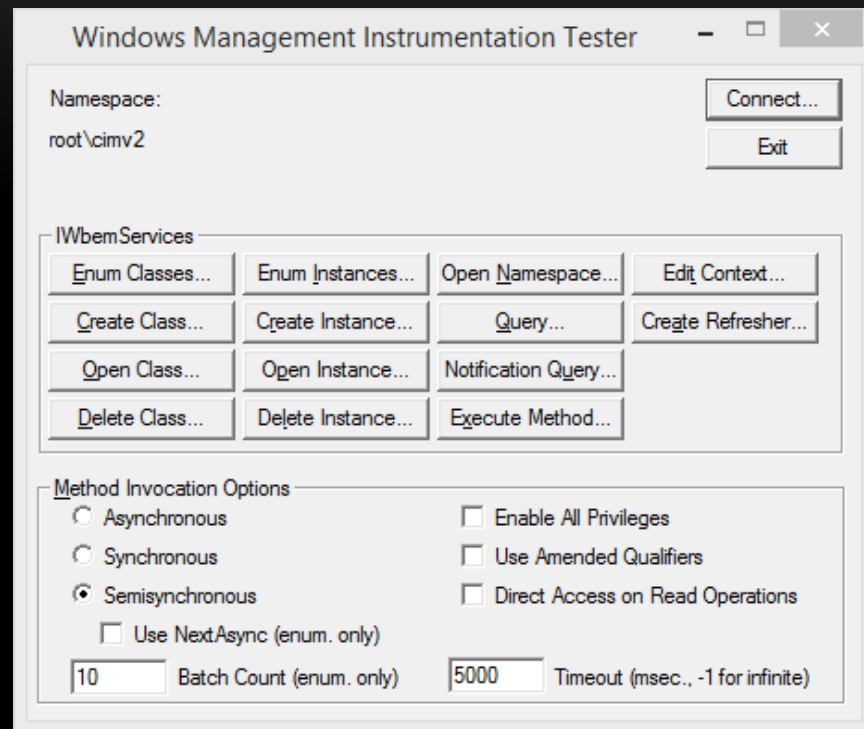
☒ NULL ☐ Blank

Impersonation level:

☐ Identify ☒ Impersonate ☐ Delegate

Authentication level:

☐ None ☒ Packet ☐ Connection ☐ Packet integrity ☐ Call ☐ Packet privacy



Windows Management Instrumentation Tester

Namespace:

IWbemServices

Enum Classes...	Enum Instances...	Open Namespace...	Edit Context...
Create Class...	Create Instance...	Query...	Create Refresher...
Open Class...	Open Instance...	Notification Query...	
Delete Class...	Delete Instance...	Execute Method...	

Method Invocation Options

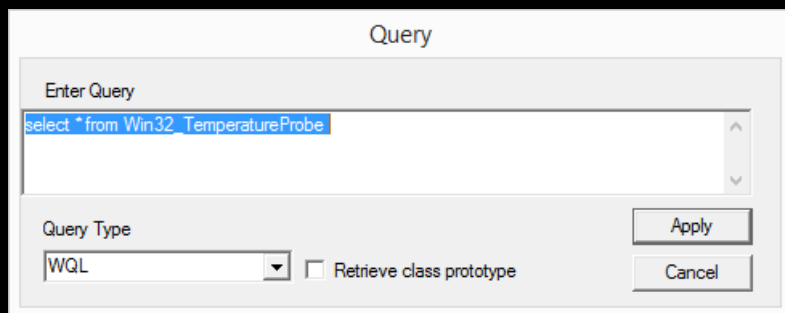
☐ Asynchronous ☐ Enable All Privileges

☐ Synchronous ☐ Use Amended Qualifiers

☒ Semisynchronous ☐ Direct Access on Read Operations

☐ Use NextAsync (enum. only)

Batch Count (enum. only) Timeout (msec., -1 for infinite)

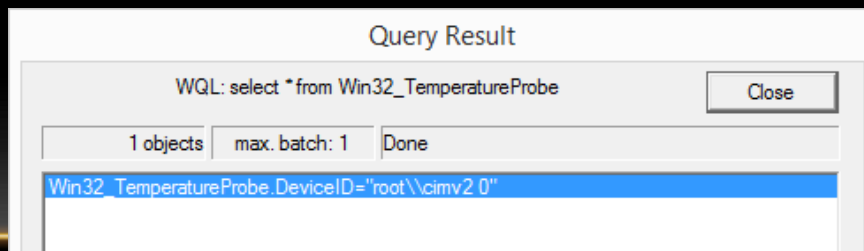


Query

Enter Query

Query Type:

☐ Retrieve class prototype

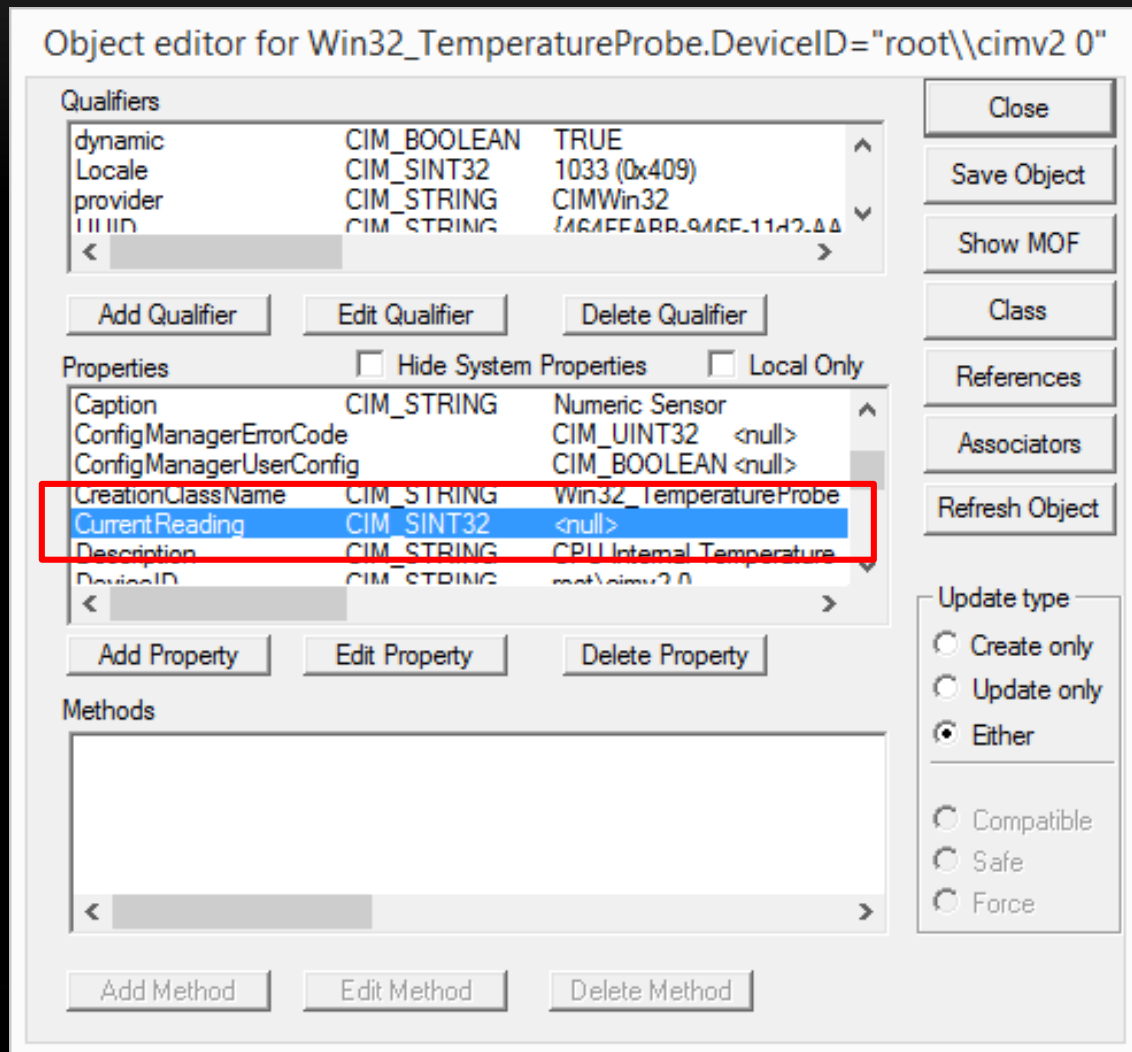


Query Result

WQL: select * from Win32_TemperatureProbe

1 objects max. batch: 1 Done

ANTI-VM TECHNIQUES



Now we know why our prior code didn't work. 😊

ANTI-VM TECHNIQUES

```
using System;
using System.Management;

namespace TestVM_3
{
    public class Program
    {
        public static void Main(string[] args)
        {
            ManagementClass tempClass =
            new ManagementClass("Win32_TemperatureProbe");
            ManagementObjectCollection tempinstance = tempClass.GetInstances();

            foreach (ManagementObject aborges in tempinstance)
            {
                try
                {
                    if (!string.IsNullOrEmpty(aborges.GetPropertyValue("Status").ToString()))
                    {
                        string buffer = aborges.GetPropertyValue("Status").ToString();
                        Console.WriteLine("\nStatus: " + buffer + " Thus, the program is running in a physical host!");
                    }
                }
                catch (NullReferenceException e)
                {
                    Console.WriteLine("\nSomething Wrong Happened!", e);
                }
            }

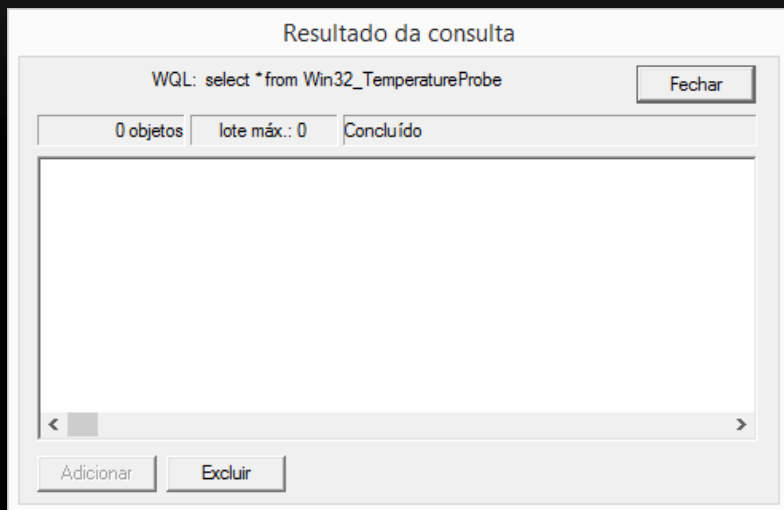
            Console.WriteLine("This program IS RUNNING in a virtual machine!");
        }
    }
}
```

ALEXANDRE BORGES – IT IS NOT ALLOWED TO COPY OR REPRODUCE THIS SLIDE.

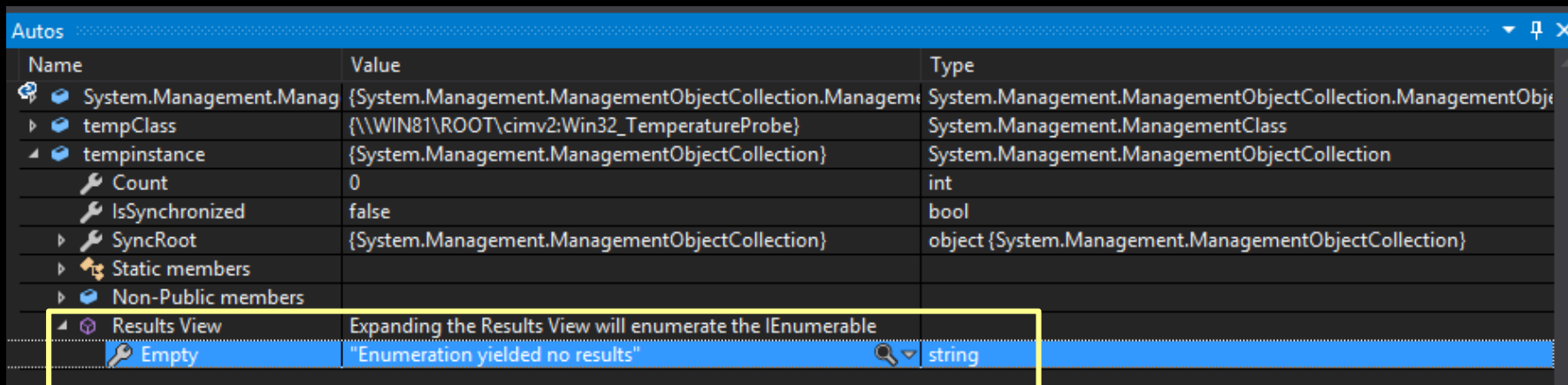
ANTI-VM TECHNIQUES

▶ [26]	{System.Management.PropertyData}	object {System.Management.PropertyData}
▶ [27]	{System.Management.PropertyData}	object {System.Management.PropertyData}
IsArray	false	bool
IsLocal	true	bool
Name	"Status"	string
Origin	"CIM_ManagedSystemElement"	string
Qualifiers	{System.Management.QualifierDataCollection}	System.Management.QualifierDataCollection
Type	String	System.Management.CimType
Value	"OK"	object {string}
▶ Non-Public member		
▶ [28]	{System.Management.PropertyData}	object {System.Management.PropertyData}
▶ [29]	{System.Management.PropertyData}	object {System.Management.PropertyData}
▶ [30]	{System.Management.PropertyData}	object {System.Management.PropertyData}
▶ [31]	{System.Management.PropertyData}	object {System.Management.PropertyData}
▶ [32]	{System.Management.PropertyData}	object {System.Management.PropertyData}
▶ [33]	{System.Management.PropertyData}	object {System.Management.PropertyData}
▶ [34]	{System.Management.PropertyData}	object {System.Management.PropertyData}
▶ Qualifiers	{System.Management.QualifierDataCollection}	System.Management.QualifierDataCollection
▶ Scope	{System.Management.ManagementScope}	System.Management.ManagementScope
▶ Site	null	System.ComponentModel.ISite
▶ SystemProperties	{System.Management.PropertyDataCollection}	System.Management.PropertyDataCollection
▶ Static members		
▶ Non-Public members		

ANTI-VM TECHNIQUES



There is **not** support for acquiring temperature data in virtual machines. Therefore, **malwares** are able to know whether they are running on virtual machines or not. 😊



ANTI-VM TECHNIQUES

❖ Physical Host:

```
C:\Users\Administrador\source\repos\TestVM_3\TestVM_3\bin\Debug>  
VM_Test3c.exe
```

Status: OK Thus, the program is running in a physical host!

❖ Virtual Machine

```
C:\Users\Administrador\source\repos\VM_Test3c\VM_Test3c\bin\Debug>  
VM_Test3c.exe
```

This program IS RUNNING in a virtual machine!

ANTI-VM TECHNIQUES

- ✓ Additional informations about detection of virtualized environments:
 - ✓ **Win32_Processor class** / **ProcessorId** attribute to acquire **NumberOfCores** (most of the time, there is only one core in virtualized systems).
 - ✓ **Win32_NetworkAdapterConfiguration WMI class** represents the attributes and behaviors of a network adapter. Therefore, it is possible to use this class to acquire MAC addresses to detect the possible virtual machines.
- ✓ **MAC Addresses:**
 - ✓ 00:16:3E -- Xen
 - ✓ 00:1C:42 -- Parallels
 - ✓ 00:00:27 -- VirtualBox
 - ✓ 00:05:69 -- VMware
 - ✓ 00:50:56 -- VMware
 - ✓ 00:0C:29 -- VMware
 - ✓ 00:1C:14 -- VMware
- ✓ **IP_ADAPTER_INFO** structure
- ✓ **Malloc(IP_ADAPTER_INFO)**
- ✓ **GetAdaptersInfo()**

ANTI-VM TECHNIQUES

- VMware: VGAuthService.exe
 - VMware: vmacthlp.exe
 - VMware: vmtoolsd.exe
 - VMware: vmwaretray.exe
 - VMware: vmwareuser
 - VirtualBox: vboxservice.exe
 - VirtualBox: vboxtray.exe
 - Parallels: prl_cc.exe
 - Parallels: prl_tools.exe
 - VirtualPC: vmsrvc.exe
 - VirtualPC: vmusrv.exe
 - Xen: xenservice.exe
 - QEMU: qemu-ga.exe
-
- CreateToolhelp32Snapshot()
 - Process32First()
 - Process32Next()

Breakpoints		CPU	Graph	Log	Notes	Memory Map	Call Stack	SEH	Script	Symbols
Type	Address	Module/Label/Exception				State	Disassembly			
Software										
	00A81F89	t_001e0000.bin				Enabled	call <JMP.&Process32Next>			
	00A81F96	t_001e0000.bin				Enabled	push dword ptr ss:[ebp-4]			
	00A83D9B	t_001e0000.bin				Enabled	mov dword ptr ss:[ebp-AD4],t_001e0000.A86084			
	7564E2DC	kernelbase.dll				Enabled	push dword ptr ds:[eax+18]			
	75AD9C5B	<ole32.dll.CoCreateInstance>				Enabled	mov edi,edi			
	75DA3C29	<shell32.dll.ShellExecuteW>				Enabled	mov edi,edi			
	76AF204D	<kernel32.dll.CreateProcessW>				Enabled	mov edi,edi			
	76AF2074	kernel32.dll				Enabled	call <kernel32.CreateProcessInternalW>			
	76AF2082	<kernel32.dll.CreateProcessA>				Enabled	mov edi,edi			
	76AF20A9	kernel32.dll				Enabled	call <kernel32.CreateProcessInternalA>			
	76B31947	<kernel32.dll.DeleteFileW>				Enabled	mov edi,edi			
	76B345DA	<kernel32.dll.DeleteFileA>				Enabled	mov edi,edi			
	76B39633	<kernel32.dll.MapViewOfFile>				Enabled	mov edi,edi			
	76B39BE1	<kernel32.dll.CreateDirectoryW>				Enabled	mov edi,edi			
	76B3DEE2	<kernel32.dll.CreateThread>				Enabled	mov edi,edi			
	76B409C2	<kernel32.dll.CreateProcessInternalW>				Enabled	push 638			
	76B45616	<kernel32.dll.WriteFile>				Enabled	mov edi,edi			
	76B4CB0C	<kernel32.dll.CreateProcessInternalA>				Enabled	push 88			
	76BF61C0	<wininet.dll.HttpQueryInfoA>				Enabled	mov edi,edi			
	7736E310	<urlmon.dll.URLDownloadToFileW>				Enabled	mov edi,edi			

0016E774	76AF0000	kernel32.76AF0000
0016E778	00A872B4	"python.exe"
0016E77C	00A872C0	"pythonw.exe"
0016E780	00A872CC	"prl_cc.exe"
0016E784	00A872D8	"prl_tools.exe"
0016E788	00A872E8	"vmsrvc.exe"
0016E78C	00A872F4	"vmusrvc.exe"
0016E790	00A87300	"xenservice.exe"
0016E794	00A87310	"vboxservice.exe"
0016E798	00A87320	"vboxtray.exe"
0016E79C	00A87330	"vboxcontrol.exe"
0016E7A0	00A87340	"vmwareservice.exe"
0016E7A4	00A87354	"vmwaretray.exe"
0016E7A8	00A87364	"tpautoconnsvc.exe"
0016E7AC	00A87378	"vmtoolsd.exe"
0016E7B0	00A87388	"vmwareuser.exe"
0016E7B4	00A87398	"sbiedll.dll"
0016E7B8	00A873A4	"sbiedllx.dll"
0016E7BC	00A873B4	"dir_watch.dll"
0016E7C0	00A873C4	"wpespy.dll"

ANTI-VM TECHNIQUES

- ✓ Many malwares authors have tried to detect **Cuckoo sandboxes**. As Cuckoo **saves the hooking information (fs:[tls_hooking_information])** and the **returns address in the TLS**, so it is possible to check the Cuckoo's presence by:
 - ✓ reading the **address of this hook information (int hook_1)**, which is usually saved at **fs:0x44**.
 - ✓ adding the size of the saved hooking information to its address (**location_1 = hook_1 + size**).
 - ✓ adding the **location_1** variable to the usual extra space used by Cuckoo for saving the hook information (**location_2 = location_1 + reserved space**).
 - ✓ Of course, if there is something (**saved return address**) within this address range (**location_1 to location_2**), so the malware can be running on Cuckoo.

ANTI-VM TECHNIQUES

Computer

- HKEY_CLASSES_ROOT
- HKEY_CURRENT_USER
- HKEY_LOCAL_MACHINE**
 - BCD00000000
 - HARDWARE**
 - ACPI
 - DESCRIPTION
 - DEVICEMAP**
 - KeyboardClass
 - PointerClass
 - Scsi**
 - Scsi Port 2
 - Scsi Bus 0**
 - Initiator Id 255
 - Target Id 0**
 - Logical Unit Id 0
 - Target Id 1**
 - Logical Unit Id 0
 - SERIALCOMM
 - VIDEO
 - RESOURCEMAP
 - SAM
 - SECURITY

Name	Type	Data
(Default)	REG_SZ	(value not set)
DeviceType	REG_SZ	DiskPeripheral
Identifier	REG_SZ	VMware, VMware Virtual S1.0
InquiryData	REG_BINARY	00 00 02 02 1f 00 00 73 56 4d 77 61 72 65 2c 20 56 4d...

- ✓ Detect if the process is running under WoW
- ✓ RegOpenKeyEx() (it depends on the first step)
- ✓ RegCloseKey()

Computer\HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\Scsi\Scsi Port 2\Scsi Bus 0\Target Id 1\Logical Unit Id 0

ANTI-VM TECHNIQUES

- ✓ **VMware:SOFTWARE\VMware, Inc.\VMware Tools**
- ✓ **HyperV: SOFTWARE\Microsoft\Virtual Machine\Guest\Parameters**
- ✓ **VirtualBox:SYSTEM\ControlSet001\Services\VBoxGuest**
- ✓ **VirtualBox:SYSTEM\ControlSet001\Services\VBoxMouse**
- ✓ **VirtualBox:SYSTEM\ControlSet001\Services\VBoxService**
- ✓ **VirtualBox:SYSTEM\ControlSet001\Services\VBoxSF**
- ✓ **VirtualBox:SYSTEM\ControlSet001\Services\VBoxVideo**
- ✓ **VirtualBox: HARDWARE\ACPI\DSDT\VBOX__**
- ✓ **VirtualBox:HARDWARE\ACPI\FADT\VBOX__**
- ✓ **VirtualBox:HARDWARE\ACPI\RSMT\VBOX__**
- ✓ **VirtualBox:SOFTWARE\Oracle\VirtualBox Guest Additions**

ANTI-VM TECHNIQUES

Installed VMware Drivers:

- "system32\drivers\vmci.sys"
 - "system32\drivers\vmhgfs.sys"
 - "system32\drivers\vmmemctl.sys"
 - "system32\drivers\vmmouse.sys"
 - "system32\drivers\vmhgfs.sys"
 - "system32\drivers\vm3dmp.sys"
 - "system32\drivers\vmmouse.sys"
 - "system32\drivers\vmrawdsk.sys"
 - "system32\drivers\vmusbmouse.sys"
- 
- GetWindowsDirectory()
 - PathCombine()
 - GetFileAttributes()

ANTI-VM TECHNIQUES

- ✓ There are dozens of other checks:
 - ✓ mouse (inactivity): **GetCursorPos()**
 - ✓ memory (small size): **GlobalMemoryStatusEx()** (Retrieves information about the system's current usage of both physical and virtual memory).
 - ✓ disk drivers characteristics (strings like wmare, vbox and so on):
 - ✓ The **SetupDiGetClassDevs()** function returns a handle to a device information set that contains requested device information elements for a local computer.
 - ✓ **SetupDiEnumDeviceInfo()** function returns a **SP_DEVINFO_DATA** structure that specifies a device information element in a device information set.
 - ✓ **SetupDiGetDeviceRegistryProperty()** function retrieves a specified Plug and Play device property (**SPDRP_HARDWAREID**)

ANTI-VM TECHNIQUES

- ✓ Malwares have checked the existence of important tools, which could be **running** in the system:

- | | |
|---------------------|------------------------|
| ✓ tcpview.exe | ✓ ImmunityDebugger.exe |
| ✓ autorunsc.exe | ✓ Wireshark.exe |
| ✓ regmon.exe | ✓ idaq.exe |
| ✓ procexp.exe | ✓ idaq64.exe |
| ✓ ProcessHacker.exe | ✓ ollydbg.exe |
| ✓ procmon.exe | ✓ windbg.exe |
| ✓ autoruns.exe | ✓ x64dbg.exe |

- ✓ Malwares have also checked the existence of these same tools in the **PATH** variable and **standard installation locations**. 😊



THANK YOU FOR ATTENDING MY LECTURE.

➤ Twitter:

@ale_sp_brazil
@blackstormsecbr

- ✓ Malware and Security Researcher.
- ✓ Speaker at DEFCON USA 2018
- ✓ Consultant, Instructor and Speaker on Malware Analysis, Memory Analysis, Digital Forensics and Rookits.
- ✓ Reviewer member of the The Journal of Digital Forensics, Security and Law.
- ✓ Referee on Digital Investigation: The International Journal of Digital Forensics & Incident Response
- ✓ Member of Digital Law and Compliance Committee at OAB/SP
- ✓ Instructor at Oracle and (ISC)2. Ex-instructor at Symantec.

➤ Website: <http://blackstormsecurity.com>

➤ LinkedIn: <http://www.linkedin.com/in/aleborges>

➤ E-mail: alexandreborges@blackstormsecurity.com