反击恶意软件: 源头做起不挖坑

DEF CON CHINA 1.0 (2019)



by Alexandre Borges

ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER



- ✓ Malware and Security Researcher.
- ✓ Speaker at DEFCON USA 2018
- ✓ Speaker at HITB 2019 Amsterdam
- ✓ Speaker at CONFidence Conf. 2019
- ✓ Speaker at BSIDES 2018/2017/2016
- ✓ Speaker at H2HC 2016/2015
- ✓ Speaker at BHACK 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.

目录:

- ❖ 介绍
- ❖ 反逆向和虚拟封装器
- ❖ METASM
- ❖ MIASM
- ❖ DTRACE on Windows
- ❖ 反虚拟化
- ❖ 结论

介绍

- ✓ 我们日常处理的恶意软件样本中都会使用几个已知的包例如： ASPack, Armadillo, Petite, FSG, UPX, MPRESS, NSPack, PECompact, WinUnpack 等等. 他们中的大多数很容易用一个脚本来解压缩。

- ✓ 我们也知道一些经常用来创建和分配内存的API，例如：

  - ✓ VirtualAlloc/Ex( )
  - ✓ HeapCreate( ) / RtlCreateHeap( )
  - ✓ HeapReAlloc( )
  - ✓ GlobalAlloc( )
  - ✓ RtlAllocateHeap( )

- ✓ 此外，我们还知道如何使用调试器、断点和从内存中转储未打包的内容来解压缩它们。来自Hasherezade的pe-sieve非常优秀。☺

- ✓ 当我们意识到恶意软件使用了一些定制的打包技术时，仍然可以从内存中转储它，用Python代码修复ImageAddress字段，并在IDA Pro中使用impscan插件对各自的IAT进行分析:

  - ✓ export VOLATILITY_PROFILE=Win7SP1x86
  - ✓ python vol.py -f memory.vmem procdump -p 2096 -D . --memory (to keep slack space)
  - ✓ python vol.py -f memory.vmem impscan --output=idc -p 2096

x64dbg
script
1/3

```
//#################################################################
// 文件名字     : dumpexe.txt (first draft)
// 注释         : 转储包含可执行程序的内存段
// 作者         : Alexandre Borges
// 日期         : today
//#################################################################

entry:

msg "转储包含可执行程序的模块."
msg "在继续之前，您必须在EP"
bc                      // 清楚已有断点
bphwc                   // 清楚已有硬件断点
bp VirtualAlloc         // 在VirtualAlloc设置断点
erun                    // 运行并将所有的第一个异常传递给应用程序

core:

sti                     // 单步执行
sti                     // 单步执行
sti                     // 单步执行
sti                     // 单步执行
sti                     // 单步执行
```

x64dbg
script 2/3

```
find cip,"C2 1000“   // 找到VirtualAlloc的返回点
bp $result                    // 设置断点
erun                 // 运行并将所有的第一个异常传递给应用程序
cmp eax,0                     // 测试eax(没有分配内存)是否等于零
je pcode             // 跳转到pcode标签
bpm eax,0,x          // 设置可执行内存断点，如果中断并恢复它。
erun                 //运行并将所有的第一个异常传递给应用程序

//尝试查找模块内存中是否有“This program”字符串.
findall $breakpointexceptionaddress,"5468697320707 26F6772616D”

cmp $result,0        // 检查是否找到
je pcode             // 跳转到pcode标签
$dumpaddr = mem.base($breakpointexceptionaddress)     //找到存储基址.
$size = mem.size($breakpointexceptionaddress)         //找到存储的大小.
savedata :memdump:,$dumpaddr,$size                    //转储段.
msgyn "Memory dumped! Do you want continue?“          //显示会话
cmp $result,1                                         //检查选择
je scode             // 跳转到Scode标签
bc                   // 清除已有断点
bphwc                // 清除已有硬件断点
ret                  // 退出
```

x64dbg
script 3/3

```
pcode:

msgyn "没有PE文件!要继续吗?"
cmp $result,0                  // 检查我们是否不想继续
je final
sti                            //单步执行
erun                           // 运行并将所有第一个异常传递给应用程序
jmp core                       // 跳转到core标签

scode:

msg "Let's go to next dump“    // 显示会话
erun                           // 运行并将所有第一个异常传递给应用程序
jmp core                       // 跳转到core标签

final:

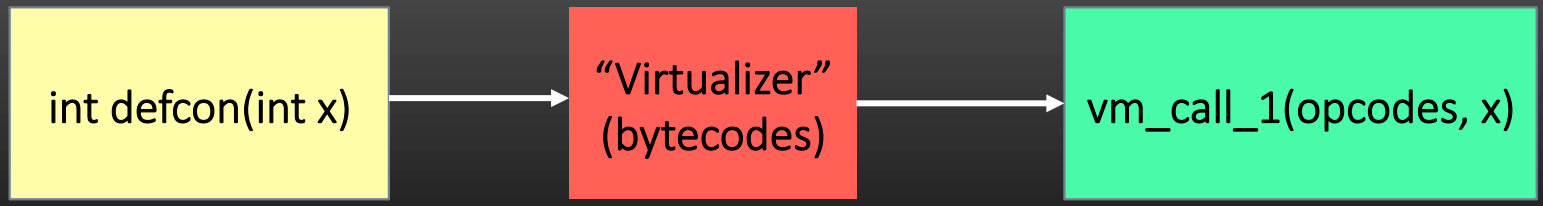
bc                             // 清除已有断点
bphwc                          // 清除已有硬件断点
ret                            // 退出
```

反逆向

- ✓ 混淆的目的是保护软件不被逆向分析，知识产权不被侵犯，对我们而言是保护恶意代码。☺ 事实上，混淆并不能真正保护软件，但它可以增加逆向人员的难度。
- ✓ 因此，混淆通过强制逆向人员花费资源和时间来破译代码。
- ✓ 分析恶意人员用VBA和Powershell编写的常见的用户恶意软件时，我们都会看到混淆的代码，所以这似乎没什么大不了的。
- ✓ 我们可以使用IDA Pro SDK编写插件来扩展IDA Pro的功能，分析一些代码和数据流，甚至自动解压奇怪的恶意文件。
- ✓ 此外，如果您在分析修改后的MBR时遇到问题，那么您甚至可以编写一个加载器来加载MBR结构并在IDA Pro中分析它。☺
- ✓ 不幸的是，有些打包程序和保护程序，如VMprotect、Themida、Arxan和Agile .Net，使用了现代的混淆技术，因此代码的逆向过程非常复杂。

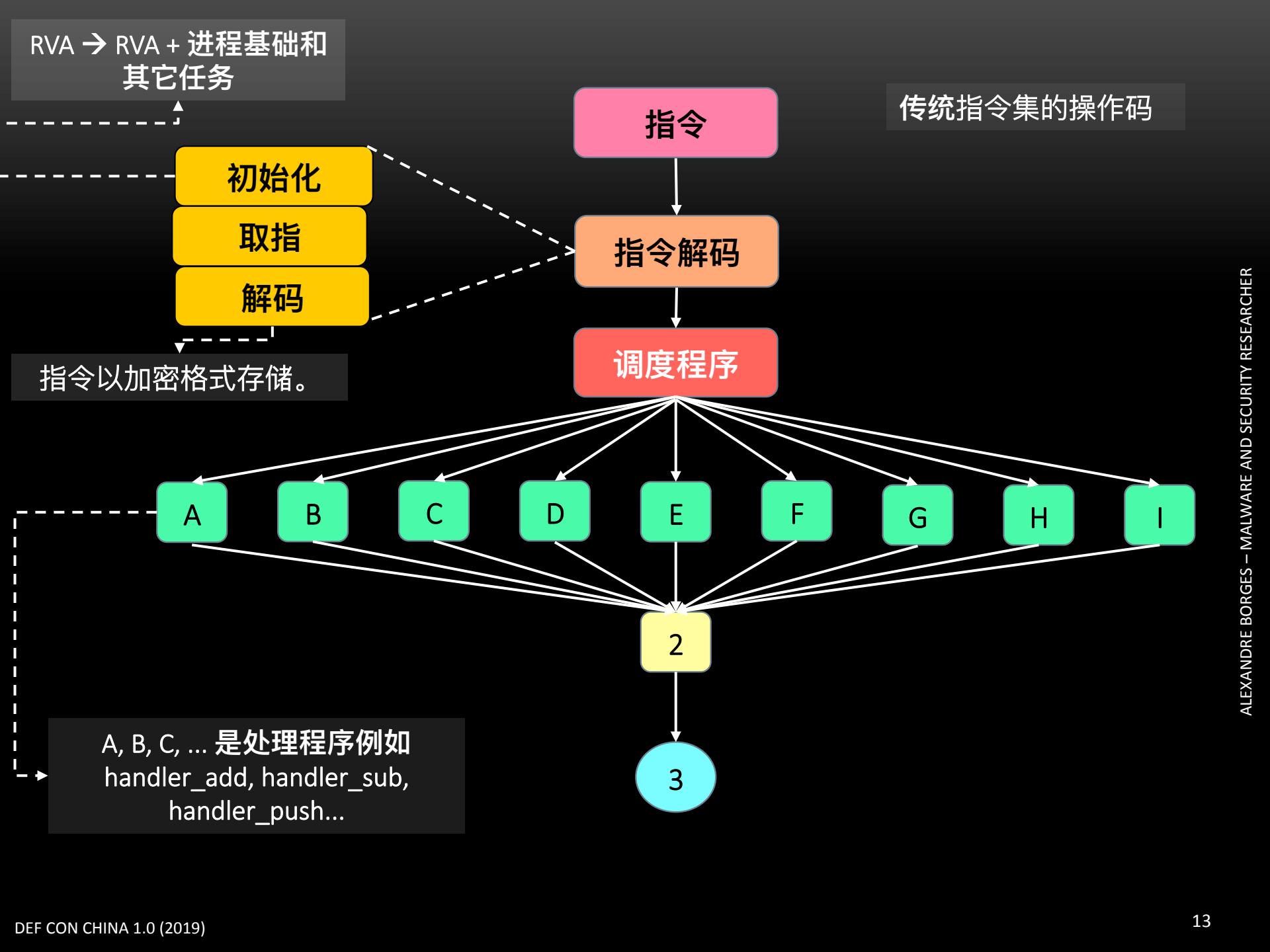
- ✓ 大多数保护程序在64位代码中使用(恶意软件)。
- ✓ 原始IAT从原始代码中删除（通常适用于任何包装器）。然而，像Themida这样的包装商的IAT只保留一个功能(TlsSetValue)。
- ✓ 他们中几乎全部包含字符串加密.
- ✓ 它们保护和检查内存完整性。因此，不可能从内存中转储干净的可执行文件(例如，使用波动性)，因为原始指令没有在内存中解码。
- ✓ 指令(x86/x64代码)被虚拟化并转换为虚拟机指令(RISC指令)。
- ✓ .NET 保护器重命名类, 方法, 域和外部引用.

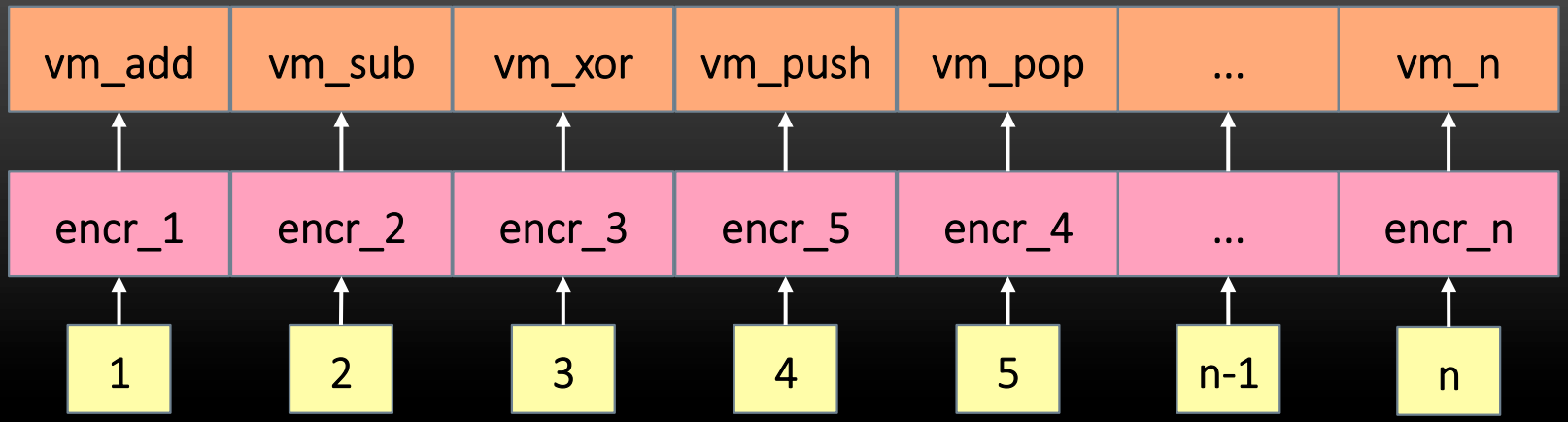
- ✓ 一些包装器可以使用指令加密内存作为额外的内存层。
- ✓ 混淆是基于堆栈的，因此很难静态地处理虚拟化代码。
- ✓ 虚拟化代码是多态的，所以有许多表示引用相同的CPU指令。
- ✓ 有很多假的push指令。
- ✓ 有许多死代码和无用代码。
- ✓ 有一些代码使用无条件跳转重新排序。
- ✓ 所有的混淆器都使用代码扁平化。
- ✓ 打包器几乎没有反调试器和反虚拟化技巧。然而，几个月前，我看到了一个不太常见的基于温度的anti-vmware技巧(稍后会详细介绍)。



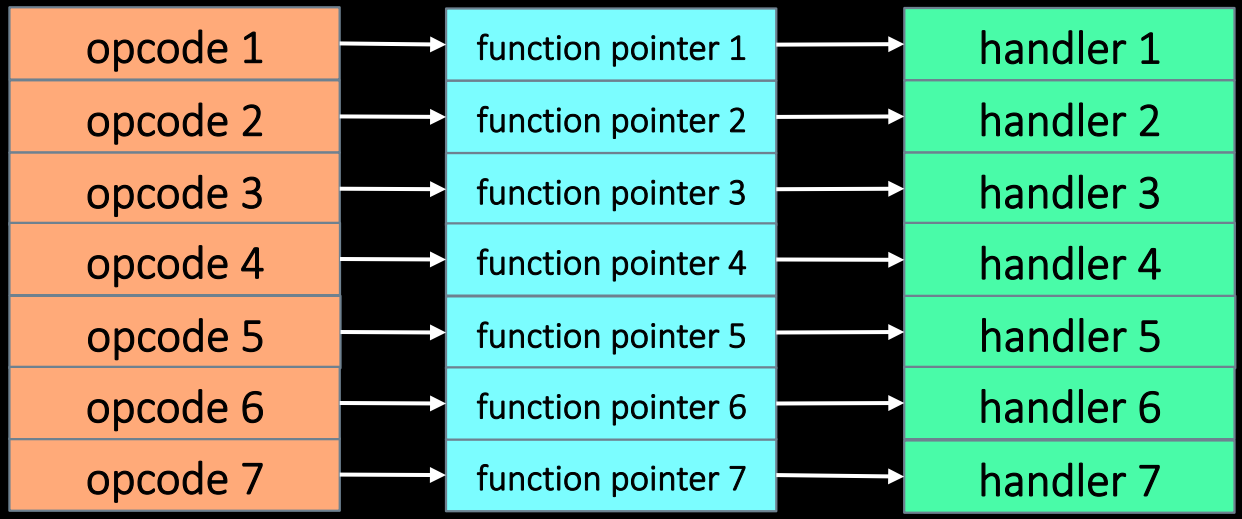
获取字节，将其解码为指令并将其分派给处理程序

- 使用虚拟机的保护器引入模糊代码:
  - ✓ 上下文切换组件，它将注册表和标志信息“传输”**到VM上下文**(虚拟机)。**相反的**移动稍后从VM**机器和本机**(x86/x64)**上下文中完成**(适合在解包过程中保持在C结构中 ☺)
  - ✓ **从**本机寄存器到虚拟寄存器的这种“转换”**可以**是一对一的，但并不总是这样。
- ✓ 在虚拟机内部，循环是:
  - ✓ **获取指令**
  - ✓ **解码**
  - ✓ 找到指向指令的指针，并在处理程序表中查找关联操作码
  - ✓ 调用目标处理程序


RVA → RVA + 进程基础和其它任务
初始化
取指
解码
指令以加密格式存储。
指令
指令解码
调度程序
传统指令集的操作码
A
B
C
D
E
F
G
H
I
2
3
A, B, C, ... 是处理程序例如 handler_add, handler_sub, handler_push...



恢复和解密函数



函数指针表
(可能加密)

- ✓ 逆向虚拟化和打包代码容易吗?当然不是。挑战可能是巨大的☺
- ✓ 记住:混淆是使用任何技巧(包括虚拟化)将代码从A转换为B。
- ✓ 要确定程序是否虚拟化并不容易。
- ✓ 处理程序彼此独立，通常设置为:
  - ✓ 寄存器
  - ✓ 加密密钥
  - ✓ 内存

- ✓ **通常**按指令类型有一个处理程序。
- ✓ 这些处理程序由VM调度器“启动”。
- ✓ **指令的操作数**使用处理程序提供的密钥(初始化代码)**加密**。
- ✓ 有时候，秘钥有4个字节，并且使用异或等操作。 ☺
- ✓ 不能虚拟化**每个函数的**序言和结语。**当心** ☺

- ✓ 你试过打开IDA Pro的封装器吗?第一眼:只有红色和灰色块(非函数和数据)。
- ✓ 最终，数据块可以容纳VM处理程序……
- ✓ 原始代码段可以在程序周围“分割”和“分散”(数据和指令混合在二进制文件中，而不是只有一个指令块)
- ✓ 引用导入函数的指令可以被归零，也可以被NOP替换。☹ 当然，它们稍后将被封装器动态地恢复(重新插入)。
- ✓ “隐藏”函数代码可以复制(memcpy())到VirtualAlloc()分配的内存中，☺ 当然，代码中必须有一个补丁才能获得这些指令。

- ✓ 传统打包程序通常不会虚拟化所有x86指令。
- ✓ 在打包过程之后，通常会看到虚拟化、本机指令和数据之间的混合。
- ✓ 原生API可以重定向到存根代码，存根代码将调用转发给(复制的)原生DLL(从各自的API)。
- ✓ API调用指令将直接引用IAT，通常使用RVA将其转换为短跳转，用于相同的导入地址(“IAT混淆”) ☺
- ✓ 更糟的是，API名称可以散列(如在shellcode中使用的那样)。☺

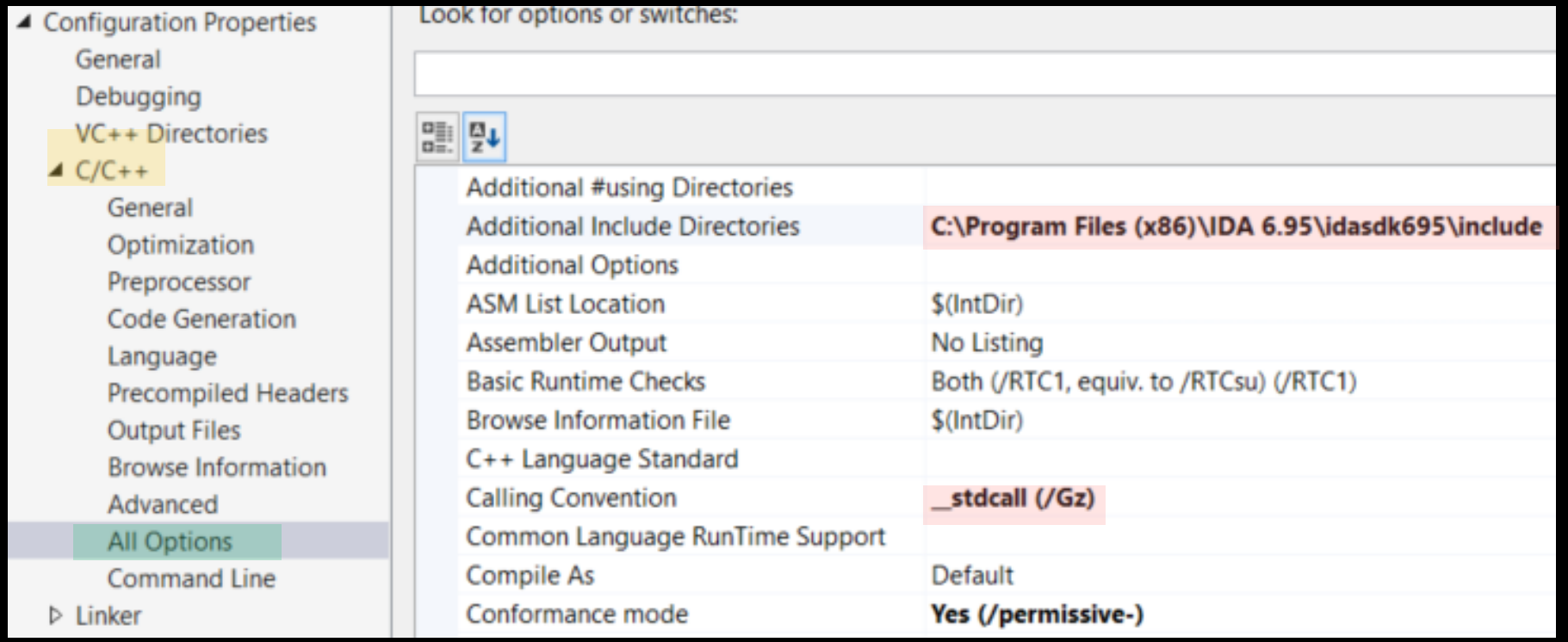
- ✓ 顺便问一下，有多少虚拟化指令存在?
- ✓ 我们是否能够根据操作数及其用途(内存访问、算术、通用等等)将虚拟指令分组?
- ✓ 注意指令的主干，将类似的指令类放在一起(例如跳转指令、直接调用、间接调用等)。
- ✓ 是否基于(类似)x86指令的虚拟化指令?
- ✓ 处理器标志的含义被修改了吗?

- ✓ 从x86模式转换到“虚拟化模式”的“关键指令”是什么？
- ✓ 记住:通常，寄存器和标志(EFLAGS)在“跨越”到VM环境之前保存到堆栈中。
- ✓ 将控制权转移回x86环境需要哪些负责任的指令?
- ✓ 很多时候，在“上下文转换”期间， 参数被推送到堆栈上. ☺

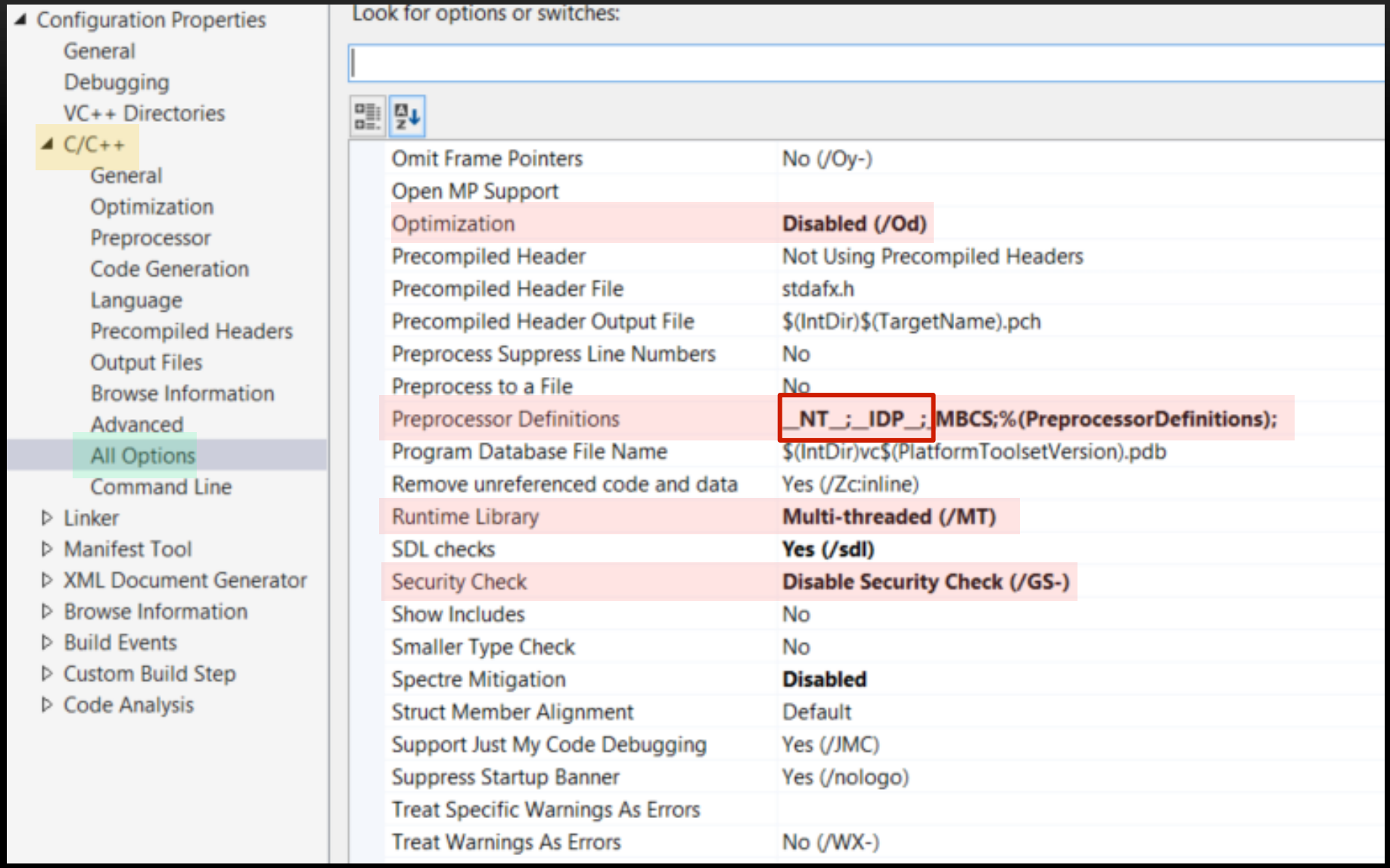
- ✓ 这是快速的建立一个IDA Pro的插件。从这里下载IDA SDK https://www.hex-rays.com/products/ida/support/download.shtml (很可能，你需要一个专业账户)。将其复制到IDA Pro安装目录中的一个文件夹(idasdk695/)。

- ✓ 在 Visual Studio 2017 创建工程(File → New → Create Project → Visual C++ → Windows Desktop → Dynamic-Link Library (DLL)).

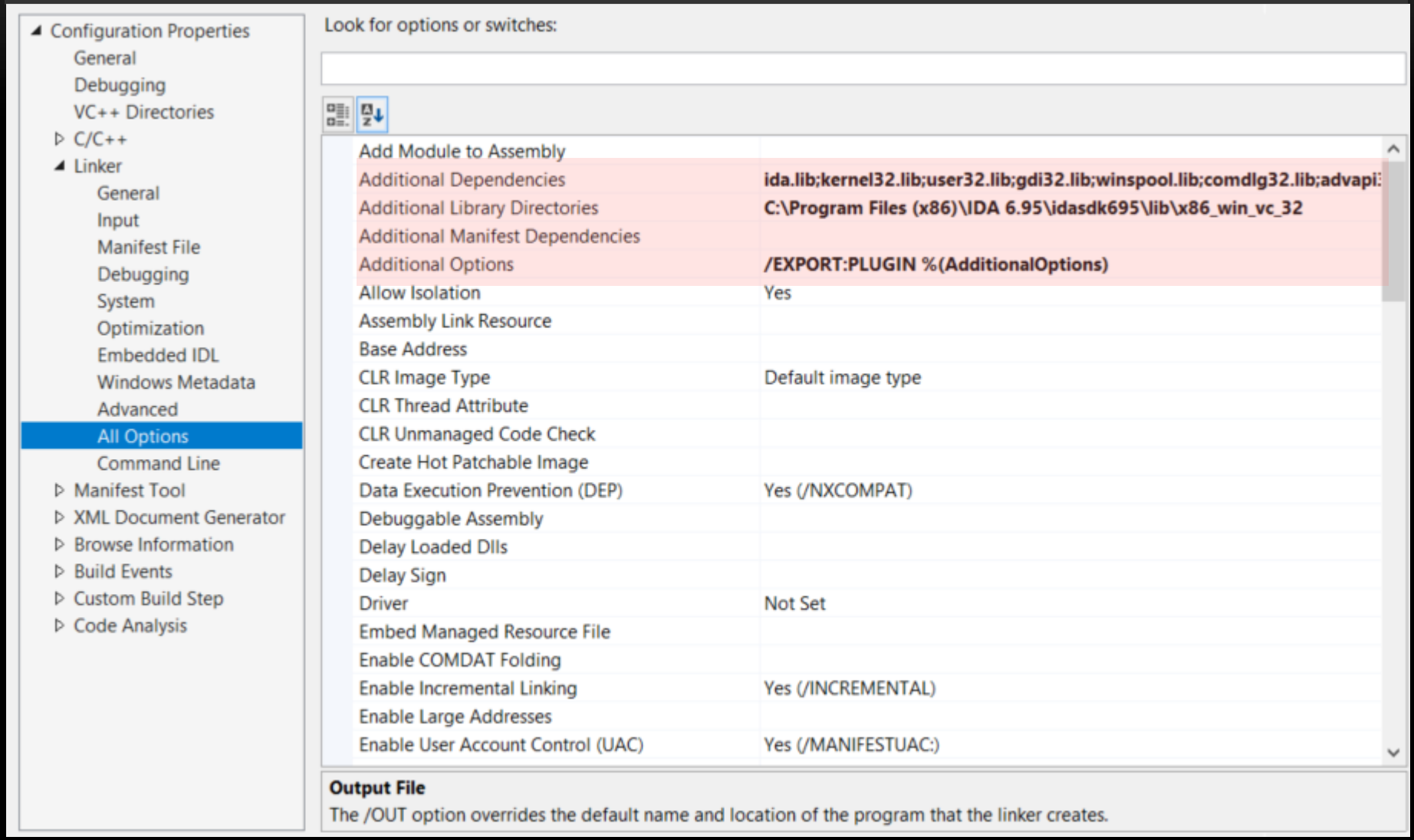
- ✓ 改变一些工程的属性，如这页和下页所示



✓ 包含“__NT__;__IDP__” 在处理器的定义中。并更改运行时库 “Multi-threaded” (MT) (take care: it is NOT /MTd).



- ✓ 添加ida.lib (from C:\Program Files (x86)\IDA 6.95\idasdk695\lib\x86_win_vc_32) 到附加依赖项及其文件夹到附加库目录。
- ✓ 添加 “/EXPORT:PLUGIN” 到附加选项.



```
1  #include <ida.hpp>
2  #include <idp.hpp>
3  #include <loader.hpp>
4  #include <allins.hpp>
5  #include <strlist.hpp>
6  #include <search.hpp>
7
8
9  int IDAP_init()
10 {
11     return PLUGIN_KEEP;
12 }
13
14 void IDAP_term(void)
15 {
16
17 }
18
19 void IDAP_run(int arg)
20 {
21     msg("Hello DEFCON CHINA! We love IDA Pro :)\n\n");
22
23     char defcon[MAXSTR];
24     string_info_t strinfo;
25     char s[] = "[a-z0-9]+[\.]{1,}[a-zA-Z0-9_-]+[\.]{1,}[a-z]{2,}";
26     auto last = BADADDR;
27     auto ea = 0;
28     auto urlcount = 1;
```

不要忘记必要的**申明**。☺

初始化

使插件对idb可用，并将插件加载到内存中。

清除任务

当用户激活插件时调用的函数。

简单(**和不完整**)的URL正则表达式。☺

```
30      for (int x = 0; x < get_strlist_qty(); x++) {
31
32          get_strlist_item(x, &strinfo);
33          if (strinfo.length < sizeof(defcon)) {
34
35              get_many_bytes(strinfo.ea, defcon, strinfo.length);
36
37              {
38                  ea = 0;
39                  ea = find_text(strinfo.ea, 0, 0, s, SEARCH_REGEX);
40
41                  if (ea == strinfo.ea) {
42                      msg("Address 0x%x - URL %d: %s\n", strinfo.ea, urlcount, defcon);
43                      urlcount++;
44                  }
45              }
46          }
47      }
48
49      return;
50
51  }
52
53  char IDAP_comment[] = "The simplest possible plugin";
54  char IDAP_help[] = "DEFCON plugin";
55  char IDAP_name[] = "DEFCON plugin";
56  char IDAP_hotkey[] = "ALT-X";
57
58  plugin_t PLUGIN =
59  {
60      IDP_INTERFACE_VERSION,
61      0,
62      IDAP_init,
63      IDAP_term,
64      IDAP_run,
65      IDAP_comment,
66      IDAP_help,
67      IDAP_name,
68      IDAP_hotkey
69  };
```

它从“strings view”中获取字符串的数量。

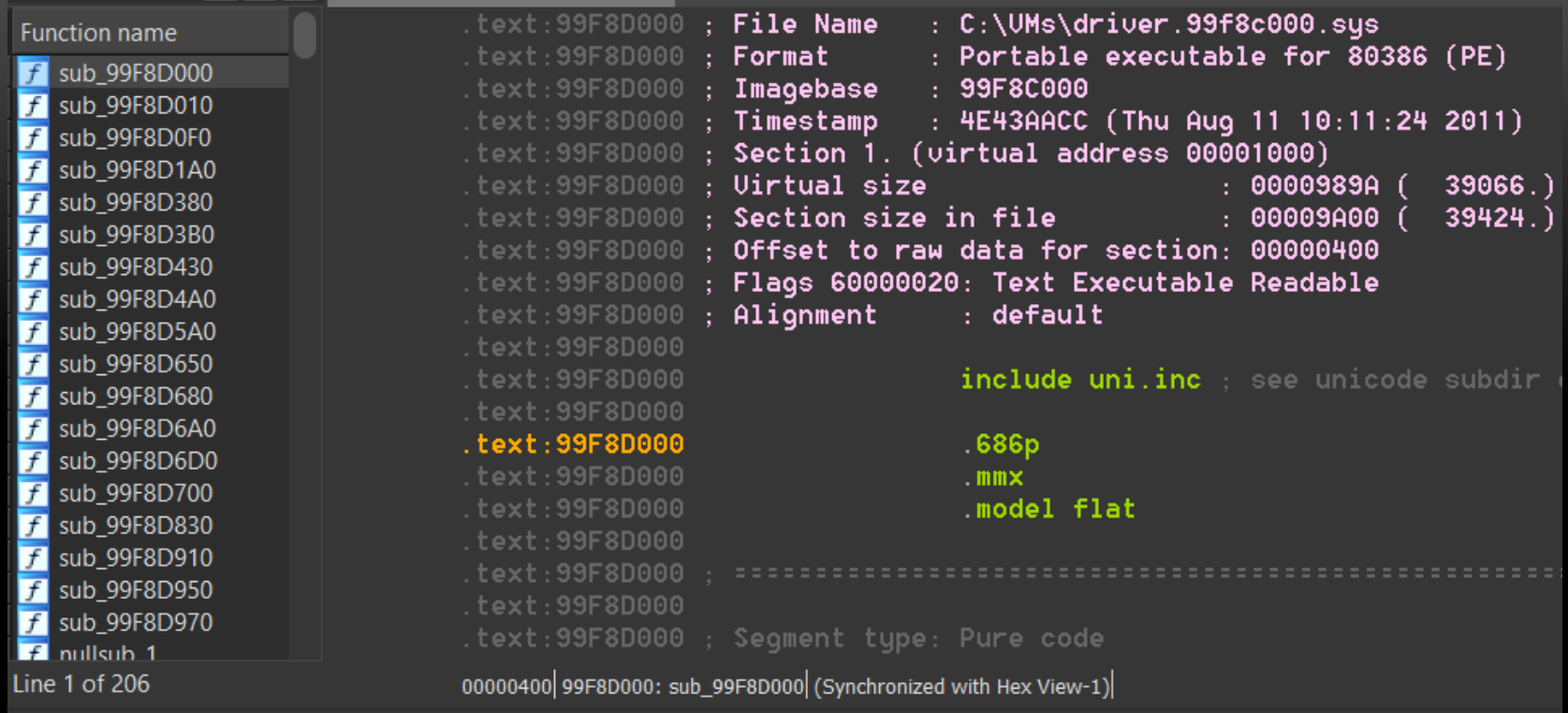
得到string

它检查字符串是否与URL regex匹配.

如果检查, 所以 ea == strinfo.ea. ☺

插件将被ALT-X组合激活. ☺

插件结构



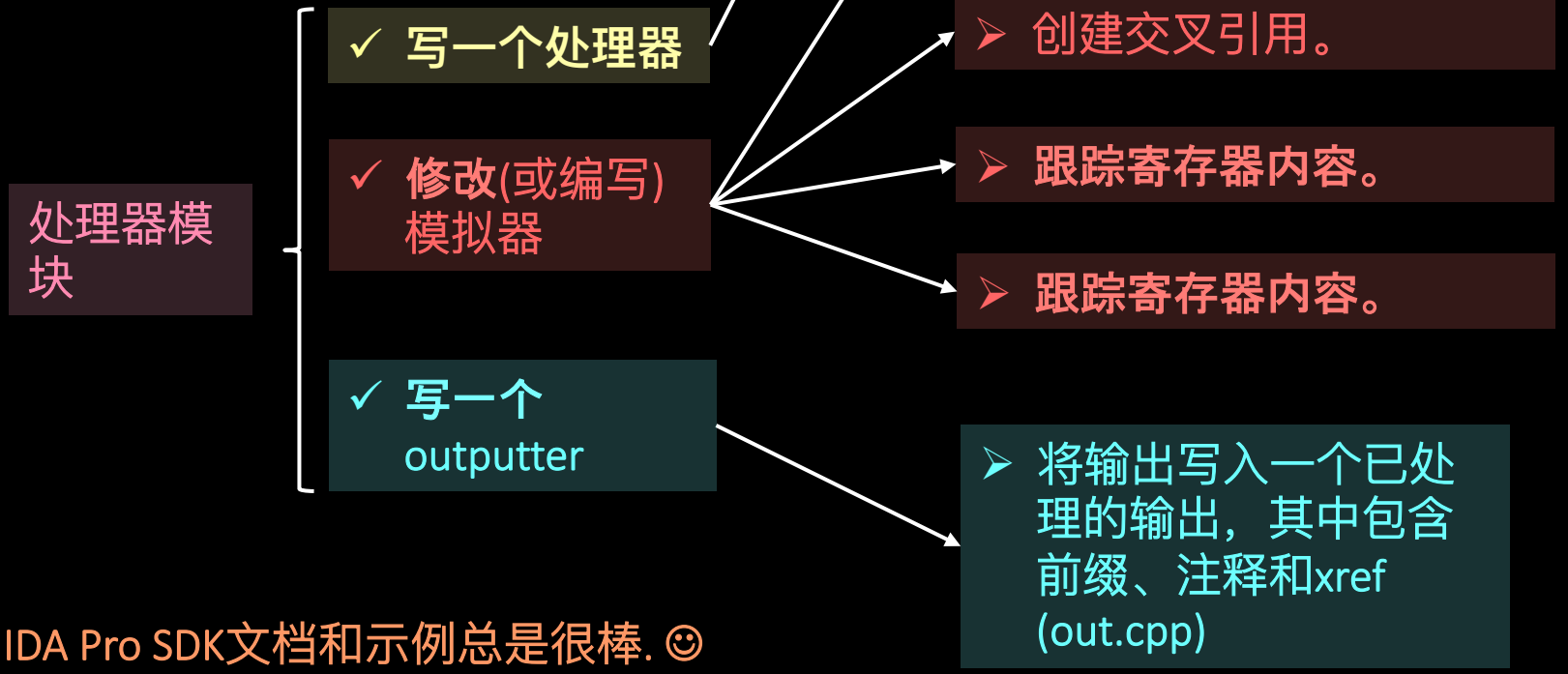

```
Hello DEFCON CHINA! We love IDA Pro :)

Address 0x99f990d8 - URL 1: ntp2.usno.navy.mil
Address 0x99f990eb - URL 2: ntp.adc.am
Address 0x99f990f6 - URL 3: tock.usask.ca
Address 0x99f99104 - URL 4: ntp.crifo.org
Address 0x99f99112 - URL 5: ntp1.arnes.si
Address 0x99f99120 - URL 6: ntp.ucsd.edu
Address 0x99f9912d - URL 7: ntp.duckcorp.org
Address 0x99f9913e - URL 8: www.nist.gov
Address 0x99f9914b - URL 9: clock.isc.org
Address 0x99f99159 - URL 10: time.windows.com
Address 0x99f9916a - URL 11: time2.one4vision.de
Address 0x99f9917e - URL 12: time.cerias.purdue.edu
Address 0x99f99195 - URL 13: clock.fihn.net
```

在此恶意驱动程序中找到url。☺

ALT - X

- ✓ IDA处理器模块仍然是处理虚拟包装程序的最佳方法之一。
- ✓ 请您记住关于如何编写IDA处理器模块的几个要点(正如hex - ray中的Ilfak所提到的):



- ✓ IDA Pro SDK文档和示例总是很棒. ☺

- ✓ 负责将x86指令转换为VM指令的VM解释器代码通常比较混乱。
- ✓ 自己的VM指令也被压缩和加密(主要是xor 'ed)
- ✓ 正如我前面提到的，通常只有一个x86指令对应许多VM指令代码。
- ✓ 有两个堆栈:一个来自x86领域，另一个来自VM领域。
- ✓ 来自虚拟化上下文的堆栈可能会向上增长，这与x86标准不同。
- ✓ 有时候，保护者不会将x86上下文复制到虚拟机中。在这种情况下，它更愿意保存上下文(注册内容+标志)以便稍后使用。

- ✓ **找出**VM**指令的大小**很有趣，我们可以将其放入表示加密密钥、数据、RVA(**位置**)、操作码(类型)等的结构中。

- ✓ 由于自定义虚拟化包装器没有针对每个x86指令的虚拟化指令，所以建议找到针对本机x86指令的处理程序(非虚拟化指令))

- ✓ **通常**，非虚拟化指令的处理程序会在短时间内从VM环境中退出，执行x86指令并返回到虚拟机环境。

- ✓ **在本例中**，x86**指令**也与虚拟化指令一起被加密和压缩。

- ✓ **常量展开**:模糊处理程序使用的一种技术，它用生成相同结果常量值的一组代码来替换内容。
- ✓ **基于模式的混淆**:用一组等价指令交换一条指令。
- ✓ 滥用内联函数。
- ✓ **反VM技术**:防止恶意软件样本在VM中运行。
- ✓ 死代码(垃圾代码):这种技术是通过插入代码来实现的，这些代码的结果将在下一行代码中被覆盖，或者更糟的是，它们将不再被使用。
- ✓ 代码复制:进入同一目的地的不同路径(由虚拟化混淆器使用)。

- ✓ 间接控制 1: 调用指令 →栈指针更新 → 返回跳过一些垃圾代码后的调用指令 (RET x).
- ✓ 间接控制 2: 恶意软件引发异常 →调用已注册的异常 → 新的指令分支。
- ✓ 隐含谓词:虽然显然存在一个求值(条件跳转:jz/jnz)，但结果总是被求值为true(或false)，这意味着无条件跳转。这样，就有了死跳转。
- ✓ 反调试: 使用复杂的技术使分析变慢.
- ✓ 多态性:它由自修改代码(如shell代码)和加密资源(类似于大多数恶意软件样本)生成。

- ✓ **调用堆栈处理**: 通过使用由ret指令组成的指令技巧更改堆栈流，从而隐藏真实的ret。
- ✓ 是否有可能消除虚拟化指令的混淆?是的，可以使用反向递归替换(类似于Metasm中的**回溯特性**，但不相等)。
- ✓ 此外，符号方程系统是另一种很好的方法。（Metasm和MIASM!)
- ✓ 有很多很好的插件，比如Code Unvirtualizer、VMAttack、VMSweeper等等，它们可以用来处理简单的虚拟化问题。
- ✓ 使用简单而有效的隐写术概念作为替换盒(s - box)，指令虚拟化器的一些改进已经出现。

✓ 代码流图扁平化:

- ✓ 该技术用于隐藏程序的实际控制流。
- ✓ 一般来说，其思想是通过删除if语句和循环来中断控制流，在一系列开关用例语句中转换流.

- ✓ **因此，有一个**分发器**将控制流**交给处理程序，每个处理程序将更新指令指针，使之指向下一个要执行的处理程序的值(虚拟化流控制)。
- ✓ **通常有一个**调用存根，它完成从本机指令到虚拟化指令的转换。

- ✓ 这种方法提出了两个反向问题:**从**CISC**到**RISC**指令的映射和从**VM将原始寄存器转换为专用寄存器。
- ✓ 因为要权衡利弊，CFG**只适用于特定的功能**

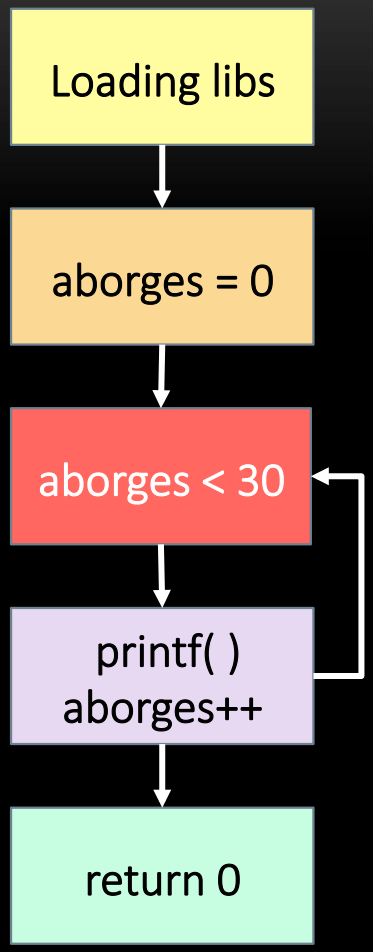
```
#include <stdio.h>

int main (void)

{
        int aborges = 0;
        while (aborges <  30)
        {
                printf("%d\n", aborges);
                aborges++;
        }

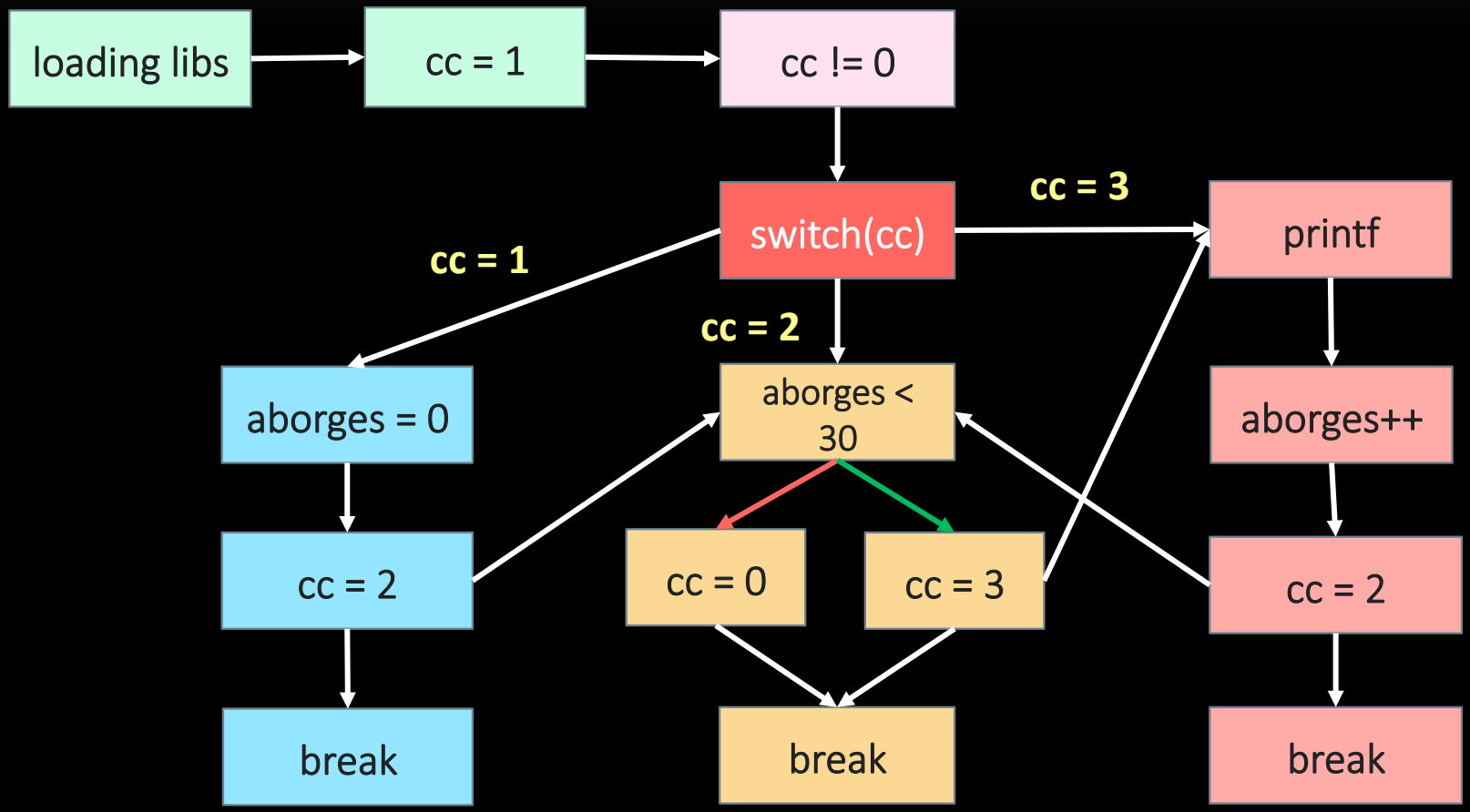
        return 0;

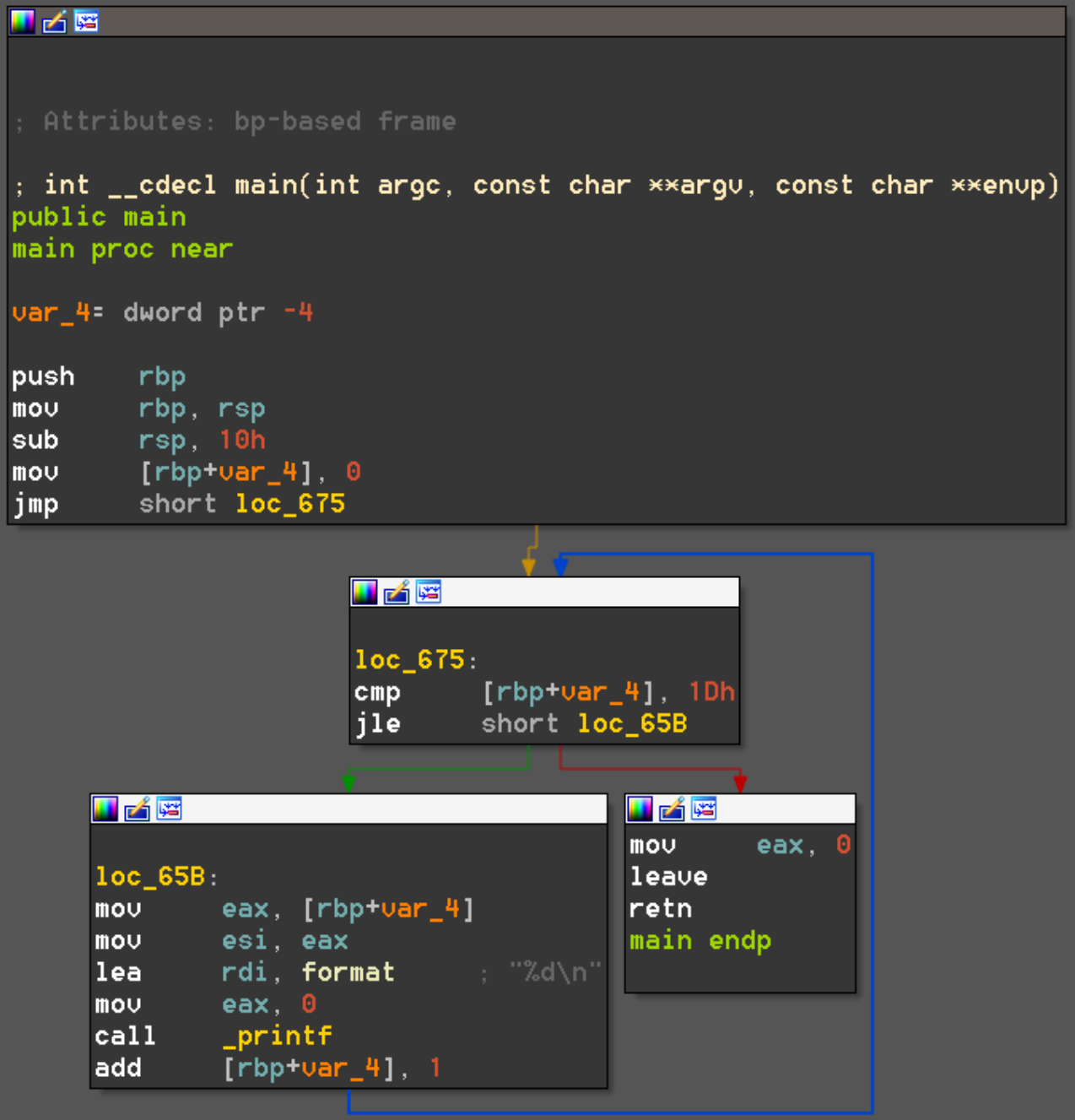
}
```



❖ 缺点:

✓ 损失的性能
✓ 容易识别CFG的扁平化



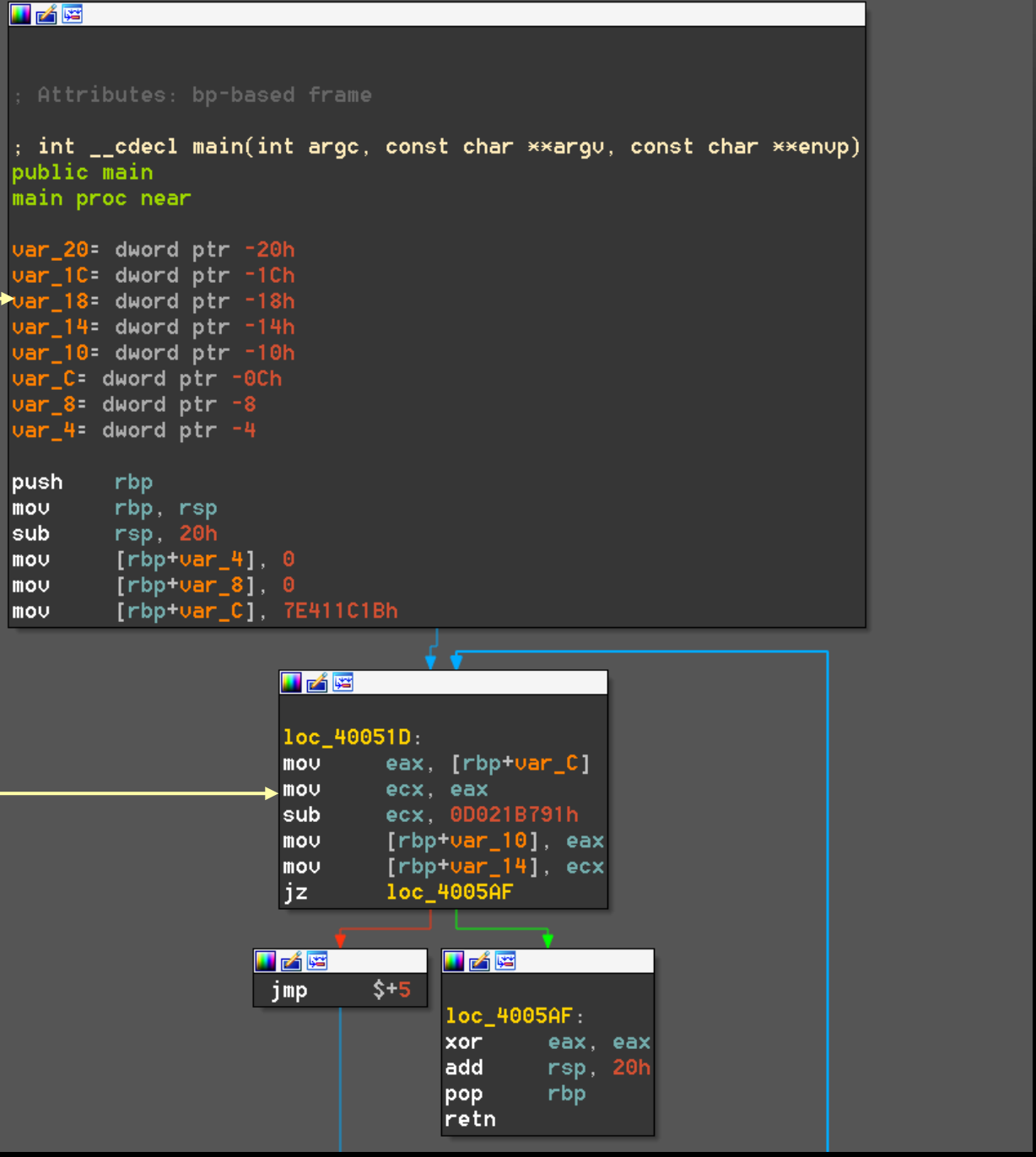

; Attributes: bp-based frame
; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near
var_4= dword ptr -4
push rbp
mov rbp, rsp
sub rsp, 10h
mov [rbp+var_4], 0
jmp short loc_675
loc_675:
cmp [rbp+var_4], 1Dh
jle short loc_65B
loc_65B:
mov eax, [rbp+var_4]
mov esi, eax
lea rdi, format ; "%d\n"
mov eax, 0
call _printf
add [rbp+var_4], 1
mov eax, 0
leave
retn
main endp

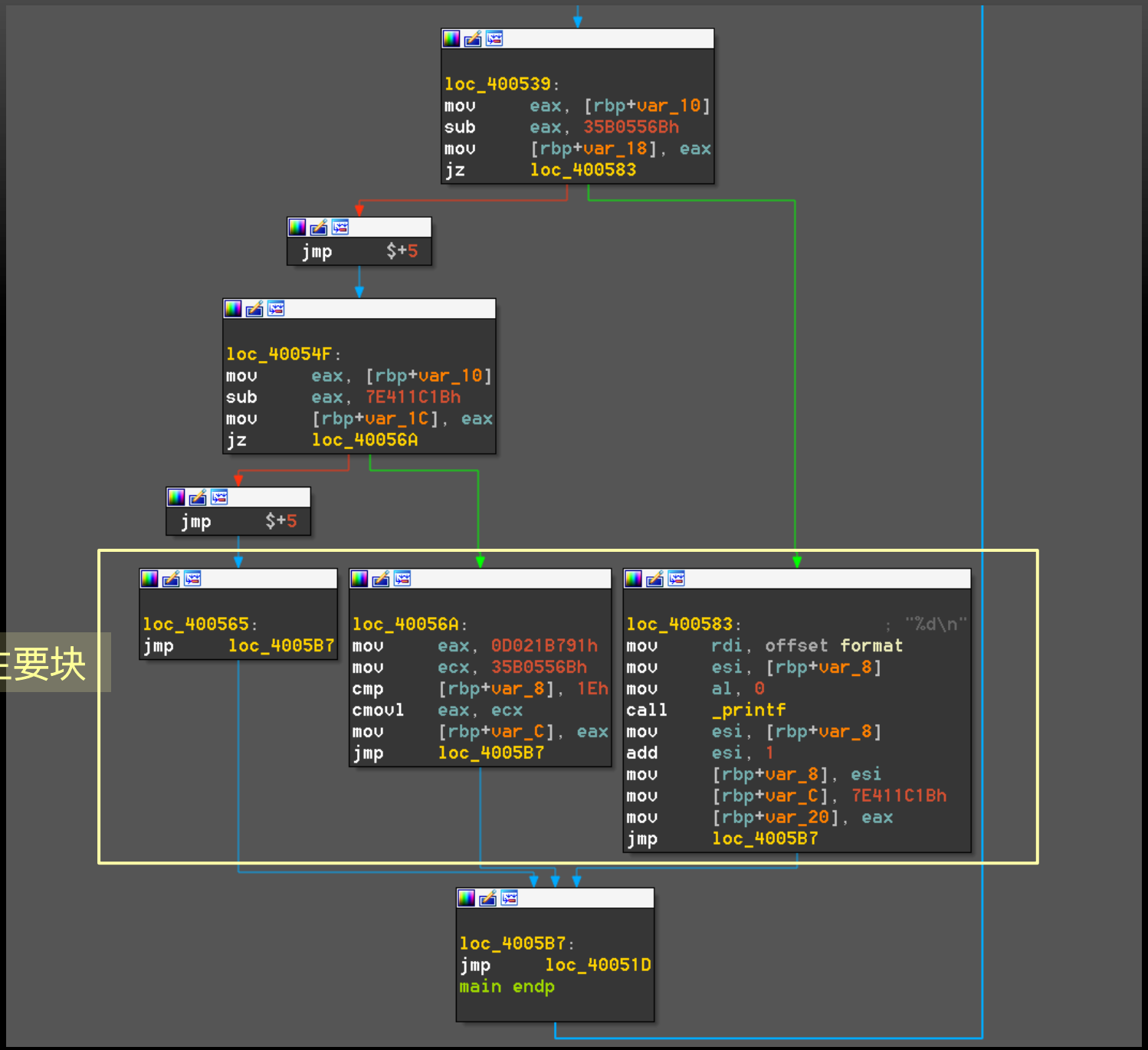
原始程序

- ✓ 用于代码模糊处理的obfusator -llvm是一个非常好的项目。为了安装它建议先添加一个交换文件(因为链接阶段):
  - ✓ fallocate -l 8GB /swapfile
  - ✓ chmod 600 /swapfile
  - ✓ mkswap /swapfile
  - ✓ swapon /swapfile
  - ✓ swapon --show
  - ✓ apt-get install llvm-4.0
  - ✓ apt-get install gcc-multilib (install gcc lib support to 32 bit)
  - ✓ git clone -b llvm-4.0 https://github.com/obfuscator-llvm/obfuscator.git
  - ✓ mkdir build ; cd build/
  - ✓ cmake -DCMAKE_BUILD_TYPE=Release -DLLVM_INCLUDE_TESTS=OFF ../obfuscator/
  - ✓ make -j7
- ✓ 可能会用到:
  - ✓ ./build/bin/clang alexborges.c -o alexborges -mllvm -fla
  - ✓ ./build/bin/clang alexborges.c -m32 -o alexborges -mllvm -fla
  - ✓ ./build/bin/clang alexborges.c -o alexborges -mllvm -fla -mllvm -sub

序言和初始任务

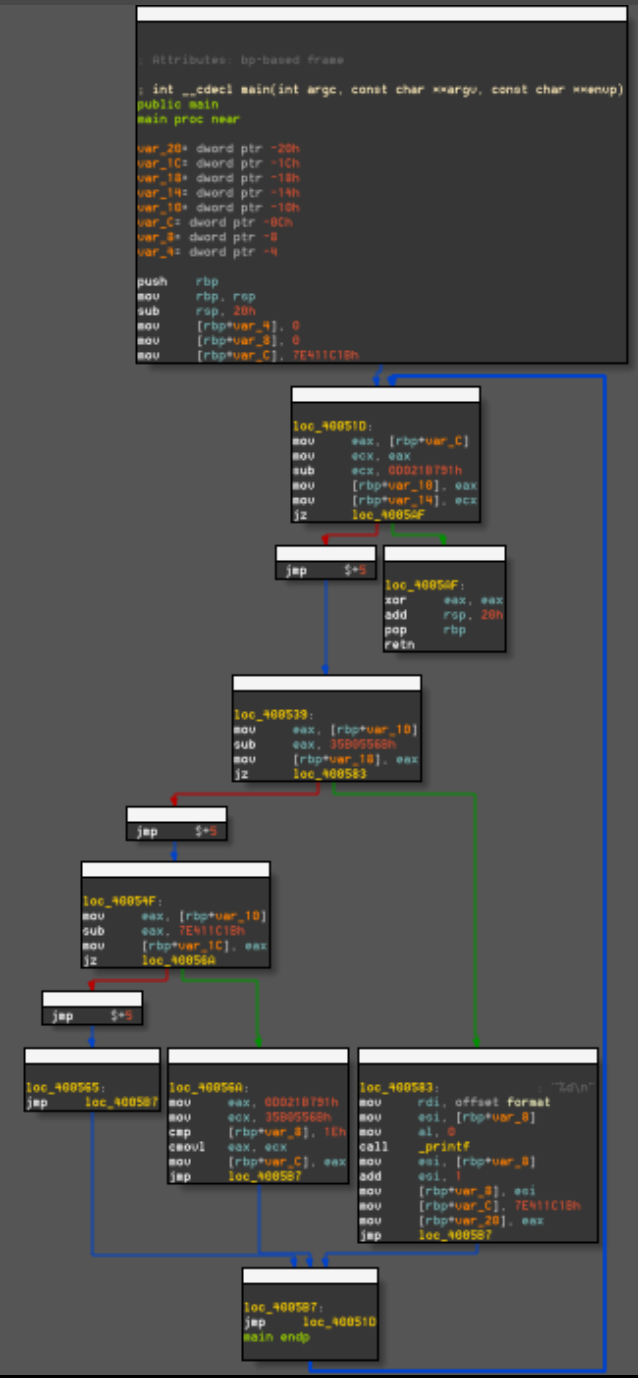
主要调度程序




loc_400539:
mov eax, [rbp+var_10]
sub eax, 35B0556Bh
mov [rbp+var_18], eax
jz loc_400583
jmp $+5
loc_40054F:
mov eax, [rbp+var_10]
sub eax, 7E411C1Bh
mov [rbp+var_1C], eax
jz loc_40056A
jmp $+5
loc_400565:
jmp loc_4005B7
loc_40056A:
mov eax, 0D021B791h
mov ecx, 35B0556Bh
cmp [rbp+var_8], 1Eh
cmovl eax, ecx
mov [rbp+var_C], eax
jmp loc_4005B7
loc_400583: ; "%d\n"
mov rdi, offset format
mov esi, [rbp+var_8]
mov al, 0
call _printf
mov esi, [rbp+var_8]
add esi, 1
mov [rbp+var_8], esi
mov [rbp+var_C], 7E411C1Bh
mov [rbp+var_20], eax
jmp loc_4005B7
loc_4005B7:
jmp loc_40051D
main endp

程序的主要块

模糊代码的概述


Attributes: bp-based frame
; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near
var_20= dword ptr -20h
var_1C= dword ptr -1Ch
var_18= dword ptr -18h
var_14= dword ptr -14h
var_10= dword ptr -10h
var_C= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4
push rbp
mov rbp, rsp
sub rsp, 20h
mov [rbp+var_4], 0
mov [rbp+var_8], 0
mov [rbp+var_C], 7E411C18h
loc_40051D:
mov eax, [rbp+var_C]
mov ecx, eax
sub ecx, 6DD21B79h
mov [rbp+var_10], eax
mov [rbp+var_14], ecx
jz loc_40054F
jmp $+5
loc_40054F:
xor eax, eax
add rsp, 20h
pop rbp
retn
loc_400539:
mov eax, [rbp+var_10]
sub eax, 35B05568h
mov [rbp+var_18], eax
jz loc_400583
jmp $+5
loc_40054F:
mov eax, [rbp+var_10]
sub eax, 7E411C18h
mov [rbp+var_1C], eax
jz loc_40056A
jmp $+5
loc_400565:
jmp loc_400587
loc_40056A:
mov eax, 6DD21B79h
mov ecx, 35B05568h
cmp [rbp+var_8], 1Eh
cmovl eax, ecx
mov [rbp+var_C], eax
jmp loc_400587
loc_400583:
mov rdi, offset format
mov esi, [rbp+var_8]
mov al, 0
call _printf
mov esi, [rbp+var_8]
add esi, 1
mov [rbp+var_8], esi
mov [rbp+var_C], 7E411C18h
mov [rbp+var_20], eax
jmp loc_400587
loc_400587:
jmp loc_40051D
main endp

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3   signed int v3; // eax@5
4   int v4; // eax@8
5   __int64 v6; // [rsp+0h] [rbp-20h]@0
6   signed int v7; // [rsp+14h] [rbp-Ch]@1
7   signed int v8; // [rsp+18h] [rbp-8h]@1
8
9   v8 = 0;
10  v7 = 2118196251;
11  while ( v7 != -803096687 )
12  {
13    if ( v7 == 900748651 )
14    {
15      v4 = printf("%d\n", (unsigned int)v8++, envp, v6, 731796000415206604 8LL);
16      v7 = 2118196251;
17      LODWORD(v6) = v4;
18    }
19    else
20    {
21      HIDWORD(v6) = v7 - 2118196251;
22      if ( v7 == 2118196251 )
23      {
24        v3 = -803096687;
25        if ( v8 < 30 )
26          v3 = 900748651;
27        v7 = v3;
28      }
29    }
30  }
31  return 0;
32 }
```

简单隐含谓词和反汇编技术

```
.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
```

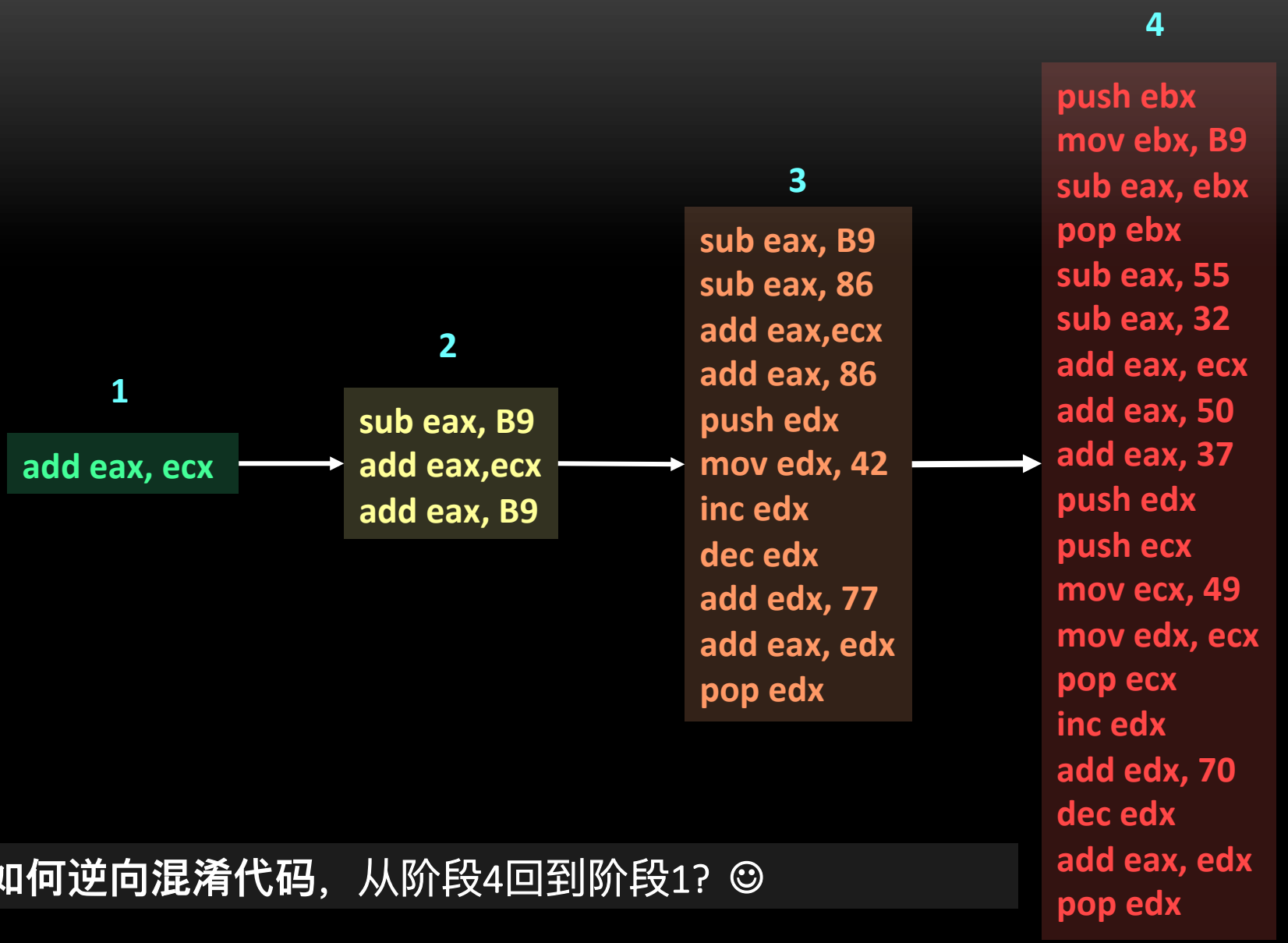
ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER


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
seg000:00000231 ; =============== S U B R O U T I N E ===
seg000:00000231
seg000:00000231
seg000:00000231 sub_231 proc near ; CODE XREF: sub_252+1F↓p
seg000:00000231
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_24B
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
Decryption instructions ☺
解密 shellcode

```
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
```

- 调用堆栈操作:
  - ✓ 你知道这里发生了什么吗? ☺

METASM



如何逆向混淆代码，从阶段4回到阶段1？☺

✓ METASM 作为反汇编器，汇编器，调试器，编译器和链接器。

✓ 主要特点:

- ✓ Ruby语言编写
- ✓ C编译器和反编译器
- ✓ 自动回溯
- ✓ 实时流程操作
- ✓ 支持以下架构:
  - ✓ Intel IA32 (16/32/64 bits)
  - ✓ PPC
  - ✓ MIPS

✓ 支持以下文件格式:

- ✓ MZ and PE/COFF
- ✓ ELF
- ✓ Mach-O
- ✓ Raw (shellcode)

- ✓ 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

✓将以下行包含到.bashrc文件中，以指示Metasm目录安装:

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

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

require "metasm"
include Metasm

mycode = Metasm::Shellcode.assemble(Metasm::Ia32.new, <<EOB)

entry:
        push ebx
        mov ebx, 0xb9
        sub eax, ebx
        pop ebx
        sub eax, 0x55
        sub eax, 0x32
        add eax, ecx
        add eax, 0x50
        add eax, 0x37
        push edx
        push ecx
        mov ecx, 0x49
        mov edx, ecx
        pop ecx
        inc edx
        add edx, 0x70
        dec edx
        add eax, edx
        pop edx
        jmp eax

EOB
```

- based on metasm.rb file and Bruce Dang code.

插入这条指令是为了使eax寄存器的计算更容易.☺

```
addrstart = 0
asmcode = mycode.init_disassembler
asmcode.disassemble(addrstart)
defcon_di = asmcode.di_at(addrstart)
defcon = defcon_di.block
puts "\n<!!!> DEF CON China 1.0:\n "
puts defcon.list


defcon.list.each{|aborges|
        puts "\n<!!!> #{aborges.instruction}"
        back = aborges.backtrace_binding()
        v = back.values
        k = back.keys
        j = k.zip(v)
        puts "DEF CON China data flow follows below:\n"
        j.each do |mykeys, myvalues|

            puts " Processing: #{mykeys} ==> #{myvalues}"

            if aborges.opcode.props[:setip]
              puts "\nDEF CON China control flow follows below:\n"
              puts " >>> #{asmcode.get_xrefs_x(aborges)}"
            end
        end
}

addrstart2 = 0
asmcode2 = mycode.init_disassembler
asmcode2.disassemble(addrstart2)
```

从start开始初始化和反汇编代码。

列出汇编代码。

初始化回溯引擎.

确定从那里返回的最后一条指令是什么。☺

```
dd = asmcode2.block_at(addrstart2)
final = asmcode2.get_xrefs_x(dd.list.last).first
puts "\n[+] final output: #{final}"

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

        when :di
                record, new, old, instruction = record
                puts "[new update] instruction #{instruction},\n --> updating expression
 once again from #{old} to #{new}\n"

        end
}

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

从上一条指令回溯。

记录回溯指令的序列。

只显示有效的指令，这真的可以改变最终的结果。

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

<!!!> DEF CON China 1.0:

0 push ebx
1 mov ebx, 0b9h
6 sub eax, ebx
8 pop ebx
9 sub eax, 55h
0ch sub eax, 32h
0fh add eax, ecx
11h add eax, 50h
14h add eax, 37h
17h push edx
18h push ecx
19h mov ecx, 49h
1eh mov edx, ecx
20h pop ecx
21h inc edx
22h add edx, 70h
25h dec edx
26h add eax, edx
28h pop edx
29h jmp eax

<!!!> push ebx
DEF CON China data flow follows below:
 Processing: esp ==> esp-4
 Processing: dword ptr [esp] ==> ebx

<!!!> mov ebx, 0b9h
DEF CON China data flow follows below:
 Processing: ebx ==> 0b9h
```

记住:这是混淆代码。☺

```
<!!!> sub eax, ebx
DEF CON China data flow follows below:
 Processing: eax ==> eax-ebx
 Processing: eflag_z ==> (((eax&0ffffffffh)-(ebx&0ffffffffh))&0ffffffffh)==0
 Processing: eflag_s ==> ((((eax&0ffffffffh)-(ebx&0ffffffffh))&0ffffffffh)>>1fh)!=0
 Processing: eflag_c ==> (eax&0ffffffffh)<(ebx&0ffffffffh)
 Processing: eflag_o ==> ((((eax&0ffffffffh)>>1fh)!=0)==(!(((ebx&0ffffffffh)>>1fh)!=0)))
&&((((eax&0ffffffffh)>>1fh)!=0)!=(((((eax&0ffffffffh)-(ebx&0ffffffffh))&0ffffffffh)>>1fh
)!=0))

<!!!> pop ebx
DEF CON China data flow follows below:
 Processing: esp ==> esp+4
 Processing: ebx ==> dword ptr [esp]

<!!!> sub eax, 55h
DEF CON China data flow follows below:
 Processing: eax ==> eax-55h
 Processing: eflag_z ==> (((eax&0ffffffffh)-((55h)&0ffffffffh))&0ffffffffh)==0
 Processing: eflag_s ==> ((((eax&0ffffffffh)-((55h)&0ffffffffh))&0ffffffffh)>>1fh)!=0
 Processing: eflag_c ==> (eax&0ffffffffh)<((55h)&0ffffffffh)
 Processing: eflag_o ==> ((((eax&0ffffffffh)>>1fh)!=0)==(!(((55h)&0ffffffffh)>>1fh)!=0)
))&&((((eax&0ffffffffh)>>1fh)!=0)!=(((((eax&0ffffffffh)-((55h)&0ffffffffh))&0ffffffffh)>
>1fh)!=0))

<!!!> sub eax, 32h
DEF CON China data flow follows below:
 Processing: eax ==> eax-32h
 Processing: eflag_z ==> (((eax&0ffffffffh)-((32h)&0ffffffffh))&0ffffffffh)==0
 Processing: eflag_s ==> ((((eax&0ffffffffh)-((32h)&0ffffffffh))&0ffffffffh)>>1fh)!=0
 Processing: eflag_c ==> (eax&0ffffffffh)<((32h)&0ffffffffh)
 Processing: eflag_o ==> ((((eax&0ffffffffh)>>1fh)!=0)==(!(((32h)&0ffffffffh)>>1fh)!=0)
))&&((((eax&0ffffffffh)>>1fh)!=0)!=(((((eax&0ffffffffh)-((32h)&0ffffffffh))&0ffffffffh)>
>1fh)!=0))
```

```
[+] final output: eax
[start] Here is the sequence of expression evaluations  eax from 0x29
[new update] instruction 26h add eax, edx,
 --> updating expression once again from eax to eax+edx
[new update] instruction 25h dec edx,
 --> updating expression once again from eax+edx to eax+edx-1
[new update] instruction 22h add edx, 70h,
 --> updating expression once again from eax+edx-1 to eax+edx+6fh
[new update] instruction 21h inc edx,
 --> updating expression once again from eax+edx+6fh to eax+edx+70h
[new update] instruction 1eh mov edx, ecx,
 --> updating expression once again from eax+edx+70h to eax+ecx+70h
[new update] instruction 19h mov ecx, 49h,
 --> updating expression once again from eax+ecx+70h to eax+0b9h
[new update] instruction 14h add eax, 37h,
 --> updating expression once again from eax+0b9h to eax+0f0h
[new update] instruction 11h add eax, 50h,
 --> updating expression once again from eax+0f0h to eax+140h
[new update] instruction 0fh add eax, ecx,
 --> updating expression once again from eax+140h to eax+ecx+140h
[new update] instruction 0ch sub eax, 32h,
 --> updating expression once again from eax+ecx+140h to eax+ecx+10eh
[new update] instruction 9 sub eax, 55h,
 --> updating expression once again from eax+ecx+10eh to eax+ecx+0b9h
[new update] instruction 6 sub eax, ebx,
 --> updating expression once again from eax+ecx+0b9h to eax-ebx+ecx+0b9h
[new update] instruction 1 mov ebx, 0b9h,
 --> updating expression once again from eax-ebx+ecx+0b9h to eax+ecx
```

Game over. ☺

The effective instructions are:

```
1 mov ebx, 0b9h
6 sub eax, ebx
9 sub eax, 55h
0ch sub eax, 32h
0fh add eax, ecx
11h add eax, 50h
14h add eax, 37h
19h mov ecx, 49h
1eh mov edx, ecx
21h inc edx
22h add edx, 70h
25h dec edx
26h add eax, edx
```

输出来自 backtracing_log.select 命令 (在逆向中)

- ✓ 仿真始终是解决实际逆向工程问题的一种优秀方法，幸运的是，我们有uEmu，还可以使用Keystone Engine 汇编器和Capstone Engine 反汇编器。☺

- ✓ Keystone Engine的作用是汇编和:

  - ✓ 支持x86、Mips、Arm等多种架构。
  - ✓ 它是用C/ C++实现的，并绑定到Python、Ruby、Powershell和C#(以及其他语言)。

- ✓ 安装Keystone：

  - ✓ 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
  - ✓ root@kali:~/programs/keystone-0.9.1# mkdir build ; 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

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

#define DEFCON "push ebx; mov ebx, 0xb9; sub eax, ebx; pop ebx; sub eax, 0x55; sub eax,
 0x32; add eax, ecx; add eax, 0x50; add eax, 0x37; push edx; push ecx; mov ecx, 0x49; m
ov edx, ecx; pop ecx; inc edx; add edx, 0x70; 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;
    }

    if (ks_asm(keyeng, DEFCON, 0, &encode, &size, &count)) {
        printf("ERROR: A fail has occured while calling ks_asm() with count = %lu, erro
r code = %u\n", count, ks_errno(keyeng));
    } else {
        size_t i;

        for (i = 0; i < size; i++) {
            printf("%02x ", encode[i]);
        }

    }

    ks_free(encode);
    ks_close(keyeng);

    return 0;
}
```

来自原始模糊代码的指令

创建一个 keystone 引擎

使用keystone引擎汇编我们的指令。

释放内存，关闭引擎.

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

KEYSTONE_LDFLAGS = -lkeystone -lstdc++ -lm

all:
        ${CC} -o defcon2019 defcon2019.c ${KEYSTONE_LDFLAGS}

clean:
        rm -rf *.o defcon2019
root@kali:~/programs/defcon#
root@kali:~/programs/defcon# make
cc -o defcon2019 defcon2019.c -lkeystone -lstdc++ -lm
root@kali:~/programs/defcon#
root@kali:~/programs/defcon# ./defcon2019
53 bb b9 00 00 00 29 d8 5b 83 e8 55 83 e8 32 01 c8 83 c0 50 83 c0 37 52 51 b9 49
00 00 00 89 ca 59 42 83 c2 70 4a 01 d0 5a root@kali:~/programs/defcon#
root@kali:~/programs/defcon#
root@kali:~/programs/defcon# ./defcon2019 | xxd -r -p - > defcon2019.bin
root@kali:~/programs/defcon#
root@kali:~/programs/defcon# hexdump -C defcon2019.bin
00000000  53 bb b9 00 00 00 29 d8  5b 83 e8 55 83 e8 32 01  |S.....).[..U..2.|
00000010  c8 83 c0 50 83 c0 37 52  51 b9 49 00 00 00 89 ca  |...P..7RQ.I.....|
00000020  59 42 83 c2 70 4a 01 d0  5a                       |YB..pJ..Z|
00000029
root@kali:~/programs/defcon# _
```

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

#define CODE "\x53\xbb\xb9\x00\x00\x00\x29\xd8\x5b\x83\xe8\x55\x83\xe8\x32\x01\xc
8\x83\xc0\x50\x83\xc0\x37\x52\x51\xb9\x49\x00\x00\x00\x89\xca\x59\x42\x83\xc2\x70
\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, &instructio
n);
        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 disassembling!\n
");

        cs_close(&cs_handle);

    return 0;
}
```

安装 Capstone: apt-get install libcapstone3 libcapstone-dev ☺

ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER

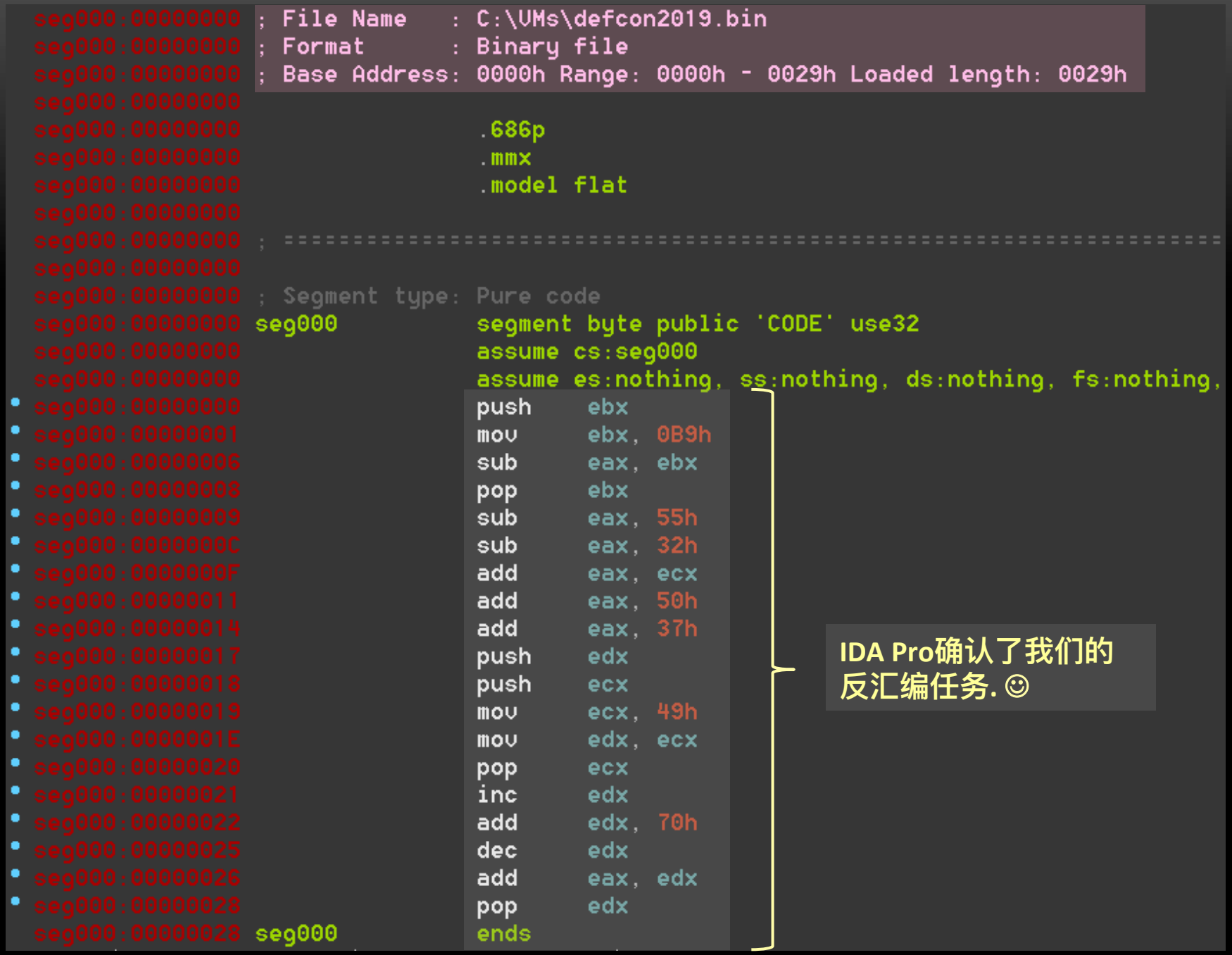
```
root@kali:~/programs/defcon/capstone# more Makefile
.PHONY: all clean

CAPSTONE_LDFLAGS = -lcapstone -lstdc++ -lm

all:
        ${CC} -o defcon2019_rev defcon2019_rev.c ${CAPSTONE_LDFLAGS}

clean:
        rm -rf *.o defcon2019_rev
root@kali:~/programs/defcon/capstone#
root@kali:~/programs/defcon/capstone# make
cc -o defcon2019_rev defcon2019_rev.c -lcapstone -lstdc++ -lm
root@kali:~/programs/defcon/capstone#
root@kali:~/programs/defcon/capstone# ./defcon2019_rev
0x1:    push            ebx
0x2:    mov             ebx, 0xb9
0x7:    sub             eax, ebx
0x9:    pop             ebx
0xa:    sub             eax, 0x55
0xd:    sub             eax, 0x32
0x10:   add             eax, ecx
0x12:   add             eax, 0x50
0x15:   add             eax, 0x37
0x18:   push            edx
0x19:   push            ecx
0x1a:   mov             ecx, 0x49
0x1f:   mov             edx, ecx
0x21:   pop             ecx
0x22:   inc             edx
0x23:   add             edx, 0x70
0x26:   dec             edx
0x27:   add             eax, edx
0x29:   pop             edx
root@kali:~/programs/defcon/capstone# _
```

OCapstone反汇编的原始代码. ☺


seg000:00000000 ; File Name : C:\VMs\defcon2019.bin
seg000:00000000 ; Format : Binary file
seg000:00000000 ; Base Address: 0000h Range: 0000h - 0029h Loaded length: 0029h
seg000:00000000 .686p
seg000:00000000 .mmx
seg000:00000000 .model flat
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,
seg000:00000000 push ebx
seg000:00000001 mov ebx, 0B9h
seg000:00000006 sub eax, ebx
seg000:00000008 pop ebx
seg000:00000009 sub eax, 55h
seg000:0000000C sub eax, 32h
seg000:0000000F add eax, ecx
seg000:00000011 add eax, 50h
seg000:00000014 add eax, 37h
seg000:00000017 push edx
seg000:00000018 push ecx
seg000:00000019 mov ecx, 49h
seg000:0000001E mov edx, ecx
seg000:00000020 pop ecx
seg000:00000021 inc edx
seg000:00000022 add edx, 70h
seg000:00000025 dec edx
seg000:00000026 add eax, edx
seg000:00000028 pop edx
seg000:00000028 seg000 ends
IDA Pro确认了我们的反汇编任务.☺

- ✓ # git clone https://github.com/unicorn-engine/unicorn.git
- ✓ # cd unicorn ; ./make.sh
- ✓ # ./make.sh install

```
1  #include <unicorn/unicorn.h>
2  #include <string.h>
3
4  // Our code to be emulated.
5
6  #define DEFCON_CODE "\x53\xbb\xb9\x00\x00\x00\x29\xd8\x5b\x83\xe8\x55\x83\xe8\x32
   \x01\xc8\x83\xc0\x50\x83\xc0\x37\x52\x51\xb9\x49\x00\x00\x00\x89\xca\x59\x42\x83\
   xc2\x70\x4a\x01\xd0\x5a"
7
8  // Emulation start address and a simple macro.
9
10 #define ADDR 0x1000000
11 #define MIN(x, y) (x < y? x : y)
12
13 // Hook the instruction execution.
14
15 static void hook_code(uc_engine *uc, uint64_t address, uint32_t size, void *user_
   data)
16 {
17         int r_eip;
18         int r_eax;
19         int r_ebx;
20         int r_ecx;
21         int r_edx;
22
23         uint8_t instr_size[16];
24
```

```
25          printf("\nTracing instruction at 0x%x , instruction size = 0x%x\n", addre
    ss, size);
26
27          uc_reg_read(uc, UC_X86_REG_EIP, &r_eip);
28          uc_reg_read(uc, UC_X86_REG_EAX, &r_eax);
29          uc_reg_read(uc, UC_X86_REG_EBX, &r_ebx);
30          uc_reg_read(uc, UC_X86_REG_ECX, &r_ecx);
31          uc_reg_read(uc, UC_X86_REG_EDX, &r_edx);
32
33  // Print the initial values of registries.
34
35          printf("\n>> EIP=0x%x ", r_eip);
36          printf(" | EAX=0x%x ", r_eax);
37          printf(" | EBX=0x%x ", r_ebx);
38          printf(" | ECX=0x%x ", r_ecx);
39          printf(" | EDX=0x%x ", r_edx);
40          printf("\n>> Executed hex code: ");
41
42          size = MIN(sizeof(instr_size), size);
43          if (!uc_mem_read(uc, address, instr_size, size)) {
44                  uint32_t i;
45                  for (i=0; i<size; i++) {
46                          printf("%x ", instr_size[i]);
47                  }
48                  printf("\n");
49          }
50  }
51
52  int main(int argc, char **argv, char **envp)
53  {
54
```

```
55 // Declare and initialize few variables
56
57         uc_engine *uc;
58         uc_hook traceinstr;
59         uc_err err;
60
61 // Set up the initial registry values.
62 // We have to set up the ESP register for emulating PUSH/POP instructions.
63
64         int r_eax = 0x4;
65         int r_ebx = 0x0;
66         int r_ecx = 0x7;
67         int r_edx = 0x0;
68         int r_esp = ADDR + 200000;
69
70         printf("\nInitial register values: \n");
71
72         printf("\n>> EAX = %x ", r_eax);
73         printf("\n>> EBX = %x ", r_ebx);
74         printf("\n>> ECX = %x ", r_ecx);
75         printf("\n>> EDX = %x ", r_edx);
76
77         printf("\n\nOur emulated code is: \n");
78
79
80 // We are emulating a 32-bit application in x86 emulator, so initialize the emula
   tor in X86-32bit mode :)
81 // If we wished to emulate in a x64 emulator, so we would use UC_MODE_64.
82
83         err = uc_open(UC_ARCH_X86, UC_MODE_32, &uc);
84         if (err != UC_ERR_OK) {
85                 printf("A fail to use uc_open() has occured and the error returne
   d is: %u\n", err);
86                 return -1;
87         }
```


ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER

```
88
89 // We are reserving 4MB memory for this emulation. Additionally, UC_PROT_ALL mean
   s: RWX.
90
91         uc_mem_map(uc, ADDR, 4 * 1024 * 1024, UC_PROT_ALL);
92
93 // write machine code to be emulated to memory
94
95         if (uc_mem_write(uc, ADDR, DEFCON_CODE, sizeof(DEFCON_CODE) - 1)) {
96                 printf("It has happened a fail during the write emulation code to
    memory!\n");
97                 return -1;
98 }
99
100 // We need to initialize the machine registers
101
102         uc_reg_write(uc, UC_X86_REG_EAX, &r_eax);
103         uc_reg_write(uc, UC_X86_REG_EBX, &r_ebx);
104         uc_reg_write(uc, UC_X86_REG_ECX, &r_ecx);
105         uc_reg_write(uc, UC_X86_REG_EDX, &r_edx);
106         uc_reg_write(uc, UC_X86_REG_ESP, &r_esp);
107
108 // uc: hook handle ; traceinstr: reference to uc_hook ; UC_HOOK_CODE: hook type ;
    hook_code: callback function
109
110         uc_hook_add(uc, &traceinstr, UC_HOOK_CODE, hook_code, NULL, 1, 0);
111
112
```

```
112
113 // Start the emulation engine and emulate code in infinite time (first zero
     below) & unlimited instructions (second zero below).
114
115         err=uc_emu_start(uc, ADDR, ADDR + sizeof(DEFCON_CODE) - 1, 0, 0);
116         if (err) {
117
118                 printf("The uc_emu_start() function has failed with error r
    eturning %u: %s\n", err, uc_strerror(err));
119
120 }
121
122 // Finally, print out the final registers values.
123
124         printf("\nThe final CPU registers contain the following content: \n
    \n");
125
126         uc_reg_read(uc, UC_X86_REG_EAX, &r_eax);
127         uc_reg_read(uc, UC_X86_REG_EBX, &r_ebx);
128         uc_reg_read(uc, UC_X86_REG_ECX, &r_ecx);
129         uc_reg_read(uc, UC_X86_REG_EDX, &r_edx);
130         printf(">>> EAX = 0x%x", r_eax);
131         printf("\n>>> EBX = 0x%x", r_ebx);
132         printf("\n>>> ECX = 0x%x", r_ecx);
133         printf("\n>>> EDX = 0x%x\n\n", r_edx);
134
135         uc_close(uc);
136
137         return 0;
138 }
```

```
root@kali:~/programs/defcon/unicorn# ./unicorn_defcon

Initial register values:

>> EAX = 4
>> EBX = 0
>> ECX = 7
>> EDX = 0

Our emulated code is:

Tracing instruction at 0x1000000 , instruction size = 0x1

>> EIP=0x1000000  | EAX=0x4  | EBX=0x0  | ECX=0x7  | EDX=0x0
>> Executed hex code: 53

Tracing instruction at 0x1000001 , instruction size = 0x5

>> EIP=0x1000001  | EAX=0x4  | EBX=0x0  | ECX=0x7  | EDX=0x0
>> Executed hex code: bb b9 0 0 0

Tracing instruction at 0x1000006 , instruction size = 0x2

>> EIP=0x1000006  | EAX=0x4  | EBX=0xb9  | ECX=0x7  | EDX=0x0
>> Executed hex code: 29 d8

Tracing instruction at 0x1000008 , instruction size = 0x1

>> EIP=0x1000008  | EAX=0xffffff4b  | EBX=0xb9  | ECX=0x7  | EDX=0x0
>> Executed hex code: 5b
```

```
Tracing instruction at 0x1000021 , instruction size = 0x1

>> EIP=0x1000021  | EAX=0xffffff52  | EBX=0x0  | ECX=0x7  | EDX=0x49
>> Executed hex code: 42

Tracing instruction at 0x1000022 , instruction size = 0x3

>> EIP=0x1000022  | EAX=0xffffff52  | EBX=0x0  | ECX=0x7  | EDX=0x4a
>> Executed hex code: 83 c2 70

Tracing instruction at 0x1000025 , instruction size = 0x1

>> EIP=0x1000025  | EAX=0xffffff52  | EBX=0x0  | ECX=0x7  | EDX=0xba
>> Executed hex code: 4a

Tracing instruction at 0x1000026 , instruction size = 0x2

>> EIP=0x1000026  | EAX=0xffffff52  | EBX=0x0  | ECX=0x7  | EDX=0xb9
>> Executed hex code: 1 d0

Tracing instruction at 0x1000028 , instruction size = 0x1

>> EIP=0x1000028  | EAX=0xb  | EBX=0x0  | ECX=0x7  | EDX=0xb9
>> Executed hex code: 5a

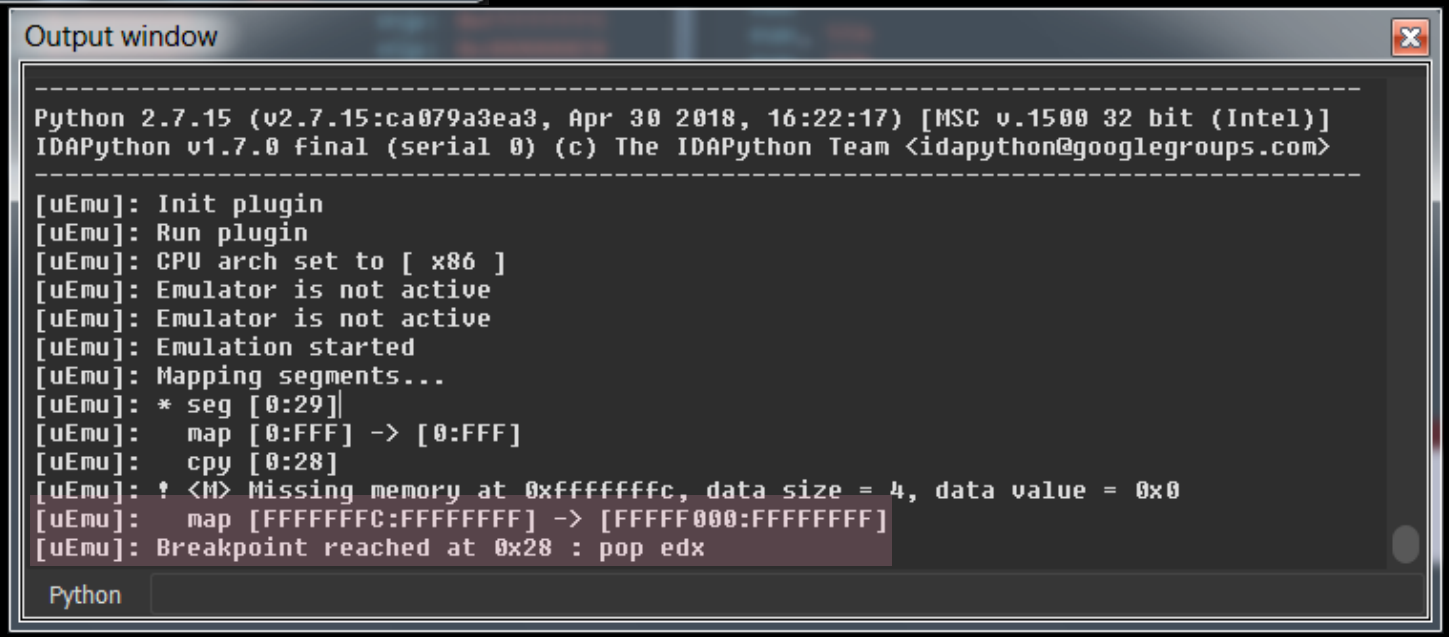
The final CPU registers contain the following content:

>>> EAX = 0xb
>>> EBX = 0x0
>>> ECX = 0x7
>>> EDX = 0x0
```

- ✓ 下载uEmu 从 https://github.com/alexhude/uEmu
- ✓ 安装Unicorn: pip install unicorn.
- ✓ 在IDA中加载 uEmu 使用 ALT+F7 热键.
- ✓ 右键单击 代码并选择uEmu子菜单。







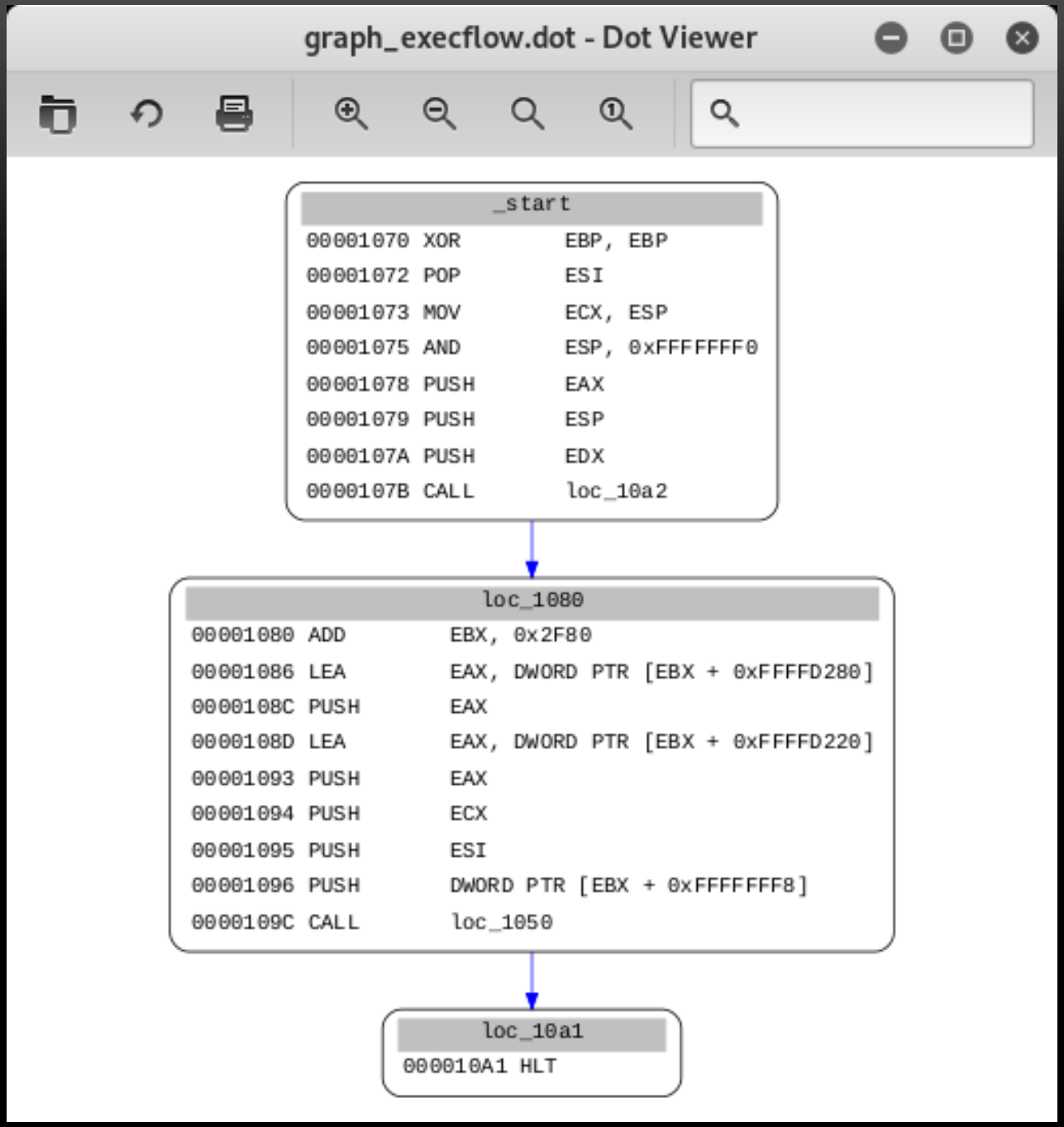
MIASM

- ✓ MIASM是逆向工程中最令人印象深刻的框架之一，它能够**分析、生成和修改**几种不同类型的程序。
- ✓ MIASM**支持从**ARM、x86、MIPS**等**不同平台汇编和反汇编程序，还可以使用JIT进行仿真。
- ✓ **因此**, MIASM **是最出色的对抗混淆的工具**.
- ✓ **安装** 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
  - ✓ ./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
- ✓ (testing MIASM) root@kali:~/programs# python /root/programs/miasm/example/disasm/full.py -m x86_32 /root/programs/shellcode

```
INFO : Load binary
INFO : ok
INFO : import machine...
INFO : ok
INFO : func ok 0000000000001070 (0)
INFO : generate graph file
INFO : generate intervals
[0x1070 0x10A2]
INFO : total lines 0
```

- ✓ (testing MIASM) xdot graph_execflow.dot


graph_execflow.dot - Dot Viewer
_start
00001070 XOR EBP, EBP
00001072 POP ESI
00001073 MOV ECX, ESP
00001075 AND ESP, 0xFFFFFFF0
00001078 PUSH EAX
00001079 PUSH ESP
0000107A PUSH EDX
0000107B CALL loc_10a2
loc_1080
00001080 ADD EBX, 0x2F80
00001086 LEA EAX, DWORD PTR [EBX + 0xFFFFD280]
0000108C PUSH EAX
0000108D LEA EAX, DWORD PTR [EBX + 0xFFFFD220]
00001093 PUSH EAX
00001094 PUSH ECX
00001095 PUSH ESI
00001096 PUSH DWORD PTR [EBX + 0xFFFFFFF8]
0000109C CALL loc_1050
loc_10a1
000010A1 HLT

```
1  from miasm2.analysis.binary import Container
2  from miasm2.analysis.machine import Machine
3  from miasm2.jitter.csts import PAGE_READ, PAGE_WRITE
4
5  with open("defcon2019.bin") as fdesc:
6      cont=Container.from_stream(fdesc)
7
8  machine=Machine('x86_32')
9  mdis=machine.dis_engine(cont.bin_stream)
10 ourblocks = mdis.dis_multiblock(0)
11 for block in ourblocks:
12     print block
13 jitter = machine.jitter("llvm")
14 jitter.init_stack()
15 s = open("defcon2019.bin").read()
16 run_addr = 0x40000000
17 jitter.cpu.EAX=3
18 jitter.cpu.ECX=6
19 jitter.vm.add_memory_page(run_addr, PAGE_READ | PAGE_WRITE, s)
20 def code_sentinelle(jitter):
21     jitter.run = False
22     jitter.pc = 0
23     return True
24 jitter.add_breakpoint(0x40000028, code_sentinelle)
25 jitter.push_uint32_t(0x40000028)
26 jitter.jit.log_regs = True
27 jitter.jit.log_mn = True
28 jitter.init_run(run_addr)
29 jitter.continue_run()
30
31 open('defcon2019_cfg.dot', 'w').write(ourblocks.dot())
32
```

打开我们的文件。容器向反汇编引擎提供字节源。

使用x86 32位体系结构实例化组装引擎。

从开始运行递归的横向反汇编。

将“llvm”设置为Jit引擎进行仿真并初始化堆栈。

设置虚拟启动地址、寄存器值和内存保护。

在最后一行代码中添加断点。

运行仿真

生成点图。

```
root@kali:~/programs/defcon# python miasm.py
WARNING: not enough bytes in str
WARNING: cannot disasm at 29
WARNING: not enough bytes in str
WARNING: cannot disasm at 29
loc_000000000000000:0x00000000
PUSH        EBX
MOV         EBX, 0xB9
SUB         EAX, EBX
POP         EBX
SUB         EAX, 0x55
SUB         EAX, 0x32
ADD         EAX, ECX
ADD         EAX, 0x50
ADD         EAX, 0x37
PUSH        EDX
PUSH        ECX
MOV         ECX, 0x49
MOV         EDX, ECX
POP         ECX
INC         EDX
ADD         EDX, 0x70
DEC         EDX
ADD         EAX, EDX
POP         EDX
->      c_next:loc_000000000000029:0x00000029
loc_000000000000029:0x00000029
```

反汇编我们的代码(再一次) ☺

```
40000000 PUSH        EBX
EAX 00000003 EBX 00000000 ECX 00000006 EDX 00000000 ESI 00000000 EDI 00000000
ESP 0123FFF8 EBP 00000000 EIP 40000000 zf 0 nf 0 of 0 cf 0
40000001 MOV         EBX, 0xB9
EAX 00000003 EBX 000000B9 ECX 00000006 EDX 00000000 ESI 00000000 EDI 00000000
ESP 0123FFF8 EBP 00000000 EIP 40000000 zf 0 nf 0 of 0 cf 0
40000006 SUB         EAX, EBX
EAX FFFFFF4A EBX 000000B9 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 FFFFFF4A EBX 00000000 ECX 00000006 EDX 00000000 ESI 00000000 EDI 00000000
ESP 0123FFFC EBP 00000000 EIP 40000000 zf 0 nf 1 of 0 cf 1
40000009 SUB         EAX, 0x55
EAX FFFFFEF5 EBX 00000000 ECX 00000006 EDX 00000000 ESI 00000000 EDI 00000000
ESP 0123FFFC EBP 00000000 EIP 40000000 zf 0 nf 1 of 0 cf 0
4000000C SUB         EAX, 0x32
EAX FFFFFEC3 EBX 00000000 ECX 00000006 EDX 00000000 ESI 00000000 EDI 00000000
ESP 0123FFFC EBP 00000000 EIP 40000000 zf 0 nf 1 of 0 cf 0



EAX FFFFFF50 EBX 00000000 ECX 00000006 EDX 00000049 ESI 00000000 EDI 00000000
ESP 0123FFF8 EBP 00000000 EIP 40000000 zf 0 nf 1 of 0 cf 0
40000021 INC         EDX
EAX FFFFFF50 EBX 00000000 ECX 00000006 EDX 0000004A ESI 00000000 EDI 00000000
ESP 0123FFF8 EBP 00000000 EIP 40000000 zf 0 nf 0 of 0 cf 0
40000022 ADD         EDX, 0x70
EAX FFFFFF50 EBX 00000000 ECX 00000006 EDX 000000BA ESI 00000000 EDI 00000000
ESP 0123FFF8 EBP 00000000 EIP 40000000 zf 0 nf 0 of 0 cf 0
40000025 DEC         EDX
EAX FFFFFF50 EBX 00000000 ECX 00000006 EDX 000000B9 ESI 00000000 EDI 00000000
ESP 0123FFF8 EBP 00000000 EIP 40000000 zf 0 nf 0 of 0 cf 0
40000026 ADD         EAX, EDX
EAX 00000009 EBX 00000000 ECX 00000006 EDX 000000B9 ESI 00000000 EDI 00000000
ESP 0123FFF8 EBP 00000000 EIP 40000000 zf 0 nf 0 of 0 cf 1
```


loc_0000000000000000
PUSH EBX
MOV EBX, 0xB9
SUB EAX, EBX
POP EBX
SUB EAX, 0x55
SUB EAX, 0x32
ADD EAX, ECX
ADD EAX, 0x50
ADD EAX, 0x37
PUSH EDX
PUSH ECX
MOV ECX, 0x49
MOV EDX, ECX
POP ECX
INC EDX
ADD EDX, 0x70
DEC EDX
ADD EAX, EDX
POP EDX
我们的目标代码. ☺
loc_0000000000000029
IOError

```
root@kali:~/programs/defcon# python
Python 2.7.16 (default, Apr  6 2019, 01:42:57)
[GCC 8.3.0] 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
>>> with open("defcon2019.bin") as fdesc:
...     cont=Container.from_stream(fdesc)
...
>>> defconmach=Machine('x86_32')
>>> defcondis=defconmach.dis_engine(cont.bin_stream)
>>> myblocks = defcondis.dis_multiblock(0)
WARNING: not enough bytes in str
WARNING: cannot disasm at 29
WARNING: not enough bytes in str
WARNING: cannot disasm at 29
>>> sym = defconmach.ira( )
>>> for block in myblocks:
...     sym.add_block(block)
...
[<miasm2.ir.ir.IRBlock object at 0x7f0fde22b870>]
[]
>>> from miasm2.ir.symbexec import SymbolicExecutionEngine
>>> symb = SymbolicExecutionEngine(sym,defconmach.mn.regs.regs_init)
>>> symbolic_pc = symb.run_at(0, step=True)
```

得到IRA转换器。

初始化并运行符号执行引擎。

```
>>> symbolic_pc = symb.run_at(0, step=True)
Instr PUSH       EBX
Assignblk:
ESP = ESP + -0x4
@32[ESP + -0x4] = EBX

________________________________________________________________________________
ESP                 = ESP_init + 0xFFFFFFFC
@32[ESP_init + 0xFFFFFFFC] = EBX_init

________________________________________________________________________________
Instr MOV        EBX, 0xB9
Assignblk:
EBX = 0xB9

________________________________________________________________________________
ESP                 = ESP_init + 0xFFFFFFFC
EBX                 = 0xB9
@32[ESP_init + 0xFFFFFFFC] = 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
```

```
EAX                 = EAX_init + ECX_init
cf                  = ((((EAX_init + ECX_init) ^ (EAX_init + ECX_init + 0xFFFFFF4
7)) & ((EAX_init + ECX_init + 0xFFFFFF47) ^ 0xFFFFFF46)) ^ (EAX_init + ECX_init)
 ^ (EAX_init + ECX_init + 0xFFFFFF47) ^ 0xB9)[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 + 0xFFFFFF47)
 ^ 0xB9)[4:5]
of                  = (((EAX_init + ECX_init) ^ (EAX_init + ECX_init + 0xFFFFFF47
)) & ((EAX_init + ECX_init + 0xFFFFFF47) ^ 0xFFFFFF46))[31:32]
nf                  = (EAX_init + ECX_init)[31:32]
@32[ESP_init + 0xFFFFFFF8] = ECX_init
@32[ESP_init + 0xFFFFFFFC] = EDX_init

Instr POP         EDX
Assignblk:
IRDst = loc_0000000000000029:0x00000029

EAX                 = EAX_init + ECX_init
cf                  = ((((EAX_init + ECX_init) ^ (EAX_init + ECX_init + 0xFFFFFF4
7)) & ((EAX_init + ECX_init + 0xFFFFFF47) ^ 0xFFFFFF46)) ^ (EAX_init + ECX_init)
 ^ (EAX_init + ECX_init + 0xFFFFFF47) ^ 0xB9)[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 + 0xFFFFFF47)
 ^ 0xB9)[4:5]
IRDst               = 0x29
of                  = (((EAX_init + ECX_init) ^ (EAX_init + ECX_init + 0xFFFFFF47
)) & ((EAX_init + ECX_init + 0xFFFFFF47) ^ 0xFFFFFF46))[31:32]
nf                  = (EAX_init + ECX_init)[31:32]
@32[ESP_init + 0xFFFFFFF8] = ECX_init
@32[ESP_init + 0xFFFFFFFC] = EDX_init
```

我们之前的测试也得出了同样的结论。☺

DTRACE on WINDOWS

- ✓ DTrace是一个动态跟踪框架，它在Solaris操作系统上非常高效并且非常有名。

- ✓ Dtrace最初由太阳微系统公司的Mike Shapiro、Adam Leventhal和Brian Cantrill编写。尽管他们从2003年开始开发DTrace，但它只在Solaris 10 03/05中引入。

- ✓ 它用于在用户和内核模式下获得系统的实时概览。此外，它还可以用来理解应用程序和系统的行为。

- ✓ 几个月前，DTrace被移植到Windows: https://github.com/opendtrace/opendtrace/tree/windows

- ✓ DTrace可以概括为一组分散在内核关键点上的探针(传感器)。因此，每次“激活”探测时，都可以注册并理解应用程序行为。

- ✓ 使用DTrace可以更容易地跟踪进程和系统的概要文件、查找哪些系统调用被“调用”、进程写入/读取了多少字节、进程打开了多少文件、跟踪被调用的系统调用序列等等。

- ✓ DTrace 的脚本使用 D语言编写 (与 awk相似).
- ✓ 探测名称由以下语法描述:

```
provider:module:function:name
```

定义如下:

- ✓ provider: 用于测量系统某一区域的探针库。在Windows上，现有的提供者有syscall、etw、profile、pid和dtrace。
- ✓ module: 内核模块，我们在这里找到探针。
- ✓ function: 函数包含探针.
- ✓ name:目标探测的特定名称或描述。

- ✓ 关键概念:

- ✓ predicates:用户定义的条件。
- ✓ actions:当探测触发时运行的任务。
- ✓ aggregations:使用聚合函数合并数据。

- ✓ 安装 DTrace:
  - ✓ Windows 10 x64 (build 18342 或更晚) 来自Windows内部程序。
  - ✓ bcdedit.exe /set dtrace on
  - ✓ 下载 DTrace package: http://download.microsoft.com/download/B/D/4/BD4B95A5-0B61-4D8F-837C-F889AAD8DAA2/DTrace.amd64.msi
  - ✓ _NT_SYMBOL_PATH=srv*C:\symbols*https://msdl.microsoft.com/download/symbols
  - ✓ 引导系统。
  - ✓ 以管理员身份打开命令提示符.
  - ✓ 如果使用fbt(函数边界跟踪)，则需要附加WinDbg并在调试模式下启动窗口。 ☺

```
C:\Users\Administrator>dtrace -l | more
   ID   PROVIDER            MODULE                          FUNCTION NAME
    1     dtrace                                                     BEGIN
    2     dtrace                                                     END
    3     dtrace                                                     ERROR
    4    syscall                                NtLockProductActivationKeys entry
    5    syscall                                NtLockProductActivationKeys return
    6    syscall                                    NtWaitHighEventPair entry
    7    syscall                                    NtWaitHighEventPair return
    8    syscall                              NtRegisterThreadTerminatePort entry
    9    syscall                              NtRegisterThreadTerminatePort return
   10    syscall                           NtAssociateWaitCompletionPacket entry
   11    syscall                           NtAssociateWaitCompletionPacket return
   12    syscall                                NtQueryPerformanceCounter entry
   13    syscall                                NtQueryPerformanceCounter return
   14    syscall                                        NtCompactKeys entry
   15    syscall                                        NtCompactKeys return
   16    syscall                              NtQuerySystemInformationEx entry
   17    syscall                              NtQuerySystemInformationEx return
   18    syscall                                        NtResetEvent entry
   19    syscall                                        NtResetEvent return
   20    syscall                                    NtGetContextThread entry
   21    syscall                                    NtGetContextThread return
   22    syscall                             NtQueryInformationThread entry
```

```
C:\>dtrace -V
dtrace: Sun D 1.13

C:\>dtrace -l | grep -v "syscall" | grep -v "etw"
   ID    PROVIDER            MODULE                          FUNCTION NAME
    1     dtrace                                                       BEGIN
    2     dtrace                                                       END
    3     dtrace                                                       ERROR
 2997    profile                                                       profile-97
 2998    profile                                                       profile-199
 2999    profile                                                       profile-499
 3000    profile                                                       profile-997
 3001    profile                                                       profile-1999
 3002    profile                                                       profile-4001
 3003    profile                                                       profile-4999
 3004    profile                                                       tick-1
 3005    profile                                                       tick-10
 3006    profile                                                       tick-100
 3007    profile                                                       tick-500
 3008    profile                                                       tick-1000
 3009    profile                                                       tick-5000
 3044    profile                                                       tick-5sec
```

```
C:\>dtrace -ln "syscall::*Read*:entry"
   ID   PROVIDER            MODULE                          FUNCTION NAME
   30    syscall                                    NtReadOnlyEnlistment entry
  140    syscall                                     NtReadRequestData entry
  170    syscall                                NtWorkerFactoryWorkerReady entry
  234    syscall                                     NtReadFileScatter entry
  608    syscall                                   NtReadVirtualMemory entry
  614    syscall                                              NtReadFile entry

C:\>dtrace -ln "syscall::*Write*:entry"
   ID   PROVIDER            MODULE                          FUNCTION NAME
   40    syscall                                             NtWriteFile entry
  116    syscall                                          NtGetWriteWatch entry
  224    syscall                                NtFlushProcessWriteBuffers entry
  332    syscall                                   NtWriteVirtualMemory entry
  356    syscall                                        NtFlushWriteBuffer entry
  370    syscall                                       NtWriteRequestData entry
  532    syscall                                       NtWriteFileGather entry
  632    syscall                                       NtResetWriteWatch entry

C:\>dtrace -ln "syscall::*View*:entry"
   ID   PROVIDER            MODULE                          FUNCTION NAME
  516    syscall                                    NtUnmapViewOfSectionEx entry
  518    syscall                                         NtMapViewOfSection entry
  638    syscall                                  NtAlpcCreateSectionView entry
  704    syscall                                  NtAlpcDeleteSectionView entry
  878    syscall                                       NtUnmapViewOfSection entry
  918    syscall                                    NtMapViewOfSectionEx entry
```

```
C:\>dtrace -Fn "syscall:::entry /execname==\"notepad.exe\"/ { @num[probefunc] = count(); }"
dtrace: description 'syscall:::entry ' matched 464 probes


  NtCreateFile                                                  1
  NtQueryAttributesFile                                         1
  NtQueryInformationFile                                        1
  NtQueryValueKey                                               1
  NtWriteFile                                                   1
  NtEnumerateKey                                                2
  NtQueryInformationToken                                       2
  NtSetInformationFile                                          2
  NtSetInformationProcess                                       2
  NtSetTimer2                                                   2
  NtWaitForWorkViaWorkerFactory                                 2
  NtTraceEvent                                                  4
  NtClearEvent                                                  6
  NtOpenKeyEx                                                   6
  NtOpenEvent                                                   7
  NtQueryKey                                                   10
  NtAssociateWaitCompletionPacket                              12
  NtSetInformationThread                                       16
  NtAlpcSendWaitReceivePort                                    30
  NtOpenFile                                                  135
  NtQueryDirectoryFileEx                                      135
  NtClose                                                     138
  NtQueryInformationProcess                                   138
  NtCallbackReturn                                            616
```

```
C:\>dtrace -n "syscall:::entry { @num[pid,execname] = count(); }"
dtrace: description 'syscall:::entry ' matched 464 probes


     5492  RuntimeBroker.                                  1
        0  DismHost.exe                                    2
        0  VSSVC.exe                                       2
        0  svchost.exe                                     2
     8376  smartscreen.ex                                  3
     1248  TrustedInstall                                  6
     1544  svchost.exe                                     6
     9260  wimserv.exe                                     6
     3584  vmtoolsd.exe                                    7
     8000  vmtoolsd.exe                                   11
     7560  cmd.exe                                        14
     1380  svchost.exe                                    15
     1568  RuntimeBroker.                                 20
     4144  svchost.exe                                    20
     3564  vmms.exe                                       24
     9408  WinRAR.exe                                     27
     4528  vmcompute.exe                                  30
      480  svchost.exe                                    46
     1988  svchost.exe                                    89
     3184  svchost.exe                                    98
     1152  ctfmon.exe                                    108
     4844  wuauclt.exe                                   126
```

```
C:\>dtrace -Fn "tick-5sec { exit(0);} syscall:::entry { @num[probefunc] = count(); }" | tail -20
dtrace: description 'tick-5sec ' matched 465 probes
  NtCreateFile                                                  771
  NtReleaseMutant                                               860
  NtQueryVirtualMemory                                          878
  NtSetInformationKey                                          1094
  NtSetInformationFile                                         1152
  NtEnumerateKey                                               1215
  NtOpenThreadToken                                            1286
  NtCreateKey                                                  1295
  NtEnumerateValueKey                                          1312
  NtQueryInformationFile                                       1953
  NtWriteFile                                                  2476
  NtQuerySecurityObject                                        2669
  NtQueryValueKey                                              3089
  NtWaitForSingleObject                                        3380
  NtQueryDirectoryFileEx                                       4225
  NtOpenFile                                                   4237
  NtQueryInformationToken                                      6111
  NtOpenKeyEx                                                  7470
  NtClose                                                     14041
  NtQueryKey                                                  15949
```

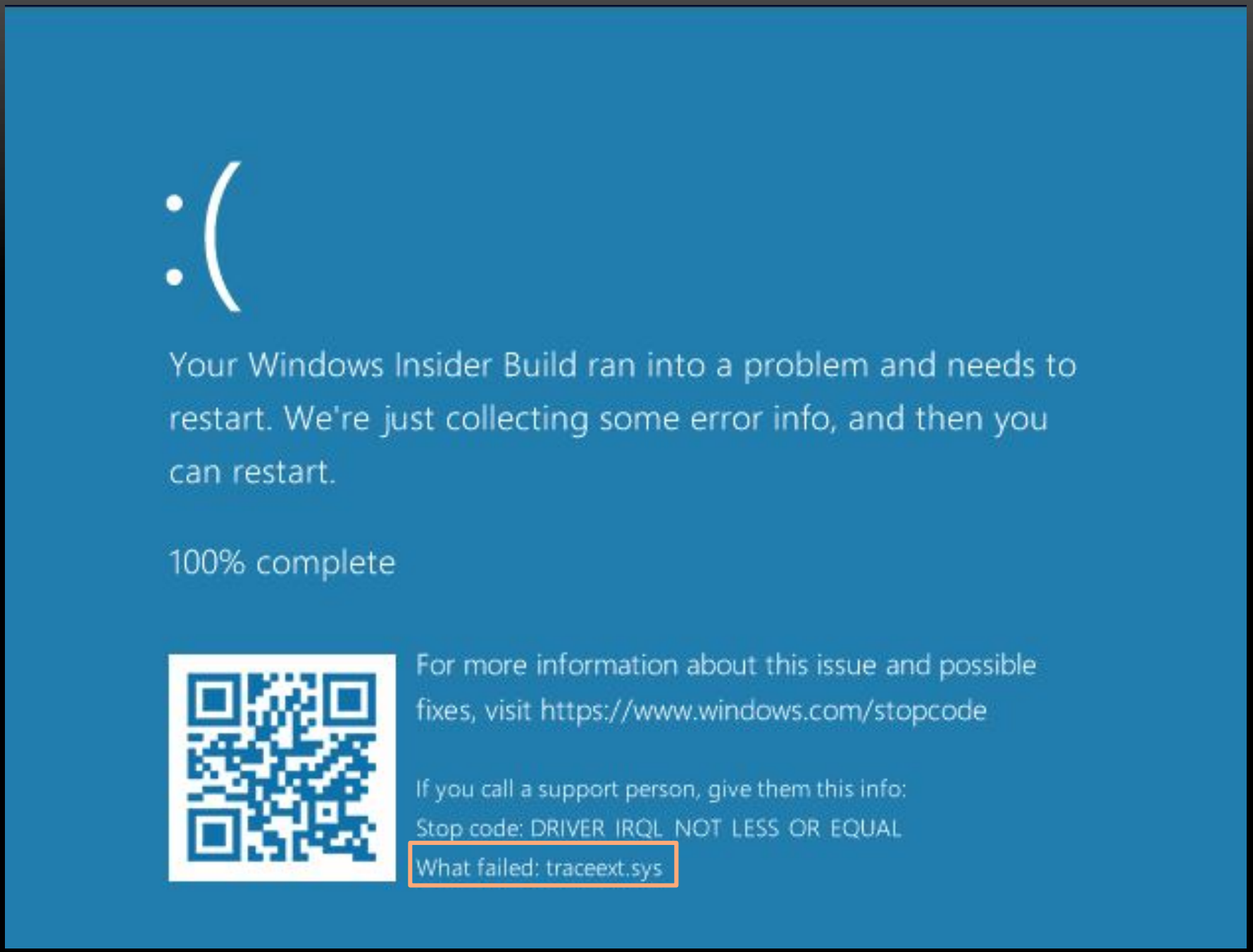
- ✓ 可以使用另一种类型的提供程序“fbt”(函数边界跟踪)，它跟踪通过内核中的NTFS执行的系统调用序列。
- ✓ **只有当**Windows 10附加内核调试器时，**才可以使用**“fbt”**提供程序**。

```
C:\>dtrace -Fn "tick-5sec { exit(0);} syscall:::entry /execname == \"chr
ome.exe\"/ { @num[probefunc] = count(); }" | tail -25
dtrace: description 'tick-5sec ' matched 465 probes
  NtDeviceIoControlFile                                    32
  NtDuplicateObject                                        40
  NtFreeVirtualMemory                                      50
  NtAllocateVirtualMemory                                  56
  NtQueryInformationThread                                 75
  NtFindAtom                                              163
  NtSetTimerResolution                                    187
  NtQuerySystemInformation                                202
  NtCreateEvent                                           328
  NtClose                                                 381
  NtQueryInformationProcess                               396
  NtClearEvent                                            428
  NtAlertThreadByThreadId                                 604
  NtWaitForAlertByThreadId                                604
  NtSetIoCompletionEx                                     684
  NtAssociateWaitCompletionPacket                        1020
  NtSetIoCompletion                                      1050
  NtDelayExecution                                       1215
  NtFlushProcessWriteBuffers                             1335
  NtRemoveIoCompletionEx                                 1702
  NtReadFile                                             2175
  NtWriteFile                                            2242
  NtSetEvent                                             2824
  NtWaitForSingleObject                                  4319
  NtRemoveIoCompletion                                   8600
```

```
C:\>dtrace -n "profile-997hz { @[pid, execname] = count( ); }"
dtrace: description 'profile-997hz ' matched 1 probe


     0  audiodg.exe                                                    1
   440  smss.exe                                                       1
   872  fontdrvhost.ex                                                 1
  1028  wlrmdr.exe                                                     1
  1136  svchost.exe                                                    1
  1368  svchost.exe                                                    1
  1400  svchost.exe                                                    1
  1532  svchost.exe                                                    1
  1652  svchost.exe                                                    1
  1728  userinit.exe                                                   1
  2176  svchost.exe                                                    1
  2512  RuntimeBroker.                                                 1
  2528  svchost.exe                                                    1
  2672  svchost.exe                                                    1
  3756  dasHost.exe                                                    1
  5004  svchost.exe                                                    1
  5828  RuntimeBroker.                                                 1
  6192  svchost.exe                                                    1
  6276  svchost.exe                                                    1
  6784  SkypeBackgroun                                                 1
  7944  SecurityHealth                                                 1
  8312  MicrosoftEdgeS                                                 1
  9084  svchost.exe                                                    1
 10052  svchost.exe                                                    1
```

```
C:\>dtrace -Fn "fbt:ntfs::/execname==\"WinRAR.exe\"/{}" | more
dtrace: description 'fbt:ntfs::' matched 7752 probes
CPU FUNCTION
  0  -> NtfsFsdDispatchWait
  0    -> memset
  0    <- memset
  0    -> NtfsFsdDispatchSwitch
  0      -> NtfsInitializeTopLevelIrp
  0      <- NtfsInitializeTopLevelIrp
  0      -> memset
  0      <- memset
  0      -> NtfsInitializeIrpContextInternal
  0      <- NtfsInitializeIrpContextInternal
  0      -> NtfsUpdateIrpContextWithTopLevel
  0      <- NtfsUpdateIrpContextWithTopLevel
  0      -> NtfsPreRequestProcessingExtend
  0      <- NtfsPreRequestProcessingExtend
  0      -> NtfsCommonQueryInformation
  0        -> NtfsAcquireExclusiveFcb
  0        <- NtfsAcquireExclusiveFcb
  0        -> TxfSetupTransactionContextFromCcb
  0        <- TxfSetupTransactionContextFromCcb
  0        -> NtfsQueryNameInfo
```


:(
Your Windows Insider Build ran into a problem and needs to restart. We're just collecting some error info, and then you can restart.
100% complete
For more information about this issue and possible fixes, visit https://www.windows.com/stopcode
If you call a support person, give them this info:
Stop code: DRIVER IRQL NOT LESS OR EQUAL
What failed: traceext.sys

```
1: kd> k
 # Child-SP          RetAddr           Call Site
00 fffffd8d`c02a0198 fffff802`21fe5469 nt!KeBugCheckEx
01 fffffd8d`c02a01a0 fffff802`21fe17a5 nt!KiBugCheckDispatch+0x69
02 fffffd8d`c02a02e0 fffff802`217b8e1e nt!KiPageFault+0x465
03 fffffd8d`c02a0478 fffff803`01e932bf traceext!StpGetArgVal+0xe
04 fffffd8d`c02a0480 fffff803`01e95e2c DTrace!dtrace_dif_variable+0x1e7
05 fffffd8d`c02a0540 fffff803`01e972d8 DTrace!dtrace_dif_emulate+0x754
06 fffffd8d`c02a0760 fffff802`217b76cd DTrace!dtrace_probe+0x478
07 fffffd8d`c02a0930 fffff802`217c16cc traceext!dtrace_probe+0x29
08 fffffd8d`c02a0980 fffff802`226949d4 traceext!StpCallbackEntry+0x7c
09 fffffd8d`c02a09e0 fffff802`21fe534d nt!KiTrackSystemCallEntry+0xd4
0a fffffd8d`c02a0a40 00007ff9`2edfc164 nt!KiSystemServiceExitPico+0x238
0b 000000ac`bc7fb918 00000000`00000000 0x00007ff9`2edfc164
1: kd> .lastevent
Last event: Break instruction exception - code 80000003 (first/second chance not available)
  debugger time: Sun Apr 28 22:00:04.067 2019 (UTC - 7:00)
1: kd> lmv m traceext
Browse full module list
start             end                 module name
fffff802`217b6000 fffff802`217d1000   traceext   (pdb symbols)          c:\symbols\traceext
    Loaded symbol image file: traceext.sys
    Image path: traceext.sys
    Image name: traceext.sys
    Browse all global symbols  functions  data
    Image was built with /Brepro flag.
    Timestamp:        414AF89D (This is a reproducible build file hash, not a timestamp)
    CheckSum:         00016962
    ImageSize:        0001B000
    Translations:     0000.04b0 0000.04e4 0409.04b0 0409.04e4
    Information from resource tables:
1: kd> x /D traceext!s*
 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

fffff802`217b8e10 traceext!StpGetArgVal (void)
fffff802`217c1770 traceext!StpDisable (void)
fffff802`217c1880 traceext!StpProvide (void)
fffff802`217c1710 traceext!StpEnable (void)
fffff802`217b8e50 traceext!StpGetContext (void)
fffff802`217c17c0 traceext!StpGetArgType (void)
fffff802`217c1970 traceext!StpDestroy (void)
fffff802`217c1650 traceext!StpCallbackEntry (<no parameter info>)
fffff802`217b9146 traceext!strcmp (<no parameter info>)
fffff802`217c16e0 traceext!StpCallbackReturn (<no parameter info>)
```

Traceext.sys: 将DTrace使用的功能公开给跟踪。

```
1: kd> uf fffff802`217b8e10
traceext!StpGetArgVal:
fffff802`217b8e10 488b542428      mov     rdx,qword ptr [rsp+28h]
fffff802`217b8e15 4885d2          test    rdx,rdx
fffff802`217b8e18 742a            je      traceext!StpGetArgVal+0x34 (fffff802`217b8e44)  Branch

traceext!StpGetArgVal+0xa:
fffff802`217b8e1a 4983e0fe        and     r8,0FFFFFFFFFFFFFFFEh
fffff802`217b8e1e 410fb74008      movzx   eax,word ptr [r8+8]
fffff802`217b8e23 443bc8          cmp     r9d,eax
fffff802`217b8e26 7d1c            jge     traceext!StpGetArgVal+0x34 (fffff802`217b8e44)  Branch

traceext!StpGetArgVal+0x18:
fffff802`217b8e28 8b4210          mov     eax,dword ptr [rdx+10h]
fffff802`217b8e2b 443bc8          cmp     r9d,eax
fffff802`217b8e2e 7d0c            jge     traceext!StpGetArgVal+0x2c (fffff802`217b8e3c)  Branch

traceext!StpGetArgVal+0x20:
fffff802`217b8e30 488b4208        mov     rax,qword ptr [rdx+8]

traceext!StpGetArgVal+0x24:
fffff802`217b8e34 4963c9          movsxd  rcx,r9d
fffff802`217b8e37 488b04c8        mov     rax,qword ptr [rax+rcx*8]
fffff802`217b8e3b c3              ret

traceext!StpGetArgVal+0x2c:
fffff802`217b8e3c 442bc8          sub     r9d,eax
fffff802`217b8e3f 488b02          mov     rax,qword ptr [rdx]
fffff802`217b8e42 ebf0            jmp     traceext!StpGetArgVal+0x24 (fffff802`217b8e34)  Branch

traceext!StpGetArgVal+0x34:
fffff802`217b8e44 33c0            xor     eax,eax
fffff802`217b8e46 c3              ret
1: kd> vertarget
Windows 10 Kernel Version 18362 MP (2 procs) Free x64
Product: WinNt, suite: TerminalServer SingleUserTS
Built by: 18362.1.amd64fre.19h1_release.190318-1202
Machine Name:
Kernel base = 0xfffff802`21e17000 PsLoadedModuleList = 0xfffff802`2225a290
Debug session time: Sun Apr 28 19:11:07.480 2019 (UTC - 7:00)
System Uptime: 0 days 2:40:06.813
```

反虚拟化

- ✓ 使用反虚拟化技术编写恶意软件样本非常容易，这些技术用于检测VMWare(检查I/O端口通信)、VirtualBox、Parallels、SeaBIOS仿真器、QEMU仿真器、Bochs仿真器、QEMU仿真器、Hyper-V、Innotek VirtualBox、sandbox (Cuckoo)。

- ✓ 此外，有几十种技术可以用于检测Vmware沙箱:

  - ✓ 检查注册表(OpenSubKey()函数)，以尝试找到与客户端中安装的工具相关的条目 (HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VirtualMachine\Guest\Parameters).

  - ✓ 使用WMI查询Win32_BIOS管理类，以与物理机器中的属性进行交互。

- ✓ 我们已经知道了世界上每一种反虚拟化技术，并且所有这些技术都有文档。

- ✓ 目前大多数技术都使用WMI，使用WMI可以快速编写C#程序。

```
using System;
using System.Management;

namespace Test_VM
{
    class Program
    {
        static void Main(string[] args)
        {
            ManagementClass biosClass =
            new ManagementClass("Win32_BIOS");
            ManagementObjectCollection biosc =
                biosClass.GetInstances();
            ManagementObjectCollection.ManagementObjectEnumerator
                biosEnumerator =
                biosc.GetEnumerator();
            while (biosEnumerator.MoveNext())
            {
                ManagementObject biosc1 =
                    (ManagementObject)biosEnumerator.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;
        }
    }
}
```

- ✓ 上一张幻灯片的代码没有任何**新信息**:
  - ✓ ManagementClass class **代表 公共信息模型**(CIM)管理类。
  - ✓ Win32_BIOS WMI class **代表** BIOS**的属性**和该类的成员使您能够使用特定的WMI类路径访问WMI**数据**。
  - ✓ GetInstances( )获取类的所有实例的集合。
  - ✓ GetEnumerator( ) 返回集合的枚举器(IEnumerator)。
  - ✓ IEnumerator.Current( ) 返回相同的对象。
  - ✓ IEnumerator.MoveNext( ) 将枚举数推进到集合的下一个元素。

❑ **物理主机**:

C:\> Test_VM.exe
属性:
版本: DELL - 6222004
序列号: D5965S1
操作系统: 0
生产商: Dell Inc.

❑ **客户虚拟机**:

E:\> Test_VM.exe
属性:
版本: LENOVO - 6040000
序列号: VMware-56 4d 8d c3 a7 c7 e5 2b-39 d6 cc 93 bf 90 28 2d
操作系统: 0
生产商: Phoenix Technologies LTD

ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER

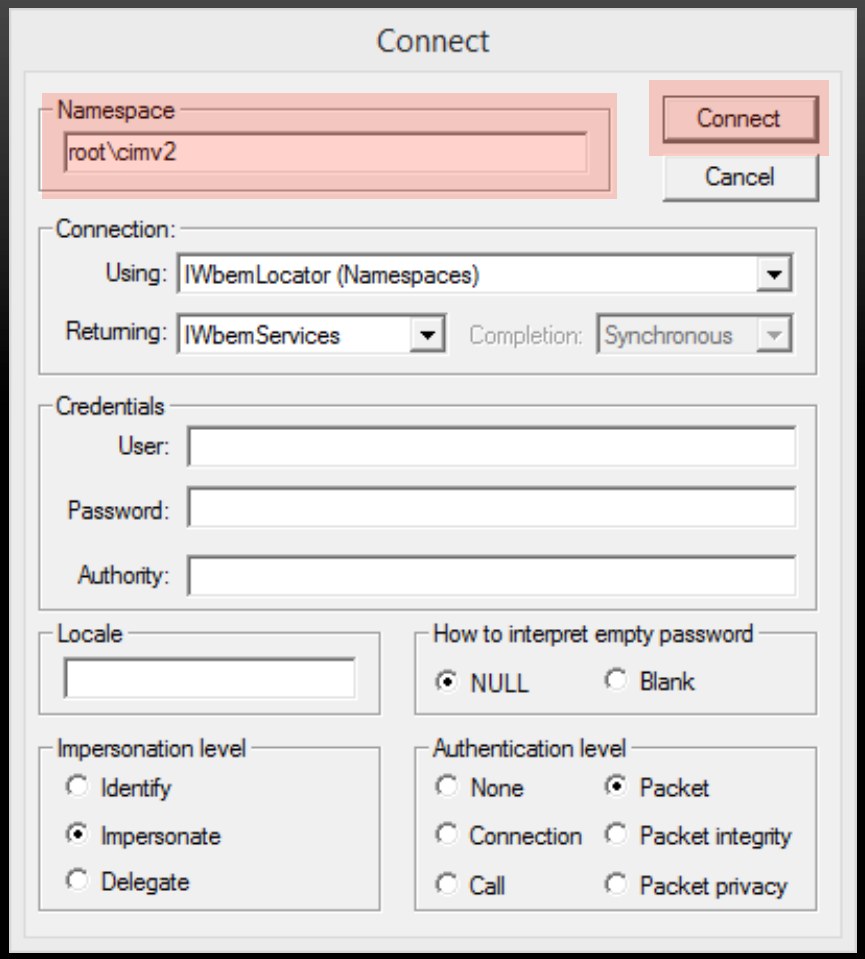
```
namespace TestVM_3
{
    class Program
    {
        static void Main(string[] args)
        {
            ManagementClass tempClass =
            new ManagementClass("Win32_TemperatureProbe");
            ManagementObjectCollection tempinstance =
                tempClass.GetInstances() ;
            foreach (ManagementObject aborges in tempinstance)
            {
                string buffer = aborges.GetPropertyValue("CurrentReading").ToString( );
                {

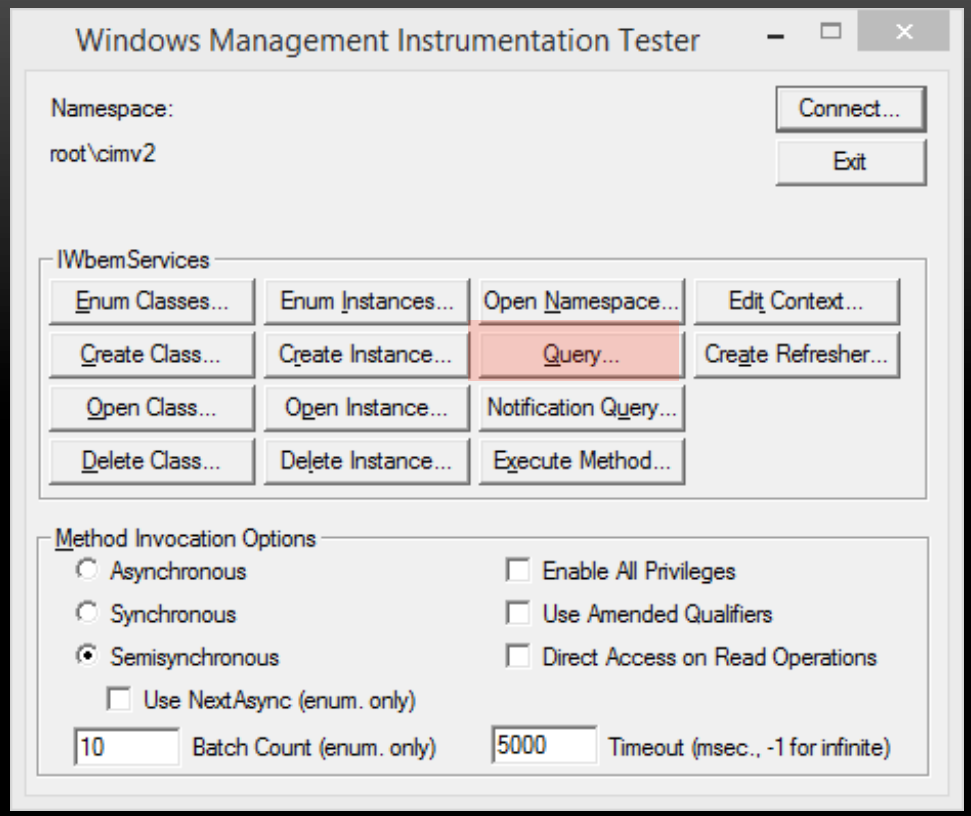
                    Console.WriteLine("Temperature:\t" + buffer);

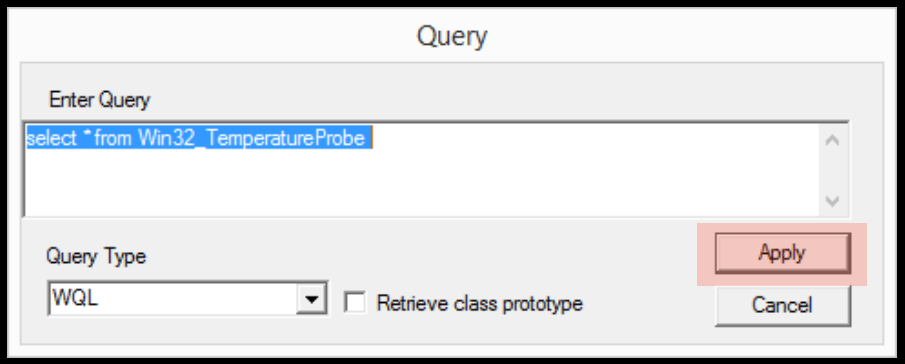
                }
            }
        }
    }
}
```

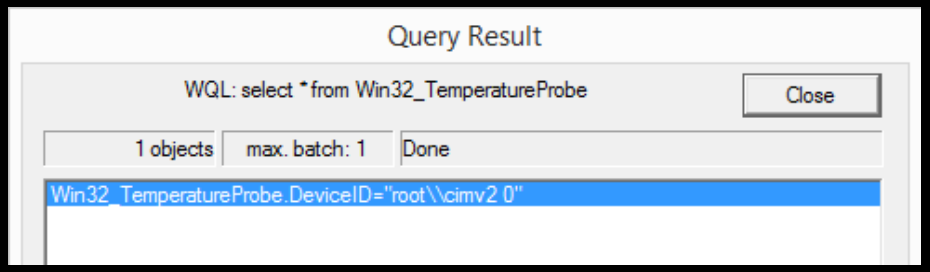
```
c:\Users\Administrador\source\repos\TestVM_3\TestVM_3\bin\Debug>TestVM_3.exe

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
```



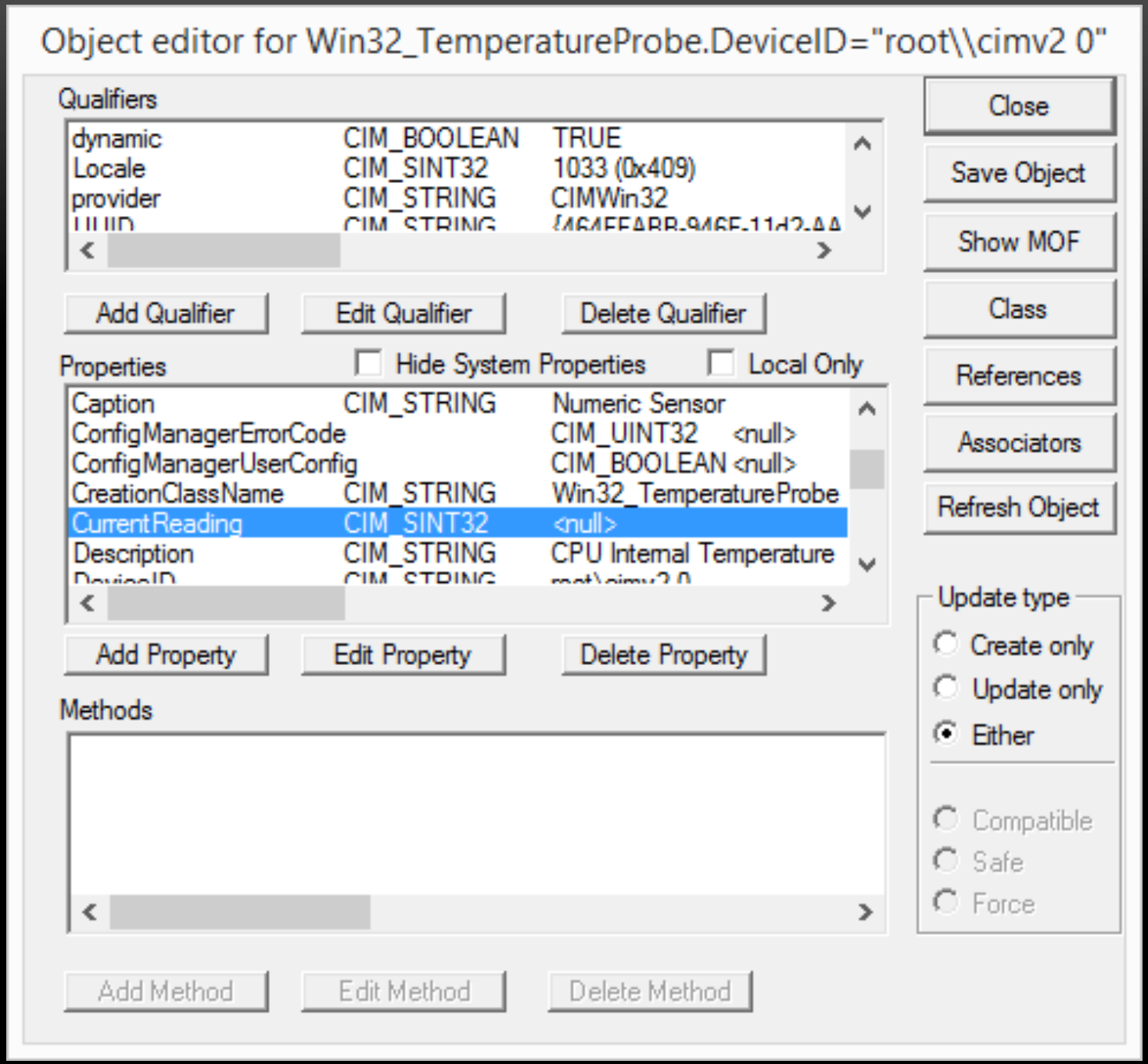






双击结果....

ALEXANDRE BORGES – MALWARE AND SECURITY RESEARCHER

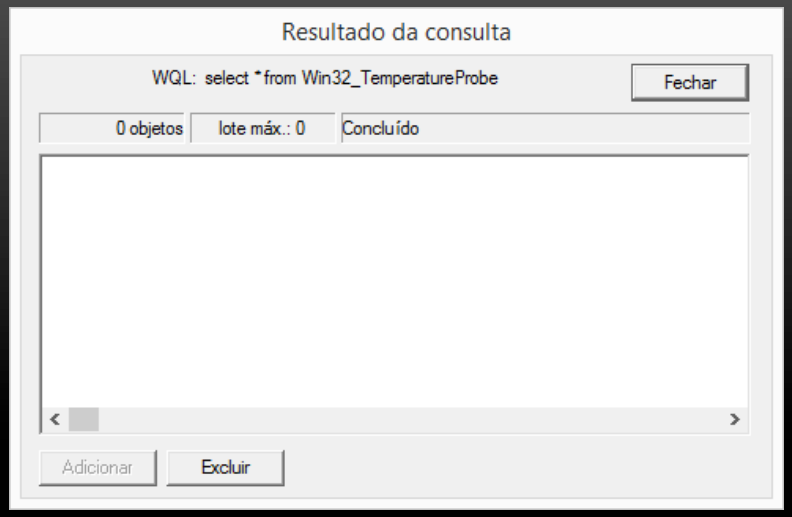

Object editor for Win32_TemperatureProbe.DeviceID="root\\cimv2 0"
Qualifiers
dynamic CIM_BOOLEAN TRUE
Locale CIM_SINT32 1033 (0x409)
provider CIM_STRING CIMWin32
Add Qualifier
Edit Qualifier
Delete Qualifier
Properties
Hide System Properties
Local Only
Caption CIM_STRING Numeric Sensor
ConfigManagerErrorCode CIM_UINT32 <null>
ConfigManagerUserConfig CIM_BOOLEAN <null>
CreationClassName CIM_STRING Win32_TemperatureProbe
CurrentReading CIM_SINT32 <null>
Description CIM_STRING CPU Internal Temperature
Add Property
Edit Property
Delete Property
Methods
Add Method
Edit Method
Delete Method
Close
Save Object
Show MOF
Class
References
Associators
Refresh Object
Update type
Create only
Update only
Either
Compatible
Safe
Force

```
using System;
using System.Management;

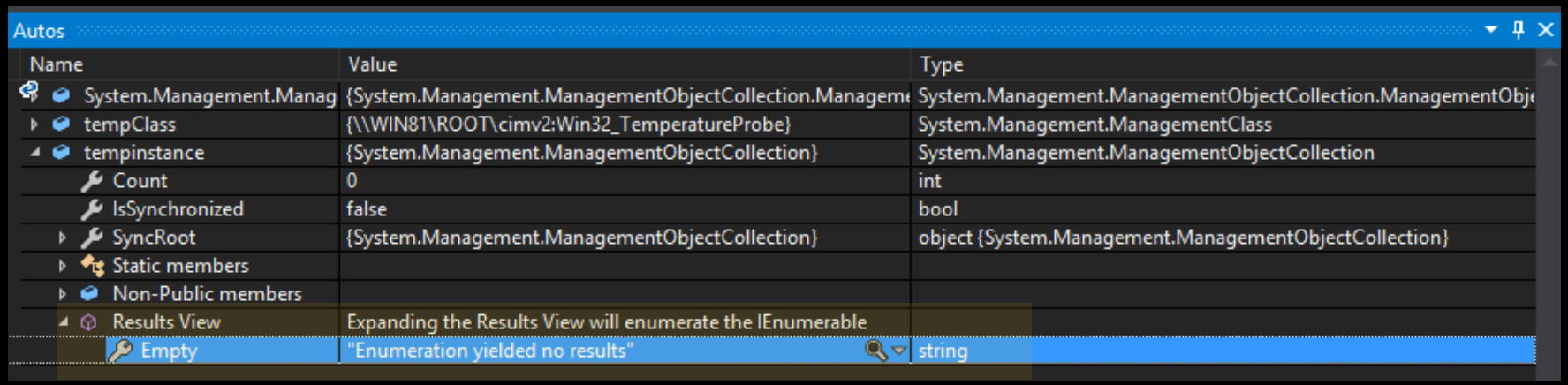
namespace TestVM_3
{
    public class Program
    {
        public static void Main(string[] args)
        {
            ManagementClass tempClass =
            new ManagementClass("Win32_TemperatureProbe");
            ManagementObjectCollection tempinstance = tempClass.GetInstances();

            foreach (ManagementObject aborges in tempinstance)
            {
                try
                {
                    if (!string.IsNullOrWhiteSpace(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!");
        }
    }
}
```

[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 membe		



- ✓ 在虚拟机中不支持获取温度数据。
- ✓ 因此，恶意软件能够知道它们是否在虚拟机上运行。☺



- ✓ 物理主机:

C:\> VM_Test2.exe

状态:好的，程序在物理主机上运行!

- ✓ 虚拟机:

C:\> VM_Test2.exe

这个程序正在虚拟机中运行!

❑ 几点结论:

- ✓ 在尝试打开现代保护器之前，了解常用的反逆向技术是非常必要的。
- ✓ MIASM和METASM是处理和消除复杂代码混淆的出色工具。
- ✓ 仿真是理解小而复杂的代码段的一种可能的替代方法。
- ✓ DTrace在Solaris上做得很好，请继续关注它可能是Windows操作系统上的一个优秀工具。 ☺
- ✓ 尽管优秀的研究已经发现了复杂的反虚拟化技术，但仍然存在许多其他简单而智能的技术，所以请当心！

❖ Acknowledgments to:

✓DEF CON的工作人员, 对我非常友好

✓你, 预留了一些时间来听我的演讲。

✓记住: 人生最美好的是人. ☺



感谢您的聆听. ☺
謝謝

- ✓ Malware and Security Researcher.
- ✓ Speaker at DEFCON USA 2018
- ✓ Speaker at HITB2019 Amsterdam
- ✓ Speaker at CONFidence Conf. 2019
- ✓ Speaker at BSIDES 2018/2017/2016
- ✓ Speaker at H2HC 2016/2015
- ✓ Speaker at BHACK 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.

- **Twitter:**

  @ale_sp_brazil
  @blackstormsecbr

- **Website:** http://blackstormsecurity.com
- **LinkedIn:** http://www.linkedin.com/in/aleborges
- **E-mail:** alexandreborges@blackstormsecurity.com

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