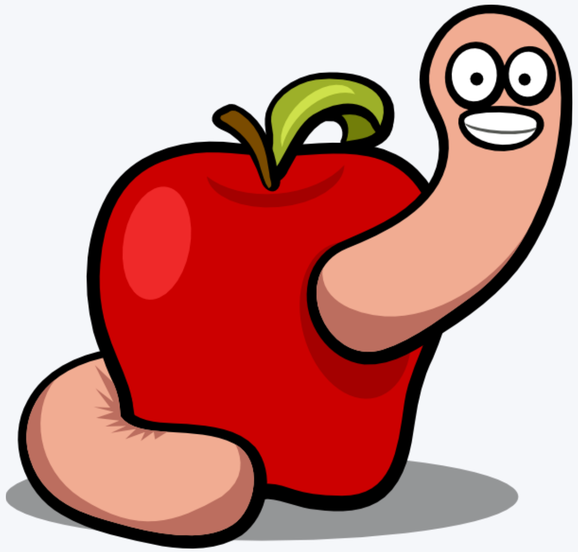
Where is the flag?

Hardcore++ version



fG! @ ØxO P O S₹C

December 2023

Who am I

U.PORTO
FEP FACULDADE DE ECONOMIA UNIVERSIDADE DO PORTO

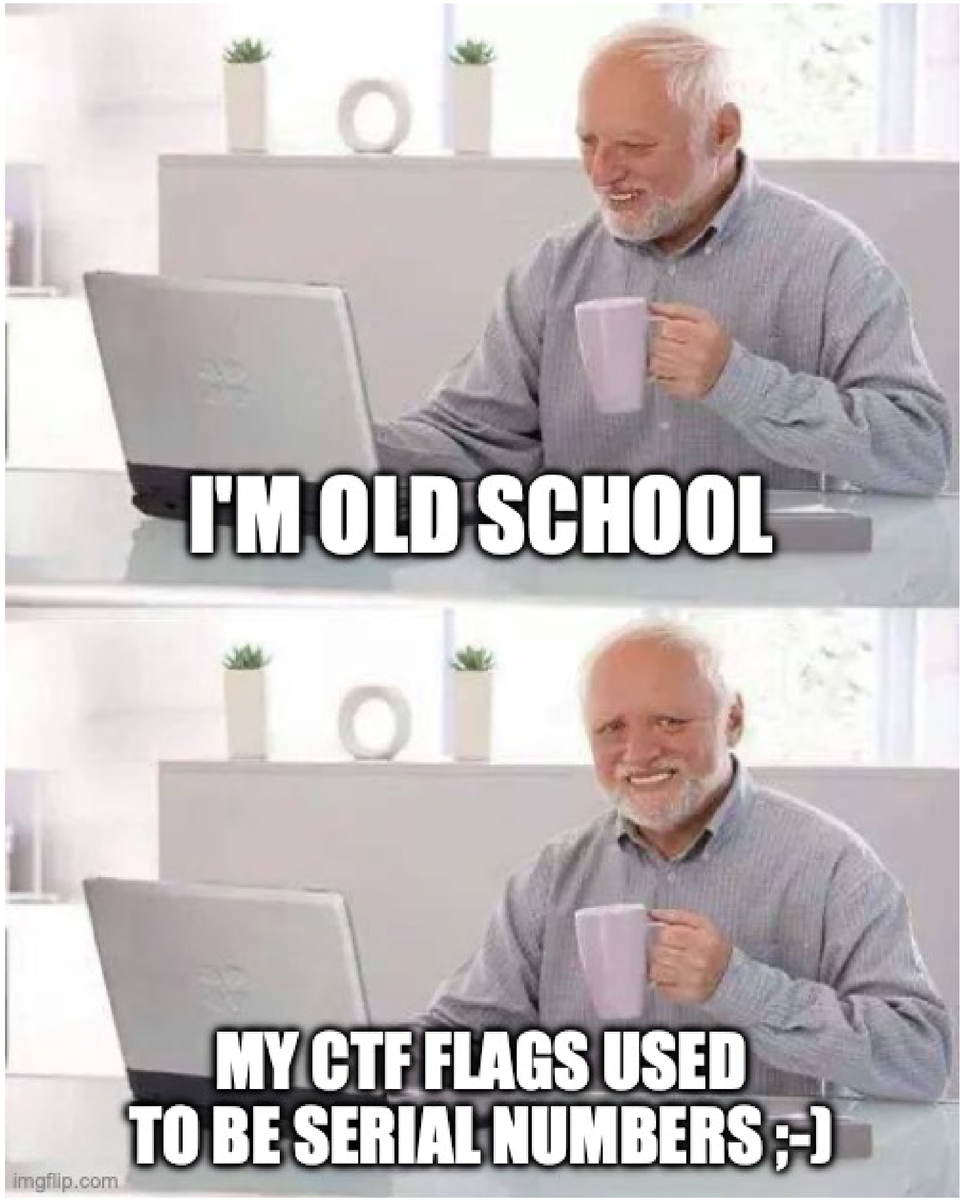
Porto Business School

MB MULTIBANCO

SentinelOne®

paloalto® NETWORKS

Who am I


I'M OLD SCHOOL
MY CTF FLAGS USED
TO BE SERIAL NUMBERS ;-)
imgflip.com

Knowledge and code:

- https://reverse.put.as
- https://github.com/gdbinit

If you like reading:

- https://links.put.as
- https://one.adayfullofpossibilities.com

Today’s Agenda

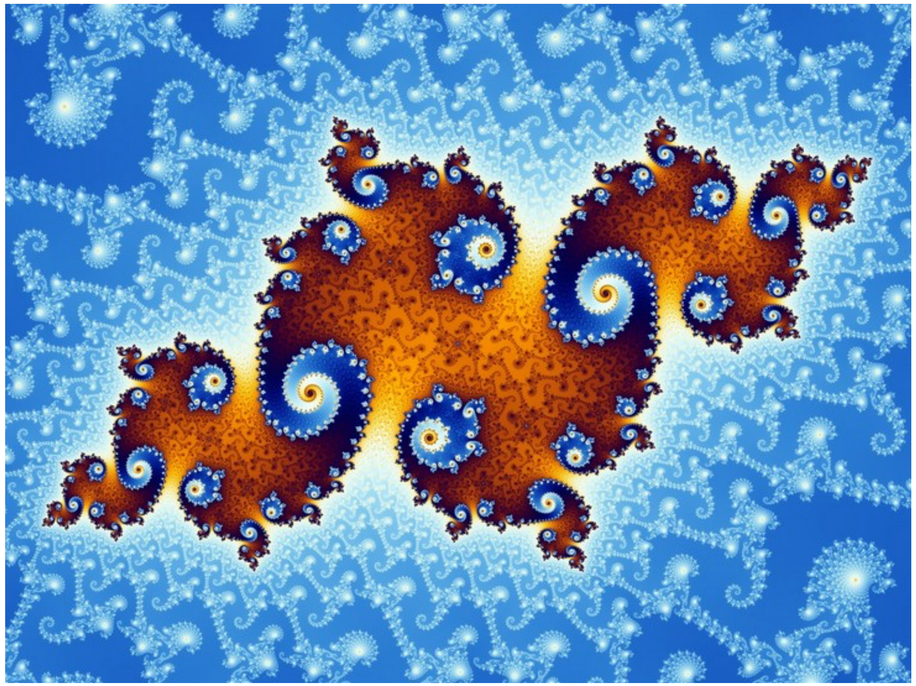
Today's Agenda

- Flare On 2023 - 10 year anniversary.
- Challenge #12 (13 total).
- A cute virtual machine :-).

"This is the second smallest challenge this year! If only that mattered."

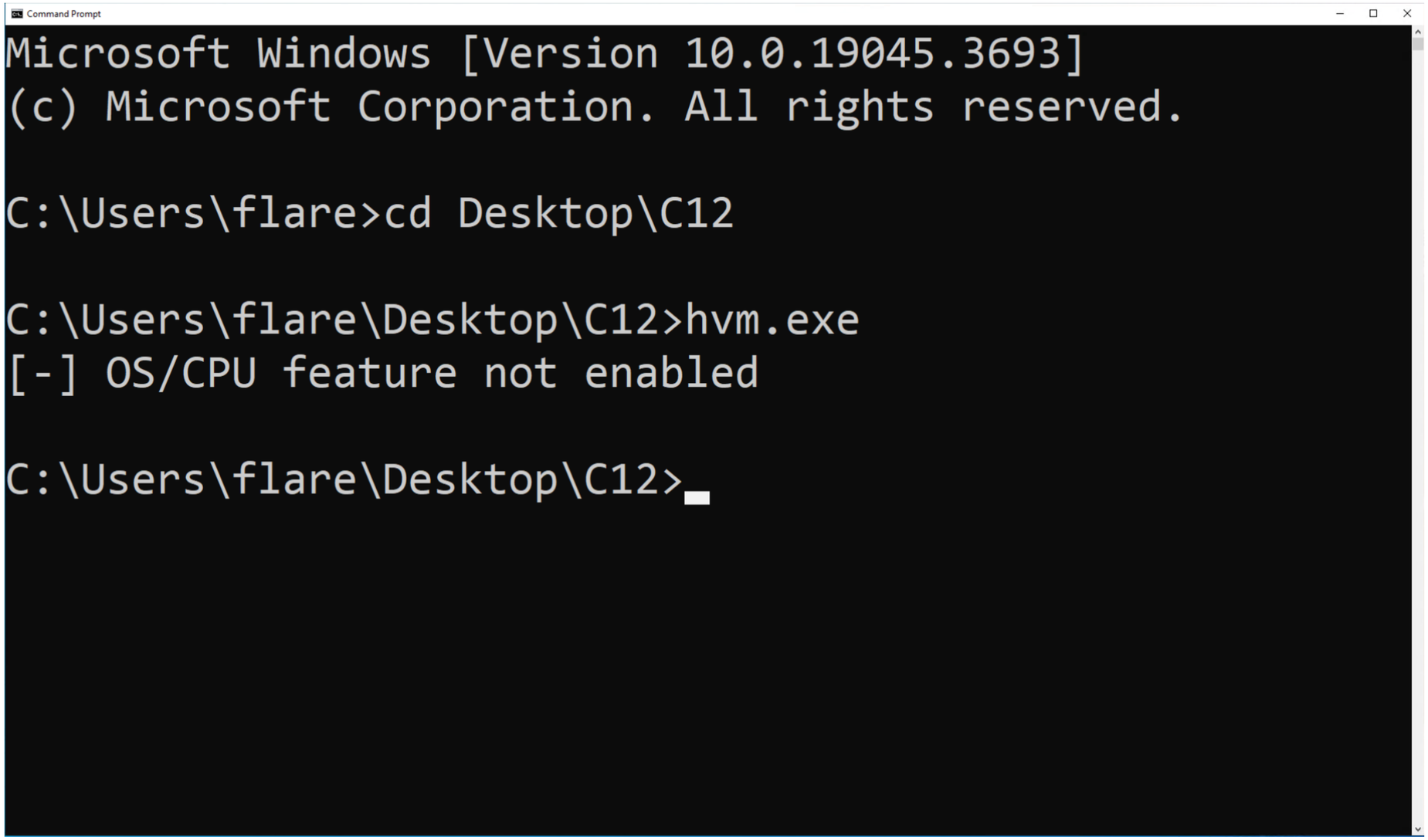
Today’s Agenda

- Reverse engineering is not a linear process.
- Every RE presentation is rewritten history.
- (Many) Different approaches to the same problem.

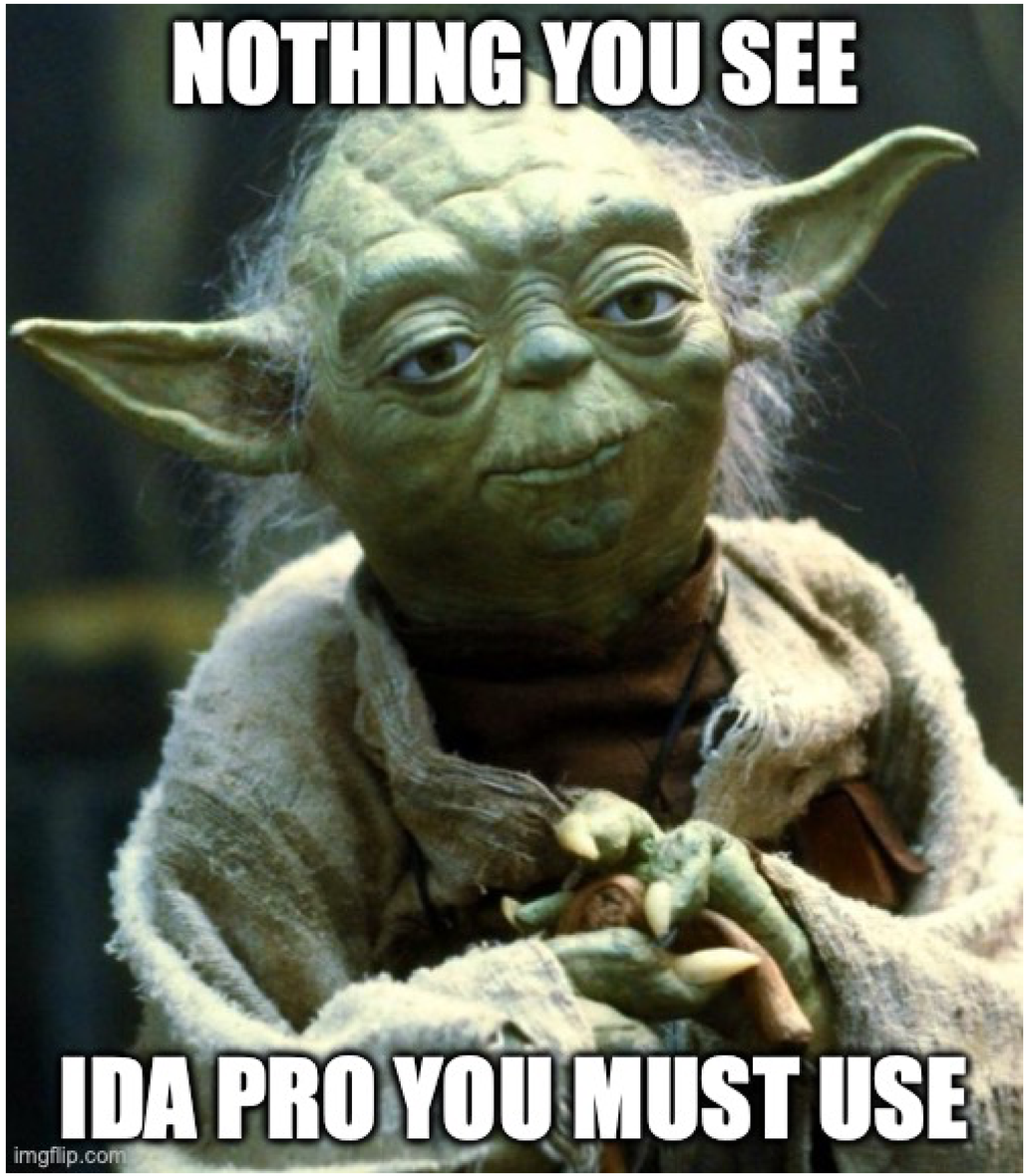


Initial Recon

Initial Recon



Initial Recon



Initial Recon

```
00000001400017C0 ; int __cdecl main(int argc, const char **argv, const char **envp)
00000001400017C0 main      proc near                    ; CODE XREF: __scrt_common_main_seh(void)+107↓p
00000001400017C0                                        ; DATA XREF: .pdata:0000000140021084↓o
00000001400017C0
00000001400017C0 Partition = qword ptr -190h
00000001400017C0 ExitContext= byte ptr -138h
00000001400017C0
00000001400017C0 ; __unwind { // __GSHandlerCheck
00000001400017C0           mov     [rsp+10h], rdx
00000001400017C5           mov     [rsp+8], ecx
00000001400017C9           push    rsi
00000001400017CA           push    rdi
00000001400017CB           sub     rsp, 1B8h
00000001400017D2           mov     rax, cs:__security_cookie
00000001400017D9           xor     rax, rsp
00000001400017DC           mov     [rsp+1A0h], rax
00000001400017E4           call    sub_140001000
00000001400017E9           test    eax, eax
00000001400017EB           jnz     short loc_140001803
00000001400017ED           lea     rcx, Format          ; "[-] OS/CPU feature not enabled\n"
00000001400017F4           call    printf
00000001400017F9           mov     eax, 0FFFFFFFFh
00000001400017FE           jmp     loc_140001D4B
```

Initial Recon

```
__int64 sub_140001000()
{
  unsigned int ret = 0;
  UINT32 WrittenSizeInBytes = 0;
  HRESULT Capability;
  unsigned int CapabilityBuffer[10];

  Capability = WHvGetCapability(WHvCapabilityCodeHypervisorPresent, CapabilityBuffer, 0x18, &WrittenSizeInBytes);

  if ( Capability >= 0 ) {
    return CapabilityBuffer[0];
  }
  return ret;
}
```

Windows Hypervisor Platform API Definitions

Article • 05/02/2022 • 4 contributors

Platform Capabilities

Function	Description
WHvGetCapability	Platform capabilities are a generic way for callers to query properties and capabilities of the hypervisor, of the API implementation, and of the hardware platform that the application is running on. The platform API uses these capabilities to publish the availability of extended functionality of the API as well as the set of features that the processor on the current system supports.

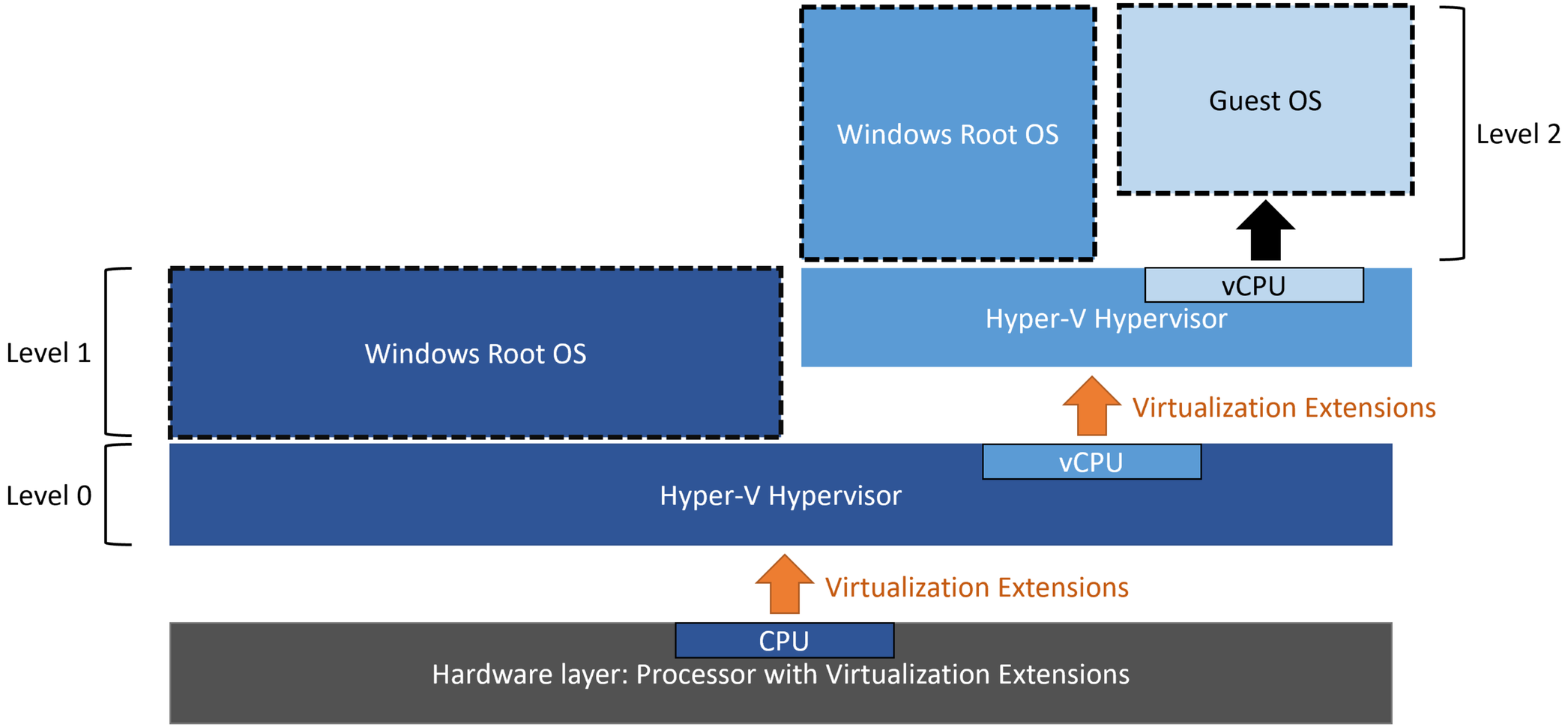
The `WHvCapabilityCodeHypervisorPresent` capability can be used to determine whether the Windows Hypervisor is running on a host and the functions of the platform APIs can be used to create VM partitions.

Initial Recon

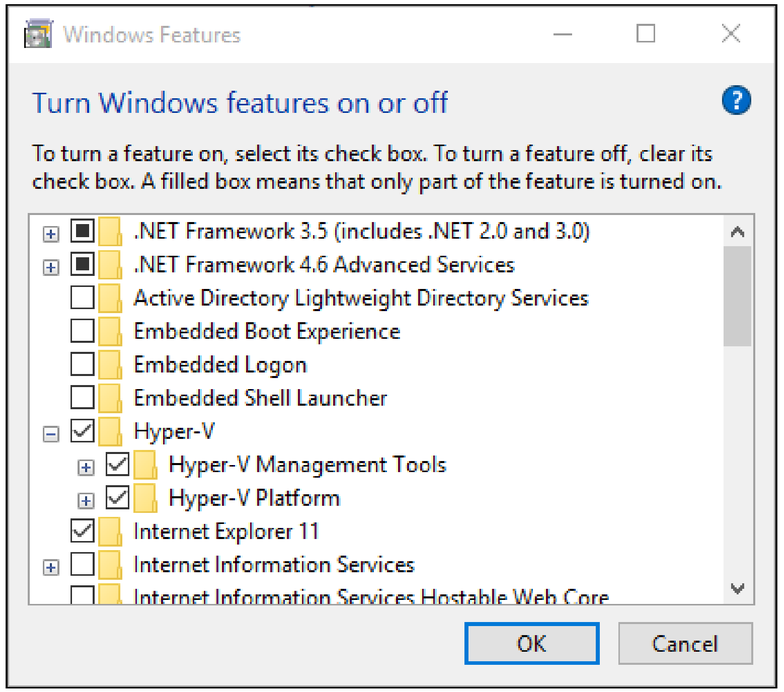
- Code requires hypervisor capability.
- If we patch this check it fails later when trying to create and run a virtual machine.
- My Windows VM is running under KVM/QEMU.
- Nested hypervisor needs to be enabled.

Initial Recon

Initial Recon



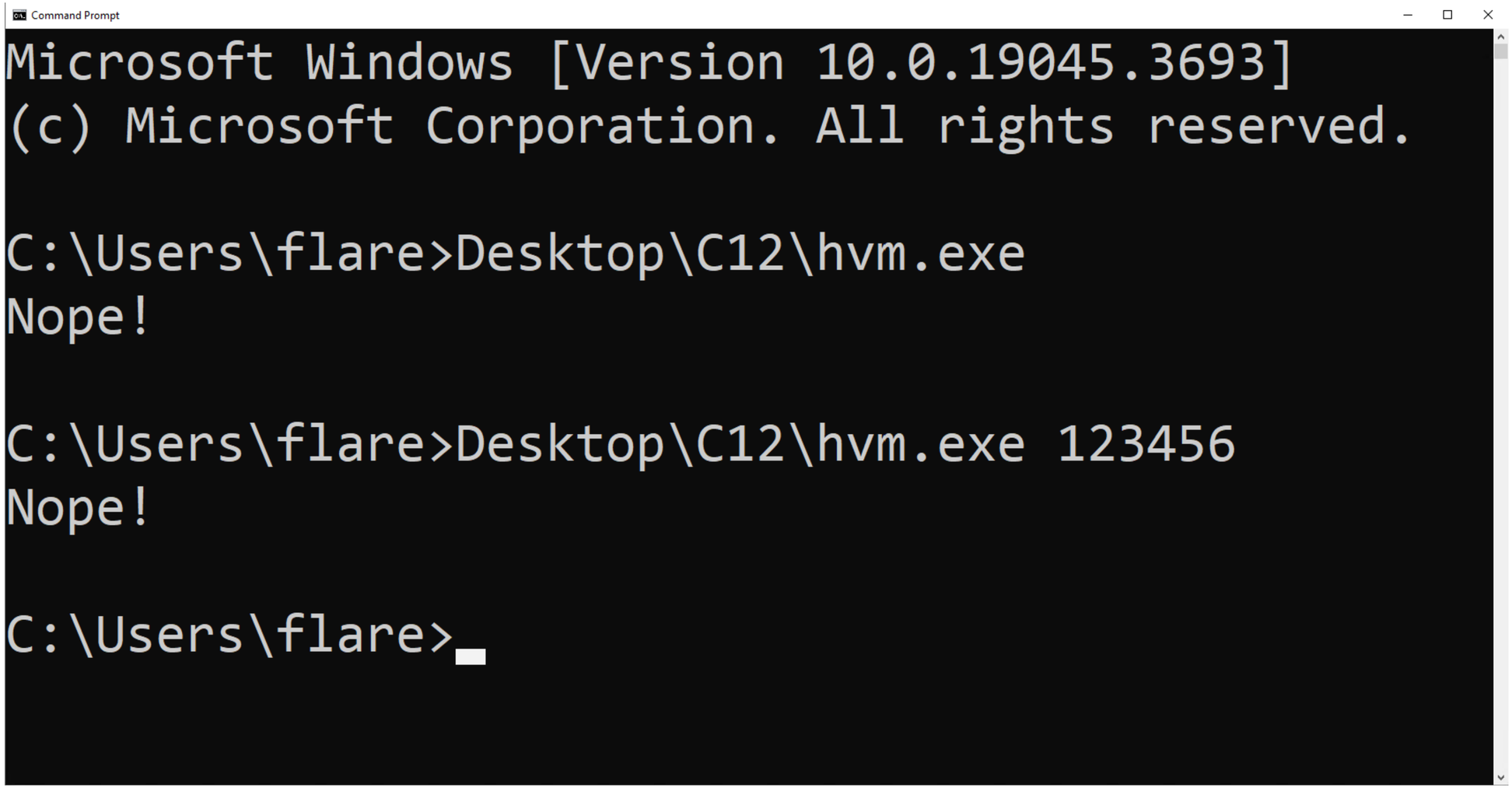
- We need to install Hyper-V (Pro or higher versions only).
- https://learn.microsoft.com/en-us/virtualization/hyper-v-on-windows/quick-start/enable-hyper-v



- And configure the host KVM hypervisor to allow nesting:
- `virsh edit flare`:

```
<cpu mode='custom' match='exact' check='partial'>
  <model fallback='allow'>Skylake-Client-noTSX-IBRS</model>
  <feature policy='disable' name='hypervisor'/>
  <feature policy='require' name='vmx'/>
</cpu>
```

Initial Recon



We love NOPs not NOPEs

We Love NOPs not NOPEs

We Love NOPs not NOPEs

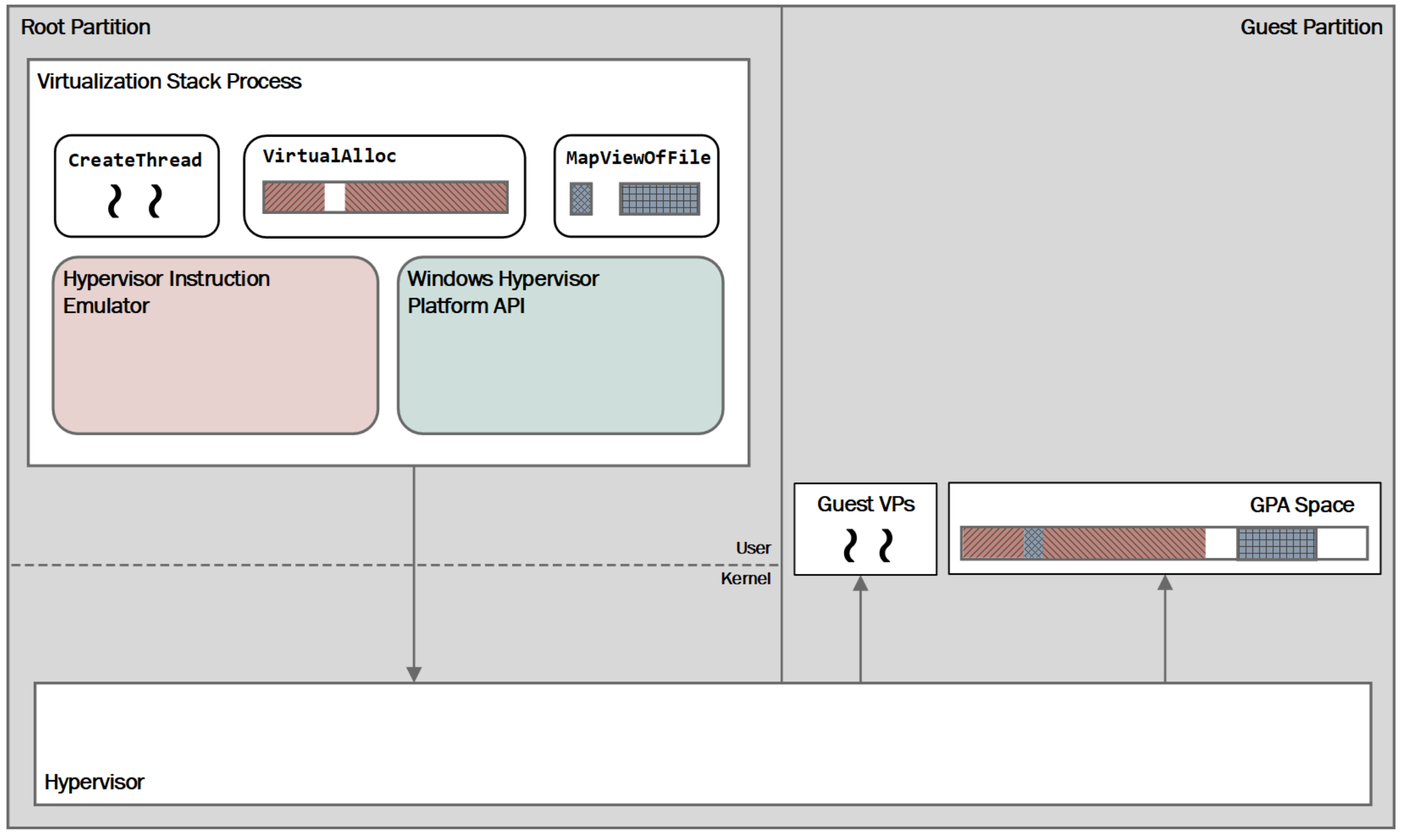
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  if ( (unsigned int)fg_IsHypervisorPresent(argc, argv, envp) ) {
    if ( argc == 3 ) {
      v10 = strlen(argv[1]);
      v9 = strlen(argv[2]);
      if ( v10 > 8 && v10 < 48 ) {
        if ( v9 > 24 && v9 < 65 ) {
          if ( v9 % 4 ) {
            printf("Nope!\n");
            return -1;
          }
```

- Two arguments are required.
- 8 < strlen(argv[1]) < 48.
- 24 < strlen(argv[2]) < 65.
- Second argument length must be a multiple of 4.
  - That’s an hint for its contents => Base64.
  - Only noticed it while writing this.

We Love NOPs not NOPEs

```
else if ( WHvCreatePartition(&Partition) >= 0 ) {

  if ( WHvSetPartitionProperty(Partition, WHvPartitionPropertyCodeProcessorCount,
    &unk_14001F048, 4) >= 0 ) {

    if ( WHvSetupPartition(Partition) >= 0 ) {
      v7 = (char *)VirtualAlloc(0, (unsigned int)SizeInBytes, 0x1000, 4);

      if ( v7 ) {
        sub_140001D90(v7, (unsigned int)SizeInBytes);
        v4 = unknown_libname_4(1, 2);
        v5 = (unsigned int)unknown_libname_4(v4, 4);

        if ( WHvMapGpaRange(Partition, v7, 0, (unsigned int)SizeInBytes, v5) >= 0 ) {

          if ( WHvCreateVirtualProcessor(Partition, 0, 0) >= 0 ) {
```

- Windows Hypervisor Platform API.
- Sample code: https://github.com/utshina/WHP-simple.

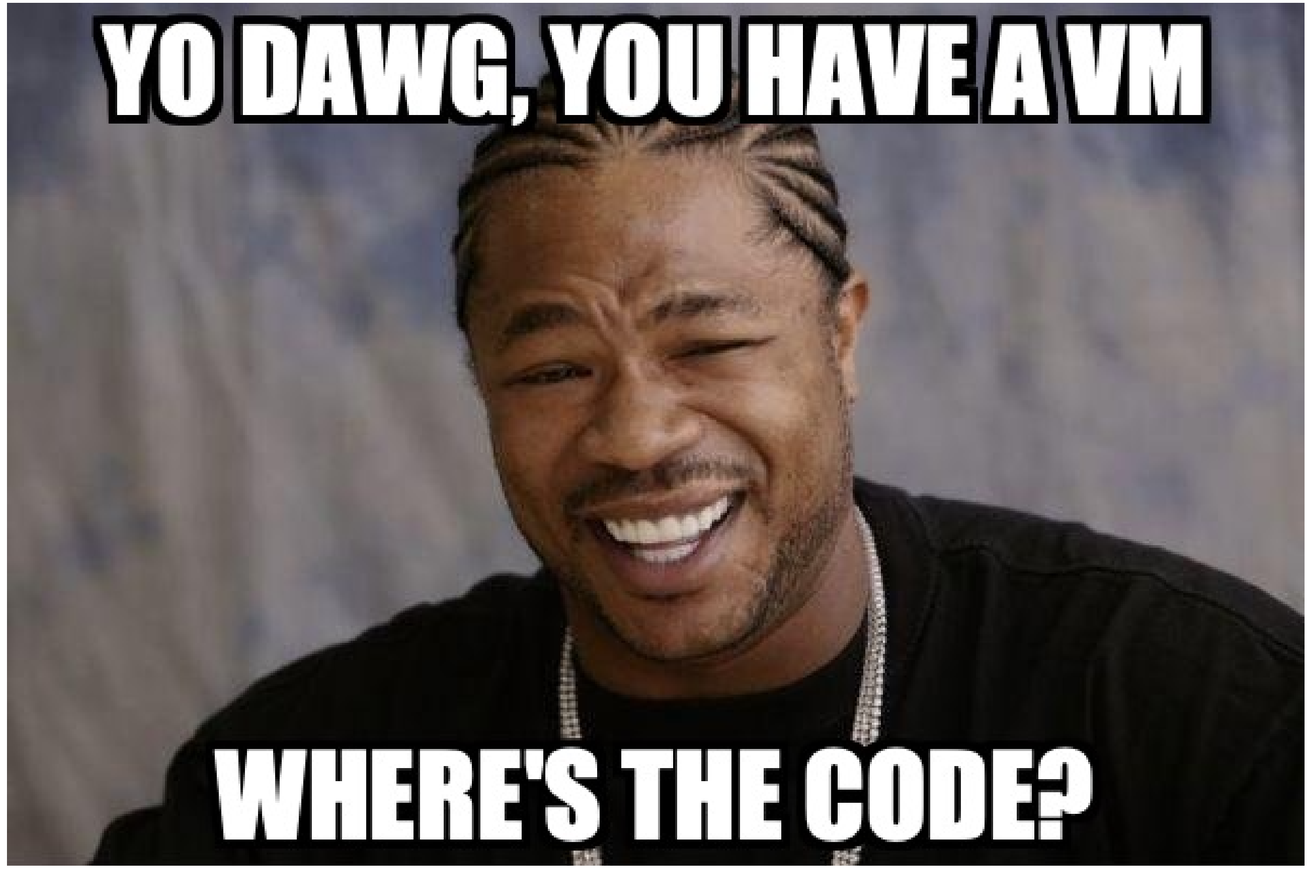


We Love NOPs not NOPEs

```
// Create the partition object
else if ( WHvCreatePartition(&Partition) >= 0 ) {
  // Configure 1 virtual CPU for the partition
  if ( WHvSetPartitionProperty(Partition, WHvPartitionPropertyCodeProcessorCount,
    &unk_14001F048, 4) >= 0 ) {
    // Create the partition in the hypervisor
    if ( WHvSetupPartition(Partition) >= 0 ) {
      // Allocate memory buffer - 0x10000 bytes
      v7 = (char *)VirtualAlloc(0, (unsigned int)SizeInBytes, 0x1000, 4);
      if ( v7 ) {
        // this is just a memset
        sub_140001D90(v7, (unsigned int)SizeInBytes);
        v4 = unknown_libname_4(1, 2);
        v5 = (unsigned int)unknown_libname_4(v4, 4);
        // map the memory buffer into the partition Guest Physical Address (GPA)
        // essentially the memory space for the virtual machine
        if ( WHvMapGpaRange(Partition, v7, 0, (unsigned int)SizeInBytes, v5) >= 0 ) {
          // create the VM CPU at index 0
          if ( WHvCreateVirtualProcessor(Partition, 0, 0) >= 0 ) {
```

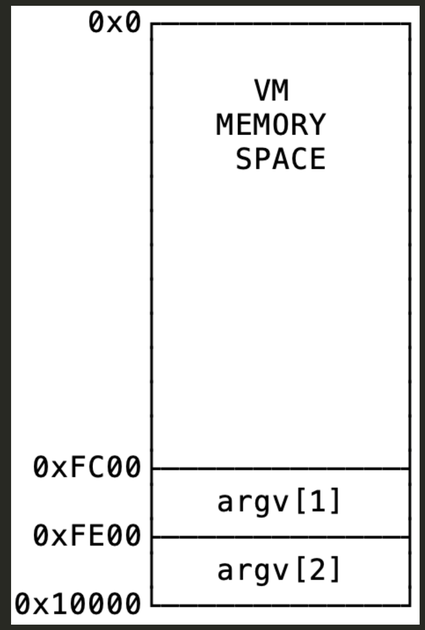
- At this point we have:
  - VM Partition.
  - VM Memory.
  - Virtual CPU.

We Love NOPs not NOPEs



We Love NOPs not NOPEs

```
if ( WHvCreateVirtualProcessor(Partition, 0, 0) >= 0 ) {

  if ( (int)sub_140001070(Partition) >= 0 ) {
    sub_140001440(&v7);
    v14 = &v7[(unsigned int)SizeInBytes - 1024];
    memmove(v14, argv[1], v10);
    v15 = &v7[(unsigned int)SizeInBytes - 512];
    memmove(v15, argv[2], v9);
    memset(ExitContext, 0, sizeof(ExitContext));
    v11 = 1;
    v13 = 0;
    while ( v11 ) {
      if ( WHvRunVirtualProcessor(Partition, 0, ExitContext, 0xE0) >= 0 ) {
```



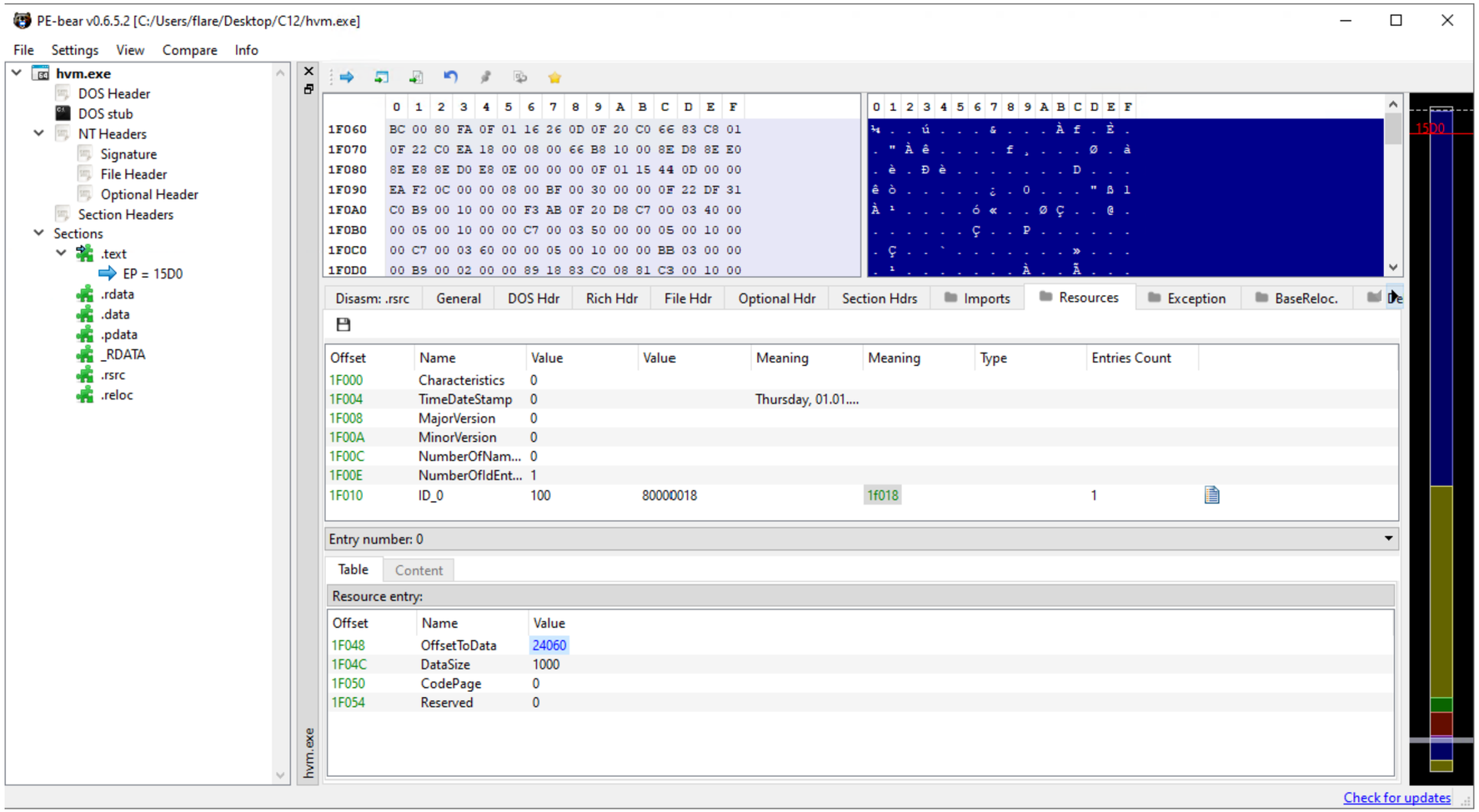
We Love NOPs not NOPEs

```
__int64 __fastcall sub_140001070(void *a1)
{
  WHV_REGISTER_NAME RegisterNames[3];
  WHV_REGISTER_VALUE RegisterValues;
  _QWORD v5[2];

  HRESULT v2 = WHvSetVirtualProcessorRegisters(a1, 0, &::RegisterNames, 0x12, &::RegisterValues);
  if ( v2 < 0 )
    return (unsigned int)v2;
  memset(&RegisterValues, 0, sizeof(RegisterValues));
  RegisterValues.Reg128.Low64 = 0;
  RegisterValues.Reg128.Dword[2] = -1;
  RegisterValues.FpControlStatus.LastFpCs = 51;
  RegisterValues.FpControlStatus.Reserved2 = RegisterValues.FpControlStatus.Reserved2 & 0xF00 | 0xA09B;
  memset(v5, 0, sizeof(v5));
  v5[0] = 4096;
  LODWORD(v5[1]) = 4096;
  RegisterNames[0] = WHvX64RegisterCs;
  return (unsigned int)WHvSetVirtualProcessorRegisters(a1, 0, RegisterNames, 1, &RegisterValues);
}
```

We Love NOPs not NOPEs

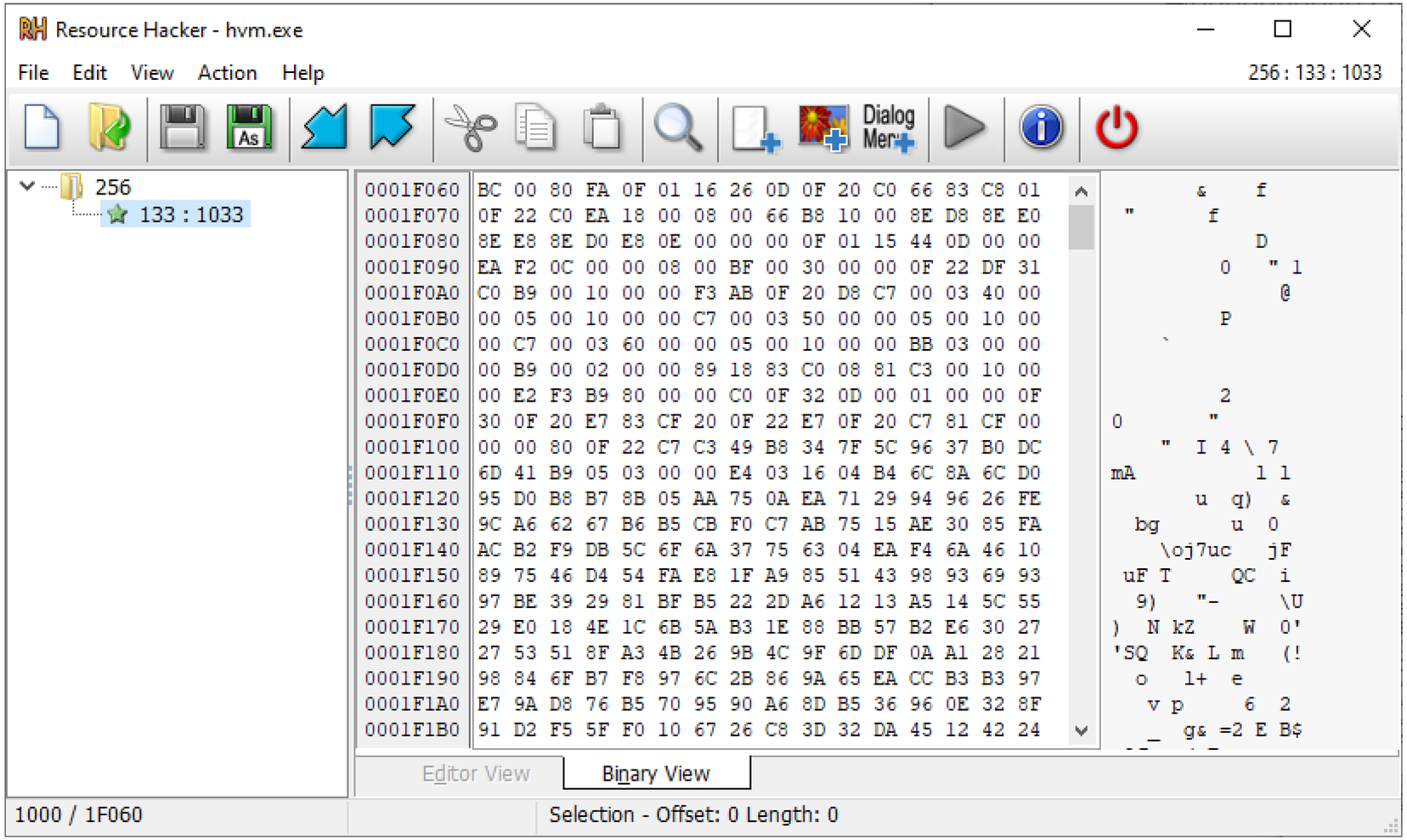
```
void *__fastcall sub_140001440(void **a1)
{
  DWORD v2;
  HMODULE ModuleHandleA;
  HRSRC ResourceA;
  HGLOBAL Resource;
  const void *v6;

  ModuleHandleA = GetModuleHandleA(0);
  ResourceA = FindResourceA(ModuleHandleA, (LPCSTR)0x85, (LPCSTR)0x100);
  Resource = LoadResource(ModuleHandleA, ResourceA);
  v2 = SizeofResource(ModuleHandleA, ResourceA);
  v6 = LockResource(Resource);
  return memmove(*a1, v6, v2);
}
```

We Love NOPs not NOPEs



dd if=hvm.exe of=extracted.bin bs=1 skip=127072 count=4096

We Love NOPs not NOPEs



We Love NOPs not NOPEs

```
if ( WHvCreateVirtualProcessor(Partition, 0, 0) >= 0 ) {
  // set virtual processor registers
  if ( (int)sub_140001070(Partition) >= 0 ) {
    // extract binary resource into the virtual CPU RAM
    sub_140001440(&guest_RAM);
    // copy the first argument into virtual CPU RAM
    v14 = (char *)guest_RAM + (unsigned int)RAM_size - 1024;
    memmove(v14, argv[1], argv1_len);
    // copy the second argument into virtual CPU RAM
    v15 = (char *)guest_RAM + (unsigned int)RAM_size - 512;
    memmove(v15, argv[2], argv2_len);
    memset(ExitContext, 0, sizeof(ExitContext));
    should_run = 1;
    v13 = 0;
    // start the virtual CPU and loop
    while ( should_run ) {
      if ( WHvRunVirtualProcessor(Partition, 0, ExitContext, 0xE0) >= 0 ) {
```

I’m a virtual CPU, you can’t see me!

I’m a Virtual CPU, you can’t see me!

I’m a Virtual CPU, you can’t see me!

- Reasonable guess that we extracted the virtual CPU code.
- We need to start disassembling it at offset 0.
  - Nothing was set to a different address.
- And start with 16-bit disassemble mode.
- Since they still boot in 8088 16-bit real mode!

I’m a Virtual CPU, you can’t see me!

```
0000000000000000                    ; Segment type: Pure code
0000000000000000                    seg000    segment byte public 'CODE' use16
0000000000000000                              assume cs:seg000
0000000000000000                              assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
0000000000000000 BC 00 80                     mov     sp, 8000h               ; set stack address
0000000000000003 FA                           cli                             ; disable interrupts
0000000000000004 0F 01 16 26 0D               lgdt    fword ptr ds:dword_D26 ; load the GDT
0000000000000009 0F 20 C0                     mov     eax, cr0                ; get ready to enter in protected mode
000000000000000C 66 83 C8 01                  or      eax, 1                  ; set bit 0 in CR0 register
0000000000000010 0F 22 C0                     mov     cr0, eax                ; set CR0 with the new bit
0000000000000013 EA 18 00 08 00               jmp     far ptr loc_98          ; clear the instruction cache with a far jmp to 0x18
```

16-bit mode

I’m a Virtual CPU, you can’t see me!

```
0000000000000018 66 B8 10 00          mov     ax, 10h
000000000000001C 8E D8                mov     ds, eax
000000000000001E                      assume ds:nothing
000000000000001E 8E E0                mov     fs, eax
0000000000000020                      assume fs:nothing
0000000000000020 8E E8                mov     gs, eax
0000000000000022                      assume gs:nothing
0000000000000022 8E D0                mov     ss, eax
0000000000000024                      assume ss:nothing
0000000000000024 E8 0E 00 00 00       call    sub_37
0000000000000029 0F 01 15 44 0D 00 00 lgdt    fword ptr ds:dword_D40+104h
0000000000000030 EA F2 0C 00 00 08 00 jmp     far ptr dword_D40+32h
```

32-bit mode

I’m a Virtual CPU, you can’t see me!

```
0000000000000037                        sub_37    proc near                   ; CODE XREF: seg000:00000024↑p
0000000000000037 BF 00 30 00 00                   mov     edi, 3000h
000000000000003C 0F 22 DF                         mov     cr3, edi            ; load CR3 with page directory location
000000000000003F 31 C0                            xor     eax, eax
0000000000000041 B9 00 10 00 00                   mov     ecx, 1000h
0000000000000046 F3 AB                            rep stosd
0000000000000048 0F 20 D8                         mov     eax, cr3
000000000000004B C7 00 03 40 00 00                mov     dword ptr [eax], 4003h
0000000000000051 05 00 10 00 00                   add     eax, 1000h
0000000000000056 C7 00 03 50 00 00                mov     dword ptr [eax], 5003h
000000000000005C 05 00 10 00 00                   add     eax, 1000h
0000000000000061 C7 00 03 60 00 00                mov     dword ptr [eax], 6003h
0000000000000067 05 00 10 00 00                   add     eax, 1000h
000000000000006C BB 03 00 00 00                   mov     ebx, 3
0000000000000071 B9 00 02 00 00                   mov     ecx, 200h
0000000000000076
0000000000000076                        loc_76:                             ; CODE XREF: sub_37+4A↓j
0000000000000076 89 18                            mov     [eax], ebx
0000000000000078 83 C0 08                         add     eax, 8
000000000000007B 81 C3 00 10 00 00                add     ebx, 1000h
0000000000000081 E2 F3                            loop    loc_76
0000000000000083 B9 80 00 00 C0                   mov     ecx, 0C0000080h     ; start configuring long mode
0000000000000088 0F 32                            rdmsr
000000000000008A 0D 00 01 00 00                   or      eax, 100h           ; set the LME flag (bit 8) in EFER MSR 0xC0000080
000000000000008F 0F 30                            wrmsr
0000000000000091 0F 20 E7                         mov     edi, cr4
0000000000000094 83 CF 20                         or      edi, 20h            ; set PAE enable bit in CR4
0000000000000097 0F 22 E7                         mov     cr4, edi
000000000000009A 0F 20 C7                         mov     edi, cr0
000000000000009D 81 CF 00 00 00 80                or      edi, 80000000h      ; enable paging bit in CR0
00000000000000A3 0F 22 C7                         mov     cr0, edi
00000000000000A6 C3                               retn
00000000000000A6                        sub_37    endp
```

I’m a Virtual CPU, you can’t see me!

- All this code is dealing with CPU transition from reset to long mode (64-bit).
- GDT, page tables, etc are irrelevant to us.
- What we really want is the entrypoint.
- Don’t forget we need to disassemble the different stages with the correct instruction size.
- IDA is confused with the far jump addresses.

I’m a Virtual CPU, you can’t see me!

```
0000000000000CF2 66 B8 10 00                  mov     ax, 10h
0000000000000CF6 8E D8                        mov     ds, eax
0000000000000CF8                              assume ds:nothing
0000000000000CF8 8E E0                        mov     fs, eax
0000000000000CFA                              assume fs:nothing
0000000000000CFA 8E E8                        mov     gs, eax
0000000000000CFC                              assume gs:nothing
0000000000000CFC 8E D0                        mov     ss, eax
0000000000000CFE                              assume ss:nothing
0000000000000CFE 48 B8 EF BE AD DE EF BE+     mov     rax, 0DEADBEEFDEADBEEFh
0000000000000CFE AD DE
0000000000000D08 E8 A5 FE FF FF               call    loc_BB2
0000000000000D0D F4                           hlt
```

64-bit mode

I’m a Virtual CPU, you can’t see me!

```
0000000000000BB2                        loc_BB2:                                ; CODE XREF: seg000:0000000000000D08↓p
0000000000000BB2 49 B8 50 B0 0B E2 FB 57+       mov     r8, 1ACF57FBE20BB050h
0000000000000BB2 CF 1A
0000000000000BBC 41 B9 1B 00 00 00              mov     r9d, 1Bh
0000000000000BC2 E4 03                          in      al, 3                   ; DMA controller, 8237A-5.
0000000000000BC2                                                                ; channel 1 current word count
0000000000000BC4 B7 06                          mov     bh, 6
0000000000000BC6 93                             xchg    eax, ebx
0000000000000BC7 57                             push    rdi
0000000000000BC8 EC                             in      al, dx
0000000000000BC9 8A FA                          mov     bh, dl
0000000000000BC9                        ; ---------------------------------------------------------------------------
0000000000000BCB C7                             db 0C7h
0000000000000BCC D2                             db 0D2h
0000000000000BCD 67                             db  67h ; g
0000000000000BCE D9                             db 0D9h
0000000000000BCF C4                             db 0C4h
0000000000000BD0 DB                             db 0DBh
0000000000000BD1 3A                             db  3Ah ; :
0000000000000BD2 DA                             db 0DAh
0000000000000BD3 89                             db  89h
0000000000000BD4 D3                             db 0D3h
```

I’m a Virtual CPU, you can’t see me!

- The code looks a bit weird (can’t disassemble all the bytes).
- The IN instruction is I/O Port related and triggers a VM EXIT.
- Host takes control (think of INT3 and debuggers).
- Used for communication between the host and guest.
- Two bytes long (wait for it).

I’m a Virtual CPU, you can’t see me!

```
while ( should_run ) {
  if ( WHvRunVirtualProcessor(Partition, 0, ExitContext, 0xE0u) >= 0 ) {
    v12 = ExitContext[0];
    if ( ExitContext[0] == 2 ) {
      sub_140001310(Partition, &v16);
      if ( (ExitContext[17] & 1) != 0 ) {
        sub_140001730(guest_RAM, v16 - 16 - v18, (unsigned int)v18, v17);
      }
      else {
        sub_140001730(guest_RAM, v16 + 2, (unsigned int)v18, v17);
      }
      sub_140001220(Partition);
    } else if ( v12 == 8 ) {
      v13 = sub_1400013E0(Partition);
      should_run = 0;
    } else {
      should_run = 0;
    }
  }
} // while
```

I’m a Virtual CPU, you can’t see me!

```
// Exit reasons
typedef enum WHV_RUN_VP_EXIT_REASON
{
    WHvRunVpExitReasonNone                      = 0x00000000,

    // Standard exits caused by operations of the virtual processor
    WHvRunVpExitReasonMemoryAccess              = 0x00000001,
    WHvRunVpExitReasonX64IoPortAccess           = 0x00000002,
    WHvRunVpExitReasonUnrecoverableException    = 0x00000004,
    WHvRunVpExitReasonInvalidVpRegisterValue    = 0x00000005,
    WHvRunVpExitReasonUnsupportedFeature        = 0x00000006,
    WHvRunVpExitReasonX64InterruptWindow        = 0x00000007,
    WHvRunVpExitReasonX64Halt                   = 0x00000008,
    WHvRunVpExitReasonX64ApicEoi                = 0x00000009,

    // Additional exits that can be configured through partition properties
    WHvRunVpExitReasonX64MsrAccess              = 0x00001000,
    WHvRunVpExitReasonX64Cpuid                  = 0x00001001,
    WHvRunVpExitReasonException                 = 0x00001002,

    // Exits caused by the host
    WHvRunVpExitReasonCanceled                  = 0x00002001
} WHV_RUN_VP_EXIT_REASON;
```

I’m a Virtual CPU, you can’t see me!

- The virtual CPU loop code deals with the I/O Port VM Exit.
- Two break conditions.
- First is the interesting one.

```
WHvDeleteVirtualProcessor(Partition, 0);
VirtualFree(guest_RAM, 0, 0x8000);
WHvDeletePartition(Partition);
if ( v13 == 0x1337 ) {
  qmemcpy(v20, &unk_1400144B0, 0x2A);
  for ( i = 0; i < 41; ++i )
    printf("%c", argv[2][i] ^ (unsigned int)v20[i]);
  printf("@flare-on.com\n");
} else {
  printf("Nope!\n");
}
```

I’m a Virtual CPU, you can’t see me!

- Smells like multiple stage encryption/obfuscation.
- Host must do something to guest RAM (since the original payload stops making sense after the VM exit).
- Guest decrypts/decodes the flag buffer?

```
__int64 __fastcall sub_1400013E0(void *a1)
{
  WHV_REGISTER_NAME RegisterNames[4];
  WHV_REGISTER_VALUE RegisterValues;
  RegisterNames[0] = WHvX64RegisterRax;
  WHvGetVirtualProcessorRegisters(a1, 0, RegisterNames, 1u, &RegisterValues);
  return RegisterValues.Reg128.Dword[0];
}
```

Ping? Pong!

Ping? Pong!

- We need to understand guest to host transition.

```
if ( ExitContext[0] == 2 )
{
  sub_140001310(Partition, &v16);
  if ( (ExitContext[17] & 1) != 0 ) {
    sub_140001730(guest_RAM, v16 - 16 - v18, (unsigned int)v18, v17);
  }
  else {
    sub_140001730(guest_RAM, v16 + 2, (unsigned int)v18, v17);
  }
  sub_140001220(Partition);
}
```

Ping? Pong!

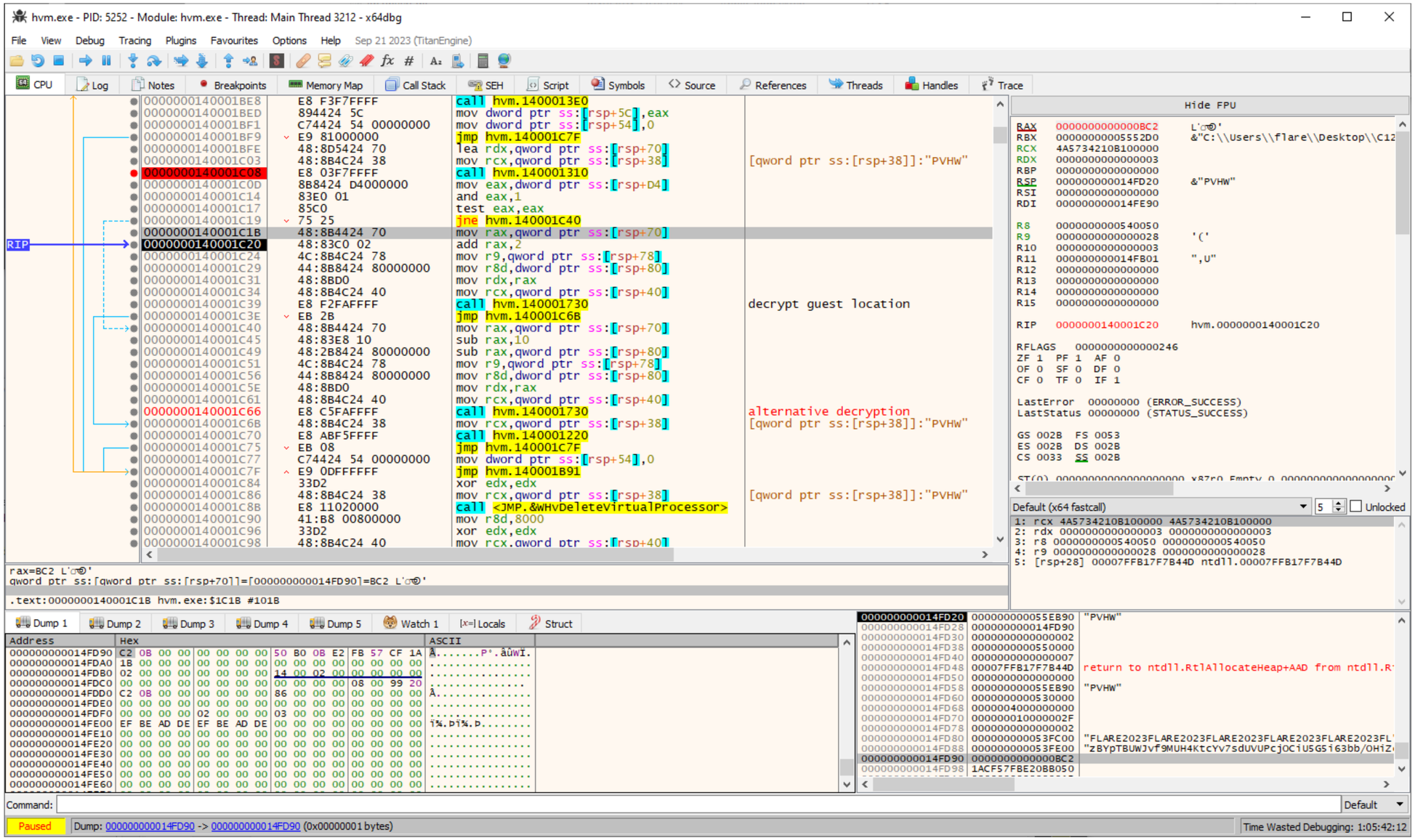
```
__int64 __fastcall sub_140001310(void *a1, UINT64 *a2)
{
  __int64 result;
  WHV_REGISTER_NAME RegisterNames[4];
  WHV_REGISTER_VALUE RegisterValues;
  UINT64 v5;
  __int64 v6;

  RegisterNames[0] = WHvX64RegisterRip;
  RegisterNames[1] = WHvX64RegisterR8;
  RegisterNames[2] = WHvX64RegisterR9;
  WHvGetVirtualProcessorRegisters(a1, 0, RegisterNames, 3, &RegisterValues);
  *a2 = RegisterValues.Reg128.Low64;
  a2[1] = v5;
  result = v6;
  a2[2] = v6;
  return result;
}
```

Ping? Pong!

- The first function just reads the contents of the VM registers.
- RIP, R8 and R9.
- A reasonable guess would be a key (R8) and size (R9).
- Next function does something with those values.

```
0000000000000BB2                        loc_BB2:
0000000000000BB2 49 B8 50 B0 0B E2 FB 57+       mov     r8, 1ACF57FBE20BB050h
0000000000000BB2 CF 1A
0000000000000BBC 41 B9 1B 00 00 00              mov     r9d, 1Bh
0000000000000BC2 E4 03                          in      al, 3
```

Ping? Pong!


hvm.exe - PID: 5252 - Module: hvm.exe - Thread: Main Thread 3212 - x64dbg
File View Debug Tracing Plugins Favourites Options Help Sep 21 2023 (TitanEngine)
CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace
RIP
0000000140001BE8 E8 F3F7FFFF call hvm.1400013E0
0000000140001BED 894424 5C mov dword ptr ss:[rsp+5C],eax
0000000140001BF1 C74424 54 00000000 mov dword ptr ss:[rsp+54],0
0000000140001BF9 E9 81000000 jmp hvm.140001C7F
0000000140001BFE 48:8D5424 70 lea rdx,qword ptr ss:[rsp+70]
0000000140001C03 48:8B4C24 38 mov rcx,qword ptr ss:[rsp+38] [qword ptr ss:[rsp+38]]:"PVHW"
0000000140001C08 E8 03F7FFFF call hvm.140001310
0000000140001C0D 8B8424 D4000000 mov eax,dword ptr ss:[rsp+D4]
0000000140001C14 83E0 01 and eax,1
0000000140001C17 85C0 test eax,eax
0000000140001C19 75 25 jne hvm.140001C40
0000000140001C1B 48:8B4424 70 mov rax,qword ptr ss:[rsp+70]
0000000140001C20 48:83C0 02 add rax,2
0000000140001C24 4C:8B4C24 78 mov r9,qword ptr ss:[rsp+78]
0000000140001C29 44:8B8424 80000000 mov r8d,dword ptr ss:[rsp+80]
0000000140001C31 48:8BD0 mov rdx,rax
0000000140001C34 48:8B4C24 40 mov rcx,qword ptr ss:[rsp+40]
0000000140001C39 E8 F2FAFFFF call hvm.140001730 decrypt guest location
0000000140001C3E EB 2B jmp hvm.140001C6B
0000000140001C40 48:8B4424 70 mov rax,qword ptr ss:[rsp+70]
0000000140001C45 48:83E8 10 sub rax,10
0000000140001C49 48:2B8424 80000000 sub rax,qword ptr ss:[rsp+80]
0000000140001C51 4C:8B4C24 78 mov r9,qword ptr ss:[rsp+78]
0000000140001C56 44:8B8424 80000000 mov r8d,dword ptr ss:[rsp+80]
0000000140001C5E 48:8BD0 mov rdx,rax
0000000140001C61 48:8B4C24 40 mov rcx,qword ptr ss:[rsp+40]
0000000140001C66 E8 C5FAFFFF call hvm.140001730 alternative decryption
0000000140001C6B 48:8B4C24 38 mov rcx,qword ptr ss:[rsp+38] [qword ptr ss:[rsp+38]]:"PVHW"
0000000140001C70 E8 ABF5FFFF call hvm.140001220
0000000140001C75 EB 08 jmp hvm.140001C7F
0000000140001C77 C74424 54 00000000 mov dword ptr ss:[rsp+54],0
0000000140001C7F E9 0DFFFFFF jmp hvm.140001B91
0000000140001C84 33D2 xor edx,edx
0000000140001C86 48:8B4C24 38 mov rcx,qword ptr ss:[rsp+38] [qword ptr ss:[rsp+38]]:"PVHW"
0000000140001C8B E8 11020000 call <JMP.&WHvDeleteVirtualProcessor>
0000000140001C90 41:B8 00800000 mov r8d,8000
0000000140001C96 33D2 xor edx,edx
0000000140001C98 48:8B4C24 40 mov rcx,qword ptr ss:[rsp+40]
Hide FPU
RAX 00000000000000BC2 L'c@'
RBX 00000000005552D0 &"C:\\Users\\flare\\Desktop\\C12
RCX 4A5734210B100000
RDX 0000000000000003
RBP 0000000000000000
RSP 000000000014FD20 &"PVHW"
RSI 0000000000000000
RDI 000000000014FE90
R8 0000000000540050
R9 0000000000000028 '('
R10 0000000000000003
R11 000000000014FB01 ",U"
R12 0000000000000000
R13 0000000000000000
R14 0000000000000000
R15 0000000000000000
RIP 0000000140001C20 hvm.0000000140001C20
RFLAGS 0000000000000246
ZF 1 PF 1 AF 0
OF 0 SF 0 DF 0
CF 0 TF 0 IF 1
LastError 00000000 (ERROR_SUCCESS)
LastStatus 00000000 (STATUS_SUCCESS)
GS 002B FS 0053
ES 002B DS 002B
CS 0033 SS 002B
Default (x64 fastcall)
1: rcx 4A5734210B100000 4A5734210B100000
2: rdx 0000000000000003 0000000000000003
3: r8 0000000000540050 0000000000540050
4: r9 0000000000000028 0000000000000028
5: [rsp+28] 00007FFB17F7B44D ntdll.00007FFB17F7B44D
rax=BC2 L'c@'
qword ptr ss:[qword ptr ss:[rsp+70]]=[000000000014FD90]=BC2 L'c@'
.text:0000000140001C1B hvm.exe:$1C1B #101B
Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 [x=] Locals Struct
Address Hex ASCII
000000000014FD90 C2 0B 00 00 00 00 00 00 50 B0 0B E2 FB 57 CF 1A Â.......P°.âûWI.
000000000014FDA0 1B 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
000000000014FDB0 02 00 00 00 00 00 00 00 14 00 02 00 00 00 00 00 ....
000000000014FDC0 00 00 00 00 00 00 00 00 00 00 00 00 08 00 99 20 ....
000000000014FDD0 C2 0B 00 00 00 00 00 00 86 00 00 00 00 00 00 00 Â...............
000000000014FDE0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
000000000014FDF0 00 00 00 00 02 00 00 00 03 00 00 00 00 00 00 00 ....
000000000014FE00 EF BE AD DE EF BE AD DE 00 00 00 00 00 00 00 00 ï¾.Þï¾.Þ........
000000000014FE10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
000000000014FE20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
000000000014FE30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
000000000014FE40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
000000000014FE50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
000000000014FE60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
000000000014FD20 0000000000055EB90 "PVHW"
000000000014FD28 000000000014FD90
000000000014FD30 0000000000000002
000000000014FD38 0000000000550000
000000000014FD40 0000000000000007
000000000014FD48 00007FFB17F7B44D return to ntdll.RtlAllocateHeap+AAD from ntdll.R
000000000014FD50 0000000000000000
000000000014FD58 0000000000055EB90 "PVHW"
000000000014FD60 0000000000530000
000000000014FD68 0000004000000000
000000000014FD70 000000010000002F
000000000014FD78 0000000000000002
000000000014FD80 000000000053FC00 "FLARE2023FLARE2023FLARE2023FLARE2023FLARE2023FL"
000000000014FD88 000000000053FE00 "zBYptBUWJvf9MUH4KtcYv7sdUVUPcjOCiU5G5i63bb/OHiZ
000000000014FD90 00000000000000BC2
000000000014FD98 1ACF57FBE20BB050
Command:
Default
Paused
Dump: 000000000014FD90 -> 000000000014FD90 (0x00000001 bytes)
Time Wasted Debugging: 1:05:42:12

Ping? Pong!

```
if ( ExitContext[0] == 2 )
{
  sub_140001310(Partition, &v16);
  if ( (ExitContext[17] & 1) != 0 ) {
    sub_140001730(guest_RAM, v16 - 16 - v18, (unsigned int)v18, v17);
  }
  else {
    sub_140001730(guest_RAM, v16 + 2, (unsigned int)v18, v17);
  }
  sub_140001220(Partition);
}
```

Ping? Pong!

Ping? Pong!

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  (...)
  __int64 v16; // [rsp+70h] [rbp-158h] BYREF
  __int64 v17; // [rsp+78h] [rbp-150h]
  __int64 v18; // [rsp+80h] [rbp-148h]
  (...)
  if ( ExitContext[0] == 2 )
  {
    sub_140001310(Partition, &v16);
    if ( (ExitContext[17] & 1) != 0 ) {
      sub_140001730(guest_RAM, v16 - 16 - v18, (unsigned int)v18, v17);
    }
    else {
      sub_140001730(guest_RAM, v16 + 2, (unsigned int)v18, v17);
    }
    sub_140001220(Partition);
  }
  (...)
}
```

- Much better when we rename the variables but remember it’s an array.

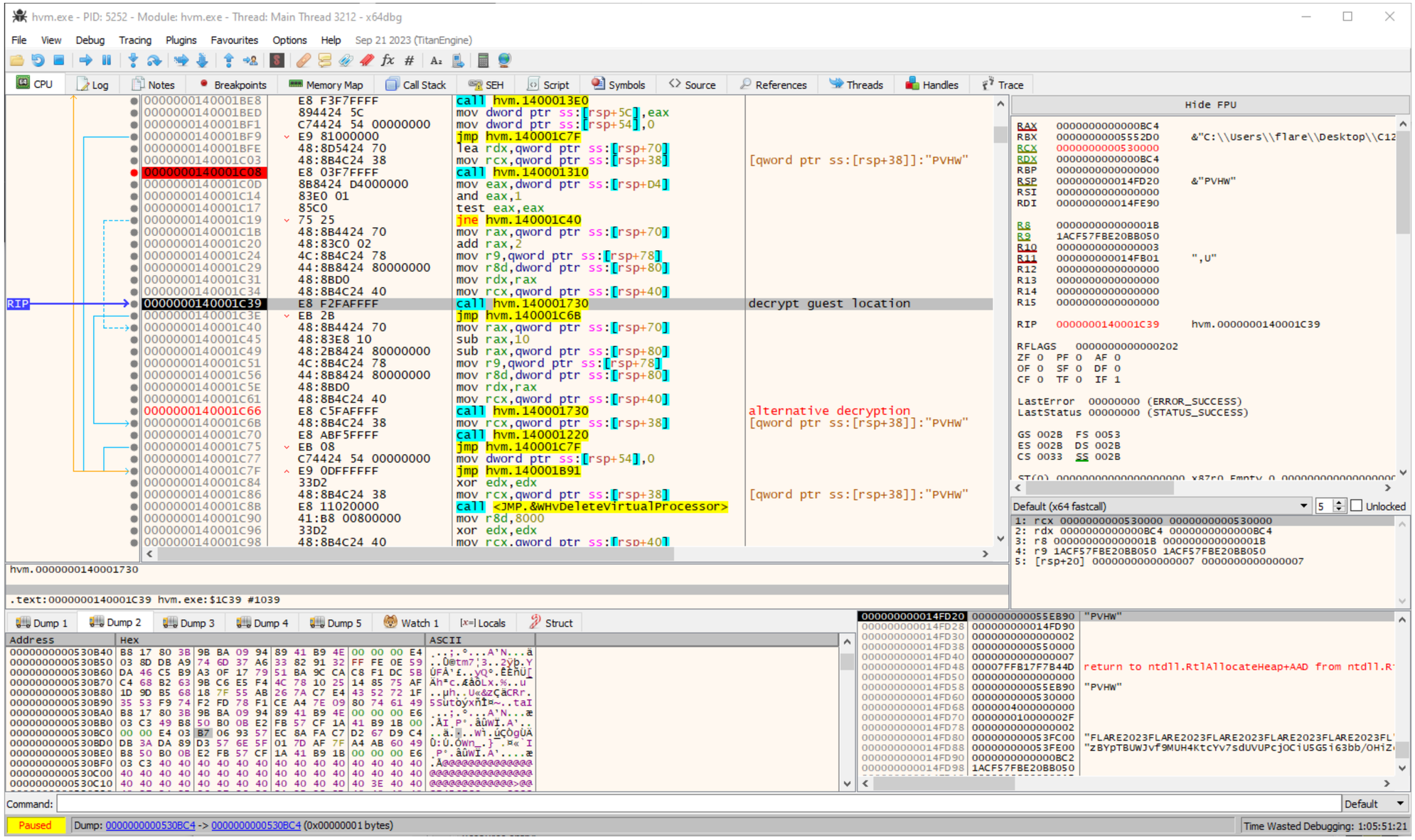
```
sub_140001310(Partition, &vm_RIP);
if ( (ExitContext[17] & 1) != 0 ) {
  sub_140001730(guest_RAM, vm_RIP - 16 - vm_R9, (unsigned int)vm_R9, vm_R8);
}
else {
  sub_140001730(guest_RAM, vm_RIP + 2, (unsigned int)vm_R9, vm_R8);
}
sub_140001220(Partition);
```

Ping? Pong!

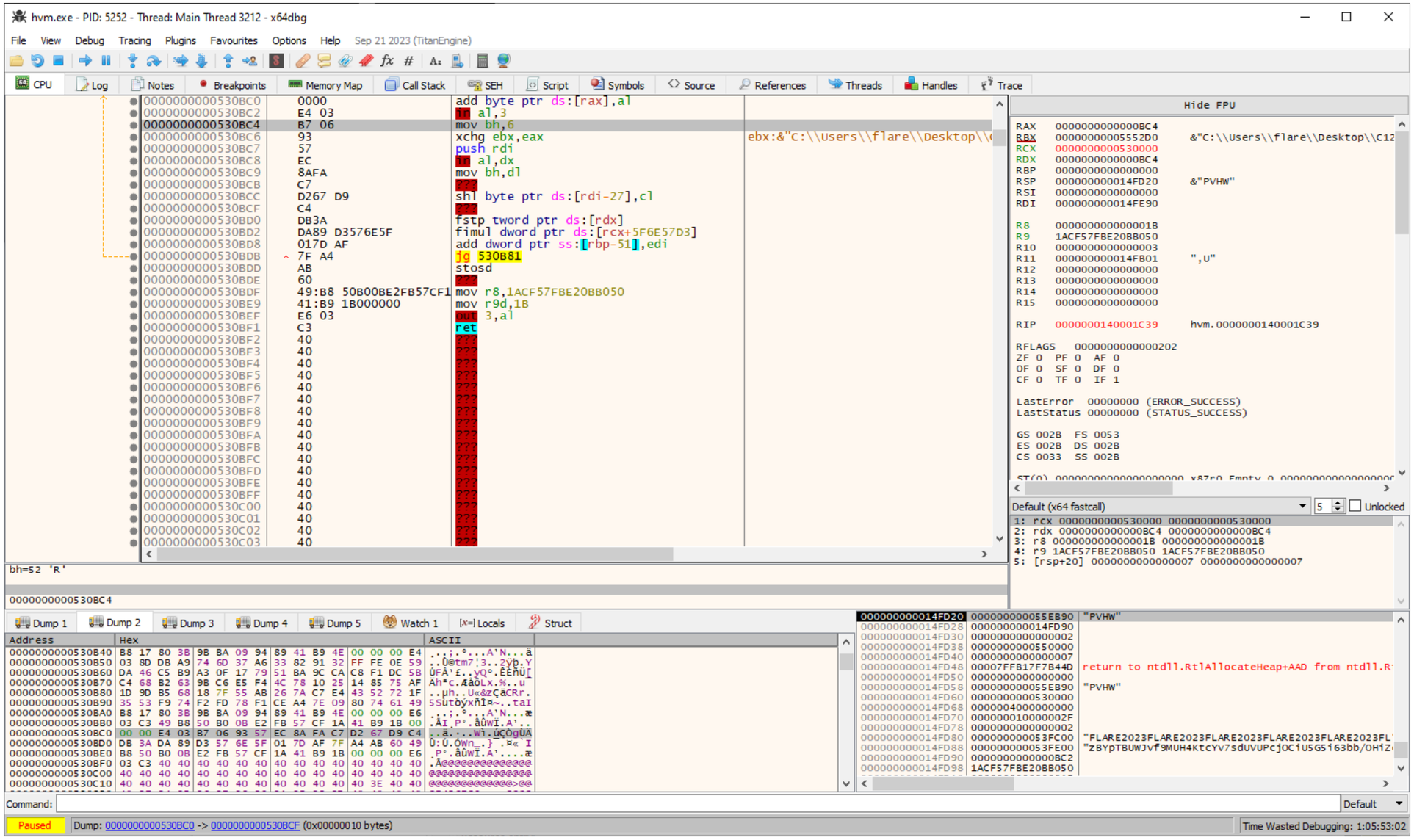
- We can easily test our hunch with a debugger.
- Avoid reversing the decryption routine!
- It’s RC4 (heavily used on Flare On).
- Breakpoint before the call to the function.
- Step over.
- Dump memory area.

Ping? Pong!

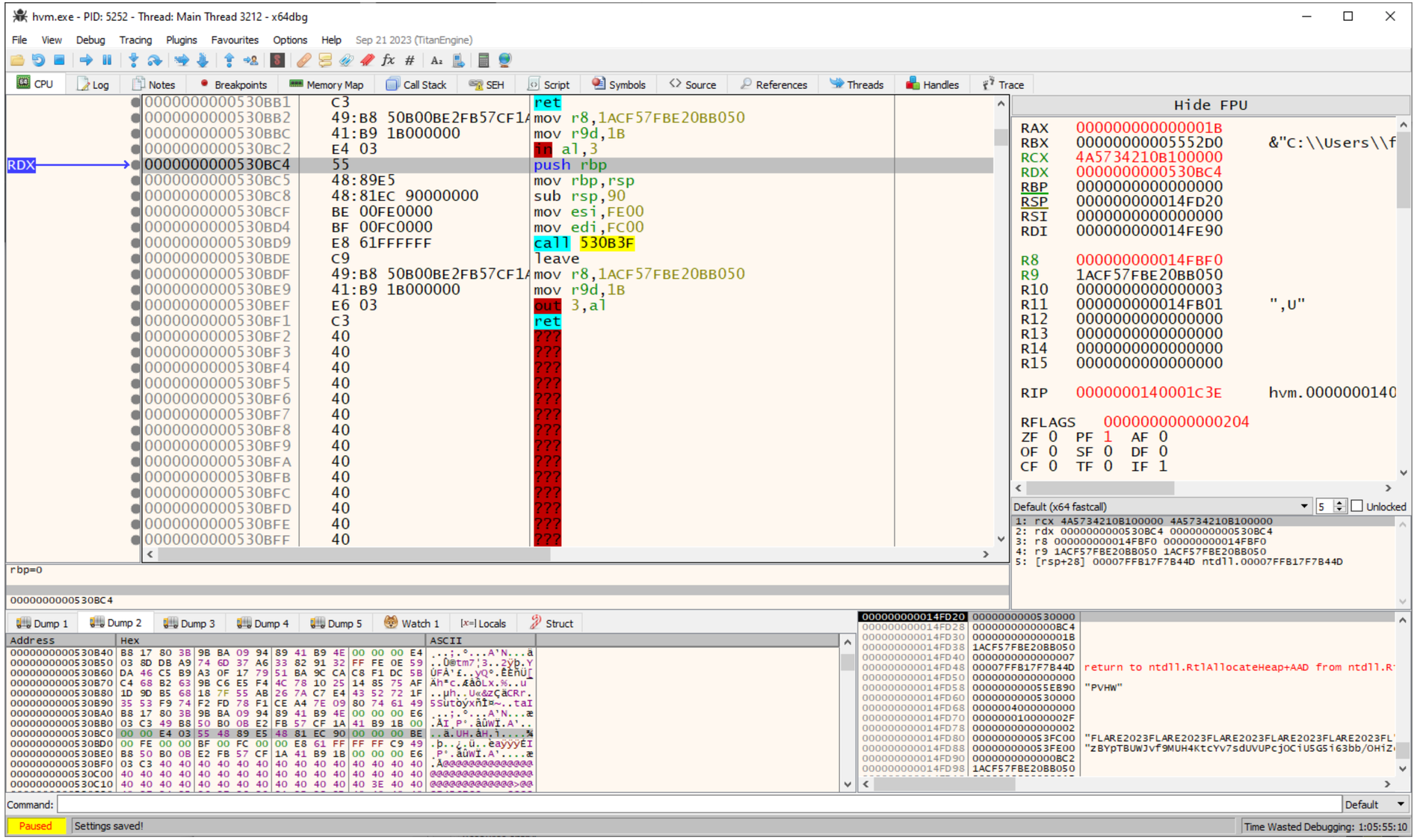
Ping? Pong!


hvm.exe - PID: 5252 - Module: hvm.exe - Thread: Main Thread 3212 - x64dbg
File View Debug Tracing Plugins Favourites Options Help Sep 21 2023 (TitanEngine)
CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace
0000000140001BE8 E8 F3F7FFFF call hvm.1400013E0
0000000140001BED 894424 5C mov dword ptr ss:[rsp+5C],eax
0000000140001BF1 C74424 54 00000000 mov dword ptr ss:[rsp+54],0
0000000140001BF9 E9 81000000 jmp hvm.140001C7F
0000000140001BFE 48:8D5424 70 lea rdx,qword ptr ss:[rsp+70]
0000000140001C03 48:8B4C24 38 mov rcx,qword ptr ss:[rsp+38] [qword ptr ss:[rsp+38]]:"PVHW"
0000000140001C08 E8 03F7FFFF call hvm.140001310
0000000140001C0D 8B8424 D4000000 mov eax,dword ptr ss:[rsp+D4]
0000000140001C14 83E0 01 and eax,1
0000000140001C17 85C0 test eax,eax
0000000140001C19 75 25 jne hvm.140001C40
0000000140001C1B 48:8B4424 70 mov rax,qword ptr ss:[rsp+70]
0000000140001C20 48:83C0 02 add rax,2
0000000140001C24 4C:8B4C24 78 mov r9,qword ptr ss:[rsp+78]
0000000140001C29 44:8B8424 80000000 mov r8d,dword ptr ss:[rsp+80]
0000000140001C31 48:8BD0 mov rdx,rax
0000000140001C34 48:8B4C24 40 mov rcx,qword ptr ss:[rsp+40]
RIP 0000000140001C39 E8 F2FAFFFF call hvm.140001730 decrypt guest location
0000000140001C3E EB 2B jmp hvm.140001C6B
0000000140001C40 48:8B4424 70 mov rax,qword ptr ss:[rsp+70]
0000000140001C45 48:83E8 10 sub rax,10
0000000140001C49 48:2B8424 80000000 sub rax,qword ptr ss:[rsp+80]
0000000140001C51 4C:8B4C24 78 mov r9,qword ptr ss:[rsp+78]
0000000140001C56 44:8B8424 80000000 mov r8d,dword ptr ss:[rsp+80]
0000000140001C5E 48:8BD0 mov rdx,rax
0000000140001C61 48:8B4C24 40 mov rcx,qword ptr ss:[rsp+40]
0000000140001C66 E8 C5FAFFFF call hvm.140001730 alternative decryption
0000000140001C6B 48:8B4C24 38 mov rcx,qword ptr ss:[rsp+38] [qword ptr ss:[rsp+38]]:"PVHW"
0000000140001C70 E8 ABF5FFFF call hvm.140001220
0000000140001C75 EB 08 jmp hvm.140001C7F
0000000140001C77 C74424 54 00000000 mov dword ptr ss:[rsp+54],0
0000000140001C7F E9 0DFFFFFF jmp hvm.140001B91
0000000140001C84 33D2 xor edx,edx
0000000140001C86 48:8B4C24 38 mov rcx,qword ptr ss:[rsp+38] [qword ptr ss:[rsp+38]]:"PVHW"
0000000140001C8B E8 11020000 call <JMP.&WHvDeleteVirtualProcessor>
0000000140001C90 41:B8 00800000 mov r8d,8000
0000000140001C96 33D2 xor edx,edx
0000000140001C98 48:8B4C24 40 mov rcx,qword ptr ss:[rsp+40]
Hide FPU
RAX 0000000000000BC4
RBX 00000000005552D0 &"C:\\Users\\flare\\Desktop\\C12
RCX 0000000000530000
RDX 0000000000000BC4
RBP 0000000000000000
RSP 000000000014FD20 &"PVHW"
RSI 0000000000000000
RDI 000000000014FE90
R8 000000000000001B
R9 1ACF57FBE20BB050
R10 0000000000000003
R11 000000000014FB01 ",U"
R12 0000000000000000
R13 0000000000000000
R14 0000000000000000
R15 0000000000000000
RIP 0000000140001C39 hvm.0000000140001C39
RFLAGS 0000000000000202
ZF 0 PF 0 AF 0
OF 0 SF 0 DF 0
CF 0 TF 0 IF 1
LastError 00000000 (ERROR_SUCCESS)
LastStatus 00000000 (STATUS_SUCCESS)
GS 002B FS 0053
ES 002B DS 002B
CS 0033 SS 002B
Default (x64 fastcall) 5 Unlocked
1: rcx 0000000000530000 0000000000530000
2: rdx 0000000000000BC4 0000000000000BC4
3: r8 000000000000001B 000000000000001B
4: r9 1ACF57FBE20BB050 1ACF57FBE20BB050
5: [rsp+20] 0000000000000007 0000000000000007
hvm.0000000140001730
.text:0000000140001C39 hvm.exe:$1C39 #1039
Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 [x=] Locals Struct
Address Hex ASCII
0000000000530B40 B8 17 80 3B 9B BA 09 94 89 41 B9 4E 00 00 00 E4 ...;.°...A'N...ä
0000000000530B50 03 8D DB A9 74 6D 37 A6 33 82 91 32 FF FE 0E 59 ..Û©tm7¦3..2ÿþ.Y
0000000000530B60 DA 46 C5 B9 A3 0F 17 79 51 BA 9C CA C8 F1 DC 5B ÚFÅ'£..yQº.ÊÈñÜ[
0000000000530B70 C4 68 B2 63 9B C6 E5 F4 4C 78 10 25 14 85 75 AF Äh²c.ÆåôLx.%..u¯
0000000000530B80 1D 9D B5 68 18 7F 55 AB 26 7A C7 E4 43 52 72 1F ..µh..U«&zÇäCRr.
0000000000530B90 35 53 F9 74 F2 FD 78 F1 CE A4 7E 09 80 74 61 49 5SùtòýxñÎ¤~.taI
0000000000530BA0 B8 17 80 3B 9B BA 09 94 89 41 B9 4E 00 00 00 E6 ..;.°...A'N...æ
0000000000530BB0 03 C3 49 B8 50 B0 0B E2 FB 57 CF 1A 41 B9 1B 00 .ÃI¸P°.âûWÏ.A'..
0000000000530BC0 00 00 E4 03 B7 06 93 57 EC 8A FA C7 D2 67 D9 C4 ..ä.·.“WìŠúÇÒgÙÄ
0000000000530BD0 DB 3A DA 89 D3 57 6E 5F 01 7D AF 7F A4 AB 60 49 Û:Ú.ÓWn_.}¯.¤«`I
0000000000530BE0 B8 50 B0 0B E2 F8 57 CF 1A 41 B9 1B 00 00 00 E6 ¸P°.âøWÏ.A'....æ
0000000000530BF0 03 C3 40 40 40 40 40 40 40 40 40 40 40 40 40 40 .Ã@@@@@@@@@@@@@@
0000000000530C00 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 @@@@@@@@@@@@@@@@
0000000000530C10 40 40 40 40 40 40 40 40 40 40 40 40 40 3E 40 40 @@@@@@@@@@@@@>@@
000000000014FD20 0000000000055EB90 "PVHW"
000000000014FD28 000000000014FD90
000000000014FD30 0000000000000002
000000000014FD38 0000000000550000
000000000014FD40 0000000000000007
000000000014FD48 00007FFB17F7B44D return to ntdll.RtlAllocateHeap+AAD from ntdll.R
000000000014FD50 0000000000000000
000000000014FD58 0000000000055EB90 "PVHW"
000000000014FD60 0000000000530000
000000000014FD68 0000004000000000
000000000014FD70 000000010000002F
000000000014FD78 0000000000000002
000000000014FD80 000000000053FC00 "FLARE2023FLARE2023FLARE2023FLARE2023FL
000000000014FD88 000000000053FE00 "zBYpTBUWJvf9MUH4KtcYv7sdUVUPCjOCiU5G5i63bb/OHiZ
000000000014FD90 00000000000000BC2
000000000014FD98 1ACF57FBE20BB050
Command: Default
Paused Dump: 0000000000530BC4 -> 0000000000530BC4 (0x00000001 bytes) Time Wasted Debugging: 1:05:51:21

Ping? Pong!


hvm.exe - PID: 5252 - Thread: Main Thread 3212 - x64dbg
File View Debug Tracing Plugins Favourites Options Help Sep 21 2023 (TitanEngine)
CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace
0000000000530BC0 0000 add byte ptr ds:[rax],al
0000000000530BC2 E4 03 in al,3
0000000000530BC4 B7 06 mov bh,6
0000000000530BC6 93 xchg ebx,eax ebx:&"C:\\Users\\flare\\Desktop\\
0000000000530BC7 57 push rdi
0000000000530BC8 EC in al,dx
0000000000530BC9 8AFA mov bh,dl
0000000000530BCB C7 ???
0000000000530BCC D267 D9 shl byte ptr ds:[rdi-27],cl
0000000000530BCF C4 ???
0000000000530BD0 DB3A fstp tword ptr ds:[rdx]
0000000000530BD2 DA89 D3576E5F fimul dword ptr ds:[rcx+5F6E57D3]
0000000000530BD8 017D AF add dword ptr ss:[rbp-51],edi
0000000000530BDB 7F A4 jg 530B81
0000000000530BDD AB stosd
0000000000530BDE 60 ???
0000000000530BDF 49:B8 50B00BE2FB57CF1 mov r8,1ACF57FBE20BB050
0000000000530BE9 41:B9 1B000000 mov r9d,1B
0000000000530BEF E6 03 out 3,al
0000000000530BF1 C3 ret
Hide FPU
RAX 0000000000000BC4
RBX 00000000005552D0 &"C:\\Users\\flare\\Desktop\\C12
RCX 0000000000530000
RDX 0000000000000BC4
RBP 0000000000000000
RSP 000000000014FD20 &"PVHW"
RSI 0000000000000000
RDI 000000000014FE90
R8 000000000000001B
R9 1ACF57FBE20BB050
R10 0000000000000003
R11 000000000014FB01 ",U"
R12 0000000000000000
R13 0000000000000000
R14 0000000000000000
R15 0000000000000000
RIP 0000000140001C39 hvm.0000000140001C39
RFLAGS 0000000000000202
ZF 0 PF 0 AF 0
OF 0 SF 0 DF 0
CF 0 TF 0 IF 1
LastError 00000000 (ERROR_SUCCESS)
LastStatus 00000000 (STATUS_SUCCESS)
GS 002B FS 0053
ES 002B DS 002B
CS 0033 SS 002B
Default (x64 fastcall) 5 Unlocked
1: rcx 0000000000530000 0000000000530000
2: rdx 0000000000000BC4 0000000000000BC4
3: r8 000000000000001B 000000000000001B
4: r9 1ACF57FBE20BB050 1ACF57FBE20BB050
5: [rsp+20] 0000000000000007 0000000000000007
bh=52 'R'
0000000000530BC4
Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 [x=] Locals Struct
Address Hex ASCII
0000000000530B40 B8 17 80 3B 9B BA 09 94 89 41 B9 4E 00 00 00 E4 ...;.°...A'N...ä
0000000000530B50 03 8D DB A9 74 6D 37 A6 33 82 91 32 FF FE 0E 59 ..Û@tm7¦3..2ÿb.Y
0000000000530B60 DA 46 C5 B9 A3 0F 17 79 51 BA 9C CA C8 F1 DC 5B ÚFÅ'£..yQ°.ÊÈñÜ[
0000000000530B70 C4 68 B2 63 9B C6 E5 F4 4C 78 10 25 14 85 75 AF Äh²c.ÆåôLx.%..u¯
0000000000530B80 1D 9D B5 68 18 7F 55 AB 26 7A C7 E4 43 52 72 1F ..µh..U«&zÇäCRr.
0000000000530B90 35 53 F9 74 F2 FD 78 F1 CE A4 7E 09 80 74 61 49 5SùtòýxñÎ¤~..taI
0000000000530BA0 B8 17 80 3B 9B BA 09 94 89 41 B9 4E 00 00 00 E6 ..;.°...A'N...æ
0000000000530BB0 03 C3 49 B8 50 B0 0B E2 FB 57 CF 1A 41 B9 1B 00 .ÃI¸P°.âûWI.A'..
0000000000530BC0 00 00 E4 03 B7 06 93 57 EC 8A FA C7 D2 67 D9 C4 ..ä.·..Wì.úÇÒgÙÄ
0000000000530BD0 DB 3A DA 89 D3 57 6E 5F 01 7D AF 7F A4 AB 60 49 Û:Ú.ÓWn_.}¯.¤«`I
0000000000530BE0 B8 50 B0 0B E2 FB 57 CF 1A 41 B9 1B 00 00 00 E6 ¸P°.âûWI.A'....æ
0000000000530BF0 03 C3 40 40 40 40 40 40 40 40 40 40 40 40 40 40 .Ã@@@@@@@@@@@@@@
0000000000530C00 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 @@@@@@@@@@@@@@@@
0000000000530C10 40 40 40 40 40 40 40 40 40 40 40 40 40 3E 40 40 @@@@@@@@@@@@>@@
000000000014FD20 0000000000055EB90 "PVHW"
000000000014FD28 000000000014FD90
000000000014FD30 0000000000000002
000000000014FD38 0000000000550000
000000000014FD40 0000000000000007
000000000014FD48 00007FFB17F7B44D return to ntdll.RtlAllocateHeap+AAD from ntdll.R
000000000014FD50 0000000000000000
000000000014FD58 0000000000055EB90 "PVHW"
000000000014FD60 0000000000530000
000000000014FD68 0000004000000000
000000000014FD70 000000010000002F
000000000014FD78 0000000000000002
000000000014FD80 0000000000053FC00 "FLARE2023FLARE2023FLARE2023FLARE2023FLARE2023FL
000000000014FD88 0000000000053FE00 "zBYpTBUWJvf9MUH4KtcYv7sdUVUPcjOCiU5G5i63bb/OHiZ
000000000014FD90 0000000000000BC2
000000000014FD98 1ACF57FBE20BB050
Command: Default
Paused Dump: 0000000000530BC0 -> 0000000000530BCF (0x00000010 bytes) Time Wasted Debugging: 1:05:53:02

Ping? Pong!


hvm.exe - PID: 5252 - Thread: Main Thread 3212 - x64dbg
File View Debug Tracing Plugins Favourites Options Help Sep 21 2023 (TitanEngine)
CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads Handles Trace
RDX
0000000000530BB1 C3 ret
0000000000530BB2 49:B8 50B00BE2FB57CF1A mov r8,1ACF57FBE20BB050
0000000000530BBC 41:B9 1B000000 mov r9d,1B
0000000000530BC2 E4 03 in al,3
0000000000530BC4 55 push rbp
0000000000530BC5 48:89E5 mov rbp,rsp
0000000000530BC8 48:81EC 90000000 sub rsp,90
0000000000530BCF BE 00FE0000 mov esi,FE00
0000000000530BD4 BF 00FC0000 mov edi,FC00
0000000000530BD9 E8 61FFFFFF call 530B3F
0000000000530BDE C9 leave
0000000000530BDF 49:B8 50B00BE2FB57CF1A mov r8,1ACF57FBE20BB050
0000000000530BE9 41:B9 1B000000 mov r9d,1B
0000000000530BEF E6 03 out 3,al
0000000000530BF1 C3 ret
Hide FPU
RAX 000000000000001B
RBX 00000000005552D0 &"C:\\Users\\f
RCX 4A5734210B100000
RDX 0000000000530BC4
RBP 0000000000000000
RSP 000000000014FD20
RSI 0000000000000000
RDI 000000000014FE90
R8 000000000014FBF0
R9 1ACF57FBE20BB050
R10 0000000000000003
R11 000000000014FB01 ",U"
R12 0000000000000000
R13 0000000000000000
R14 0000000000000000
R15 0000000000000000
RIP 0000000140001C3E hvm.0000000140
RFLAGS 0000000000000204
ZF 0 PF 1 AF 0
OF 0 SF 0 DF 0
CF 0 TF 0 IF 1
Default (x64 fastcall)
1: rcx 4A5734210B100000 4A5734210B100000
2: rdx 0000000000530BC4 0000000000530BC4
3: r8 000000000014FBF0 000000000014FBF0
4: r9 1ACF57FBE20BB050 1ACF57FBE20BB050
5: [rsp+28] 00007FFB17F7B44D ntdll.00007FFB17F7B44D
rbp=0
0000000000530BC4
Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 [x=] Locals Struct
Address Hex ASCII
000000000014FD20 0000000000530000
000000000014FD28 0000000000000BC4
000000000014FD30 000000000000001B
000000000014FD38 1ACF57FBE20BB050
000000000014FD40 0000000000000007
000000000014FD48 00007FFB17F7B44D return to ntdll.RtlAllocateHeap+AAD from ntdll.R
000000000014FD50 0000000000000000
000000000014FD58 000000000055EB90 "PVHW"
000000000014FD60 0000000000530000
000000000014FD68 0000004000000000
000000000014FD70 000000010000002F
000000000014FD78 0000000000000002
000000000014FD80 000000000053FC00 "FLARE2023FLARE2023FLARE2023FLARE2023FL
000000000014FD88 000000000053FE00 "zBYpTBUWJvf9MUH4KtcYv7sdUvUPcjOCiUSG5i63bb/OHiZ
000000000014FD90 0000000000000BC2
000000000014FD98 1ACF57FBE20BB050
Command:
Default
Paused
Settings saved!
Time Wasted Debugging: 1:05:55:10

Ping? Pong!

- Our hunch is correct, there is decryption.
- Then execution resumes after the VM exit with RIP update.

```
HRESULT __fastcall sub_140001220(void *a1)
{
  WHV_REGISTER_NAME RegisterNames[4];
  WHV_REGISTER_VALUE RegisterValues;

  RegisterNames[0] = WHvX64RegisterRip;
  WHvGetVirtualProcessorRegisters(a1, 0, RegisterNames, 1u, &RegisterValues);
  RegisterValues.Reg128.Low64 += 2;
  return WHvSetVirtualProcessorRegisters(a1, 0, RegisterNames, 1u, &RegisterValues);
}
```

Ping? Pong!

```
while ( should_run ) {
  if ( WHvRunVirtualProcessor(Partition, 0, ExitContext, 0xE0u) >= 0 ) {
    v12 = ExitContext[0];
    if ( ExitContext[0] == 2 ) { // IO Port VM Exit
      fg_read_VM_registers(Partition, &vm_RIP);
      if ( (ExitContext[17] & 1) != 0 ) {
        fg_decrypt_guest_RAM(guest_RAM, vm_RIP - 16 - vm_R9, (unsigned int)vm_R9, vm_R8);
      } else {
        fg_decrypt_guest_RAM(guest_RAM, vm_RIP + 2, (unsigned int)vm_R9, vm_R8);
      }
      fg_advance_VM_RIP(Partition);
    } else if ( v12 == 8 ) {// Halt VM Exit
      result = fg_read_VM_RAX(Partition);
      should_run = 0;
    } else {                // Anything else VM Exit
      should_run = 0;
    }
  }
}
```

Ping? Pong!

- Pretty sure it’s multi stage decryption (that’s why the loop).
- Until the VM halts and a result is read.
- If it’s the right result, “decrypt” and display the flag.
- Reasonable to test if we can dump everything at once.
- Breakpoint at the halt address and dump memory.

Ping? Pong!

Ping? Pong!

```
0000000000000BB2                        fg_00_entrypoint proc near              ; CODE XREF: seg000:0000000000000D08↓p
0000000000000BB2 49 B8 50 B0 0B E2 FB 57+       mov     r8, 1ACF57FBE20BB050h ; entrypoint
0000000000000BB2 CF 1A
0000000000000BBC 41 B9 1B 00 00 00              mov     r9d, 1Bh
0000000000000BC2 E4 03                          in      al, 3                   ; DMA controller, 8237A-5.
0000000000000BC2                                                                ; channel 1 current word count
0000000000000BC4 55                             push    rbp
0000000000000BC5 48 89 E5                       mov     rbp, rsp
0000000000000BC8 48 81 EC 90 00 00 00           sub     rsp, 90h
0000000000000BCF BE 00 FE 00 00                 mov     esi, 0FE00h
0000000000000BD4 BF 00 FC 00 00                 mov     edi, 0FC00h
0000000000000BD9 E8 61 FF FF FF                 call    fg_01_verify_args
0000000000000BDE C9                             leave
0000000000000BDF 49 B8 50 B0 0B E2 FB 57+       mov     r8, 1ACF57FBE20BB050h
0000000000000BDF CF 1A
0000000000000BE9 41 B9 1B 00 00 00              mov     r9d, 1Bh
0000000000000BEF E6 03                          out     3, al                   ; DMA controller, 8237A-5.
0000000000000BEF                                                                ; channel 1 base address and word count
0000000000000BF1 C3                             retn
0000000000000BF1                        fg_00_entrypoint endp
```

Ping? Pong!

- The OUT VM Exit prologue.
- It encrypts again the block.

```
if ( ExitContext[0] == 2 )// IO Port VM Exit
{
  fg_read_VM_registers(Partition, &vm_RIP);
  if ( (ExitContext[17] & 1) != 0 ) {
    // encrypts
    fg_decrypt_guest_RAM(guest_RAM, vm_RIP - 16 - vm_R9, (unsigned int)vm_R9, vm_R8);
  }
  else {
    // decrypts
    fg_decrypt_guest_RAM(guest_RAM, vm_RIP + 2, (unsigned int)vm_R9, vm_R8);
  }
  fg_advance_VM_RIP(Partition);
}
```

- Solutions:
  - Reverse or reuse the encryption function and manually/ script decrypt each stage.
  - We know the format: MOV R8, MOV R9, IN/OUT.
  - Or manually trace each stage and dump it.
  - They don’t overlap.
    - Few stages so I copied and stitched everything.

Are we there yet?

Are we there yet?

Are we there yet?

```
0000000000000BB2                        ; __int64 __fastcall fg_00_entrypoint(__int64, __int64, __int64, __int64)
0000000000000BB2                        fg_00_entrypoint proc near              ; CODE XREF: seg000:0000000000000D08↓p
0000000000000BB2 49 B8 50 B0 0B E2 FB 57 CF 1A          mov     r8, 1ACF57FBE20BB050h ; decryption key
0000000000000BBC 41 B9 1B 00 00 00                      mov     r9d, 1Bh        ; decryption size
0000000000000BBC                                                                ; 0xBDF - 0xBC4 = 0x1B
0000000000000BC2 E4 03                                  in      al, 3           ; ask host to decrypt
0000000000000BC4 55                                     push    rbp
0000000000000BC5 48 89 E5                               mov     rbp, rsp
0000000000000BC8 48 81 EC 90 00 00 00                   sub     rsp, 90h
0000000000000BCF BE 00 FE 00 00                         mov     esi, 0FE00h     ; argv[1]
0000000000000BD4 BF 00 FC 00 00                         mov     edi, 0FC00h     ; argv[2]
0000000000000BD9 E8 61 FF FF FF                         call    fg_01_verify_args ; f(argv[1], argv[2])
0000000000000BDE C9                                     leave
0000000000000BDF 49 B8 50 B0 0B E2 FB 57 CF 1A          mov     r8, 1ACF57FBE20BB050h
0000000000000BE9 41 B9 1B 00 00 00                      mov     r9d, 1Bh
0000000000000BEF E6 03                                  out     3, al           ; ask host to encrypt
0000000000000BF1 C3                                     retn
0000000000000BF1                        fg_00_entrypoint endp
```

Are we there yet?

```
0000000000000B3F                        ; __int64 __fastcall fg_01_verify_args(unsigned int *, char *)
0000000000000B3F                        fg_01_verify_args proc near             ; CODE XREF: fg_00_entrypoint+27↓p
0000000000000B3F 49 B8 17 80 3B 9B BA 09 94 89          mov     r8, 899409BA9B3B8017h
0000000000000B49 41 B9 4E 00 00 00                      mov     r9d, 4Eh ; 'N'
0000000000000B4F E4 03                                  in      al, 3
0000000000000B51 55                                     push    rbp
0000000000000B52 48 89 E5                               mov     rbp, rsp
0000000000000B55 48 83 EC 20                            sub     rsp, 20h
0000000000000B59 48 89 7D E8                            mov     [rbp-18h], rdi          ; argv[1]
0000000000000B5D 48 89 75 E0                            mov     [rbp-20h], rsi          ; argv[2]
0000000000000B61 48 8B 45 E8                            mov     rax, [rbp-18h]
0000000000000B65 48 89 C7                               mov     rdi, rax                ; argv[1]
0000000000000B68 E8 AB FD FF FF                         call    fg_02_verify_1st_arg_0x92A
0000000000000B6D 89 45 FC                               mov     [rbp-4], eax            ; store return value #1
0000000000000B70 48 8B 55 E0                            mov     rdx, [rbp-20h]
0000000000000B74 48 8B 45 E8                            mov     rax, [rbp-18h]
0000000000000B78 48 89 D6                               mov     rsi, rdx                ; argv[2]
0000000000000B7B 48 89 C7                               mov     rdi, rax                ; argv[1]
0000000000000B7E E8 DF FE FF FF                         call    fg_03_verify_2nd_arg_0xA74
0000000000000B83 89 45 F8                               mov     [rbp-8], eax            ; store return value #2
0000000000000B86 83 7D FC 24                            cmp     dword ptr [rbp-4], 24h ; '$' ; return value of the first argument comparison
0000000000000B8A 75 0D                                  jnz     short loc_B99
0000000000000B8C 83 7D F8 01                            cmp     dword ptr [rbp-8], 1
0000000000000B90 75 07                                  jnz     short loc_B99           ; 2nd arg must return 1
0000000000000B92 B8 37 13 00 00                         mov     eax, 1337h              ; SUCCESS!!!!
0000000000000B97 EB 05                                  jmp     short locret_B9E
0000000000000B99                        ; ---------------------------------------------------------------------------
0000000000000B99
0000000000000B99                        loc_B99:                                ; CODE XREF: fg_01_verify_args+4B↑j
0000000000000B99                                                                ; fg_01_verify_args+51↑j
0000000000000B99 B8 00 00 00 00                         mov     eax, 0                  ; FAIL!
0000000000000B9E
0000000000000B9E                        locret_B9E:                             ; CODE XREF: fg_01_verify_args+58↑j
0000000000000B9E C9                                     leave
0000000000000B9F 49 B8 17 80 3B 9B BA 09 94 89          mov     r8, 899409BA9B3B8017h
0000000000000BA9 41 B9 4E 00 00 00                      mov     r9d, 4Eh ; 'N'
0000000000000BAF E6 03                                  out     3, al
0000000000000BB1 C3                                     retn
0000000000000BB1                        fg_01_verify_args endp
```

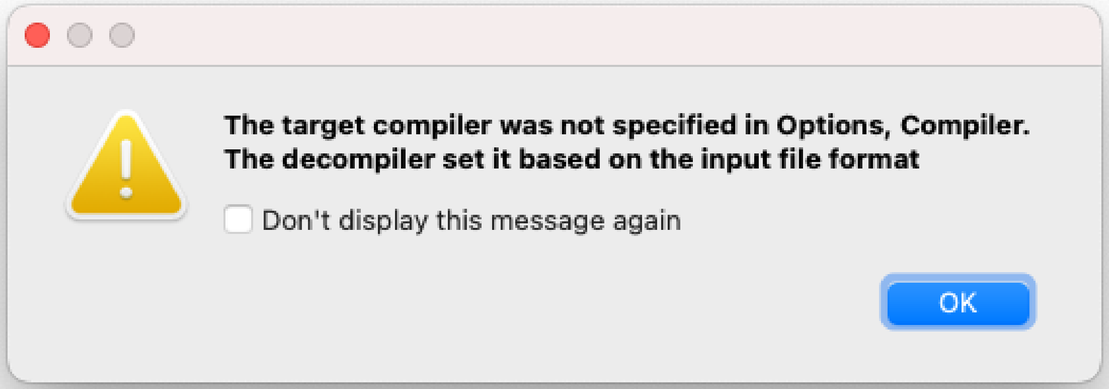
Are we there yet?

Are we there yet?

```
__int64 __fastcall fg_O2_verify_1st_arg_0x92A(char *a1)
{
  (...)
  __inbyte(3u);
  v10 = v1;
  strcpy(v6, "*#37([@AF+ .  _YB@3!-=7W][C59,>*@U_Zpsumloremips");
  strcpy(v5, "mipsumloremipsumloremips");
  v7 = fg_strlen_0x853(a1);
  v9 = 0;
  for ( i = 0; i < v7; ++i ) {
    if ( (v4[i] ^ *(_BYTE *)(i + v3)) == v6[i] )
      ++v9;
  }
  result = v9;
  __outbyte(3u, v9);
  return result;
}
```

Are we there yet?

- The decompiler is wrong on this function.
- Lost some silly time here.
- Too tired already, didn’t care about the warnings.

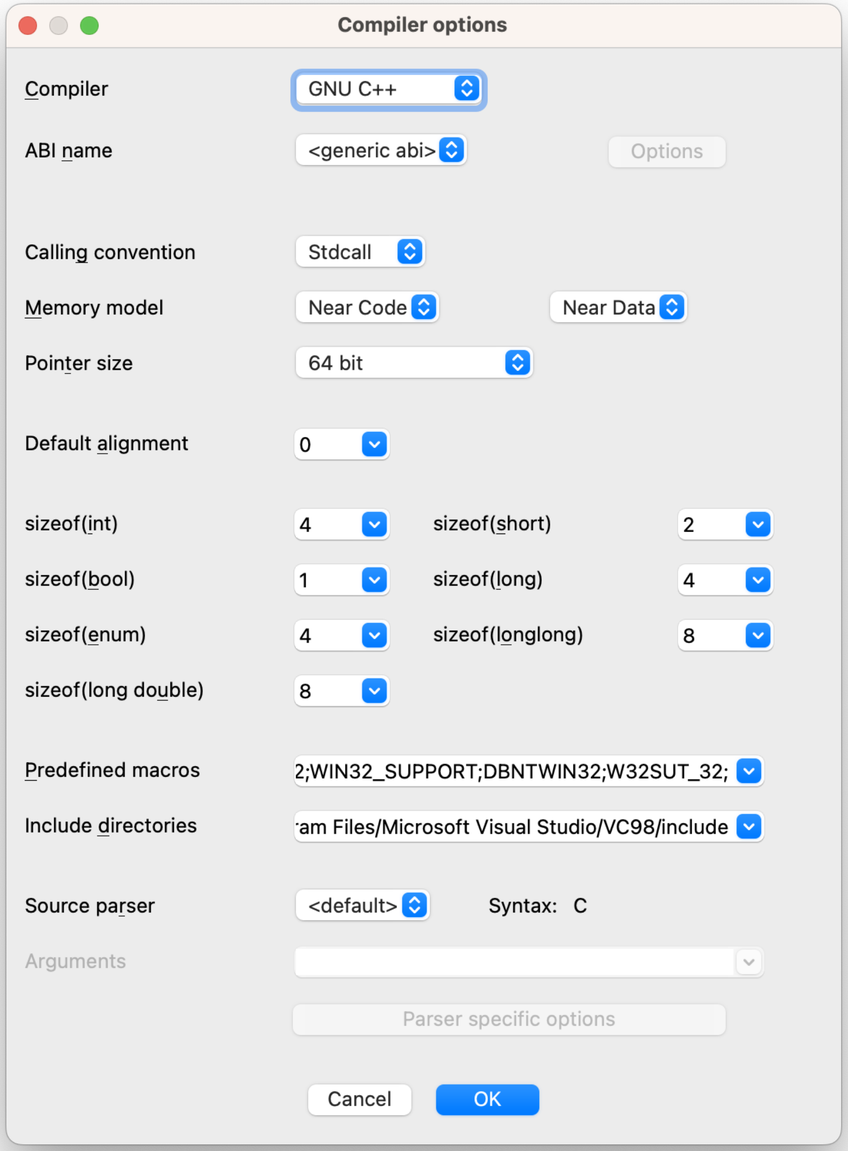


Are we there yet?

- Different call conventions (and compilers?) used in the binary and the VM payload.
- Binary uses Microsoft X64 calling convention: RCX RDX R8 R9 STACK.
- Payload is using System V AMD64 ABI: RDI RSI RDX RCX R8 R9 STACK.

```
0000000000000BCF BE 00 FE 00 00          mov     esi, 0FE00h             ; argv[1]
0000000000000BD4 BF 00 FC 00 00          mov     edi, 0FC00h             ; argv[2]
0000000000000BD9 E8 61 FF FF FF          call    fg_01_verify_args       ; f(argv[1], argv[2])
```

Are we there yet?



Are we there yet?

```
__int64 __fastcall sub_92A(__int64 a1)
{
  __int64 result;
  char v2[64];
  _DWORD v3[14];
  int i;
  unsigned int v5;

  strcpy((char *)v3, "*#37([@AF+ .  _YB@3!-=7W][C59,>*@U_Zpsumloremips");
  strcpy(v2, "loremipsumloremipsumloremipsumloremipsumloremipsumloremips");
  v3[13] = ((__int64 (__fastcall *)(__int64))loc_841)(a1);
  v5 = 0;
  for ( i = 0; i < v3[13]; ++i )
  {
    if ( ((unsigned __int8)v2[i] ^ *(_BYTE *)(i + a1)) == *((_BYTE *)v3 + i) )
      ++v5;
  }
  result = v5;
  __outbyte(3u, v5);
  return result;
}
```

Are we there yet?

- Decompilation is much better now.
- Return value must be 0x24 (36 chars).
- Easy to extract the valid string (simple XOR).

```
mov     rdi, rax                ; argv[1]
call    fg_02_verify_1st_arg_0x92A
mov     [rbp-4], eax            ; store return value #1
mov     rdx, [rbp-20h]
mov     rax, [rbp-18h]
mov     rsi, rdx                ; argv[2]
mov     rdi, rax                ; argv[1]
call    fg_03_verify_2nd_arg_0xA74
mov     [rbp-8], eax            ; store return value #2
cmp     dword ptr [rbp-4], 24h ; '$' ; return value of the first argument comparison
jnz     short loc_B99
cmp     dword ptr [rbp-8], 1
jnz     short loc_B99           ; 2nd arg must return 1
mov     eax, 1337h              ; SUCCESS!!!!
jmp     short locret_B9E
```

Are we there yet?

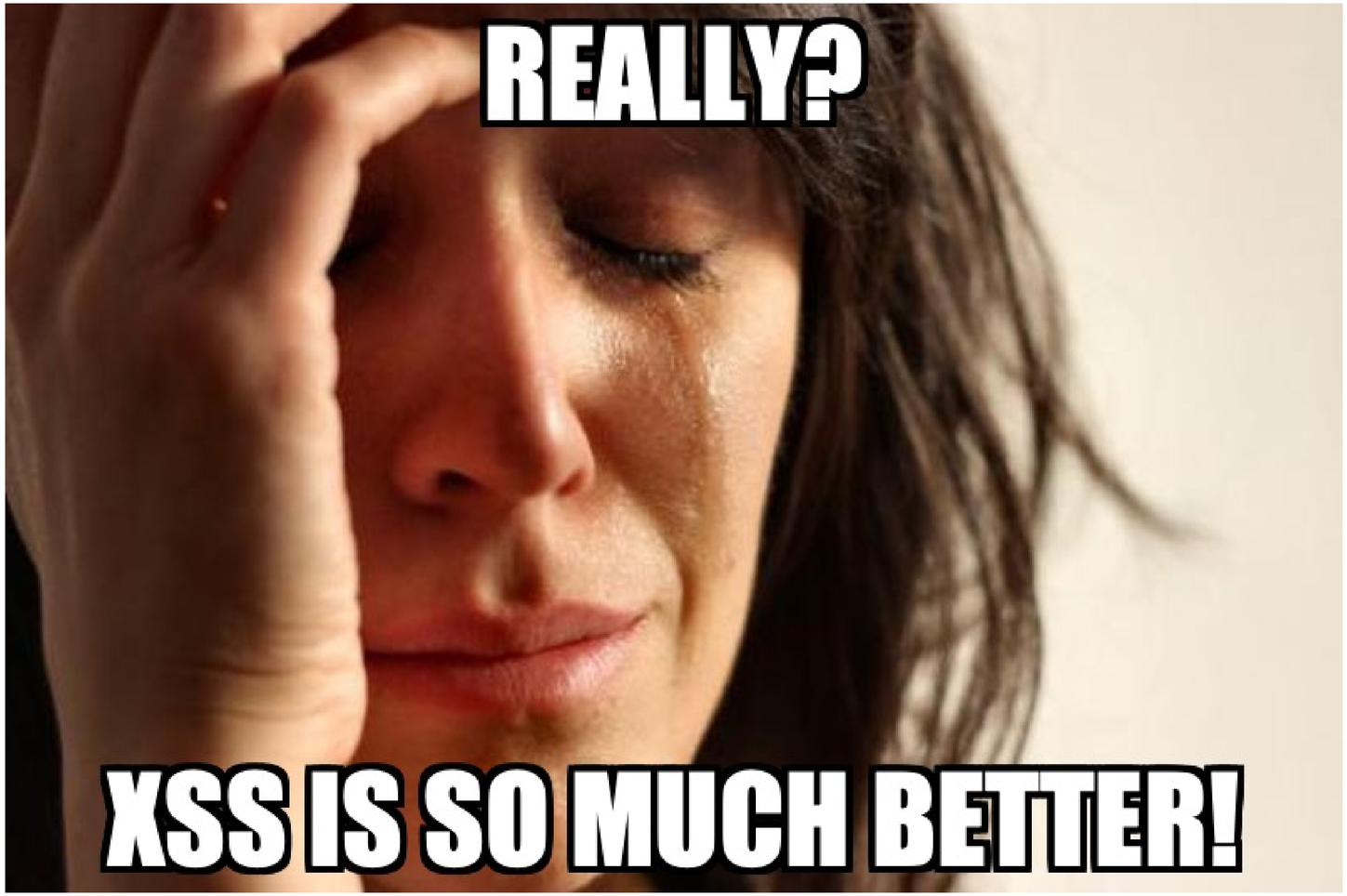
```c
#include <stdio.h>
#include <string.h>
#include <stdint.h>

int main(int argc, char *argv[]) {
    char v3[64] = {0};
    char v4[56] = {0};
    strcpy(v3, "loremipsumloremipsumloremipsumloremipsumloremips");
    strcpy(v4, "*#37([@AF+ .  _YB@3!-=7W][C59,>*@U_Zpsumloremips");

    for (int i = 0; i < 0x24; ++i) {
        int left = (int64_t)((char)v3[i]);
        int right = (int64_t)((char)v4[i]);
        int a = left ^ right;
        printf("%c", a);
    }
    printf("\n");
}

% ./getarg1
FLARE2023FLARE2023FLARE2023FLARE2023
```

Are we there yet?

```c
#include <stdio.h>
#include <string.h>
#include <stdint.h>

int main(int argc, char *argv[])
{
    char a1[] = "FLARE2023FLARE2023FLARE2023FLARE2023";
    char v2[64];
    int v3[14];

    strcpy((char *)v3, "*#37([@AF+ .  _YB@3!-=7W][C59,>*@U_Zpsumloremips");
    strcpy(v2, "loremipsumloremipsumloremipsumloremipsumloremips");
    v3[13] = strlen(a1);
    unsigned int result = 0;
    for (int i = 0; i < v3[13]; ++i ) {
        if ( ((uint8_t)v2[i] ^ (uint8_t)a1[i]) == *((char *)v3 + i) )
            ++result;
    }
    printf("0x%x\n", result);
}

% ./verifyarg1
0x24
```

One more thing...

One more thing



One more thing

- Flag is just a XOR between argv[2] and a fixed array.
- Array contents are unmodified (host data).

```
if ( result == 0x1337 )
{
  qmemcpy(v20, &unk_1400144B0, 0x2Aui64);
  for ( i = 0; i < 0x29; ++i ) {
    printf("%c", argv[2][i] ^ (unsigned int)v20[i]);
  }
  printf("@flare-on.com\n");
}
```

One more thing

```
_BOOL8 __fastcall fg_03_verify_2nd_arg_0xA74(unsigned int *argv1, char *argv2)
{
  __int64 v2;
  _BOOL8 result;
  char buf[60];
  int buf_len;

  __inbyte(3u);
  __int64 v7 = v2;
  memset(buf, 0, 49);                                   // 8 < strlen(argv[1]) < 48
                                                        // 24 < strlen(argv[2]) < 65
  int arg2_len = fg_strlen(argv2);
  buf_len = fg_decode_base64(argv2, arg2_len, buf);
  if ( (buf_len & 7) != 0 ) {
    result = 0LL;
  } else {
    fg_decrypt(buf, buf_len, *argv1);// decrypt with salsa20 and something else
    result = fg_verify_decryption((char *)argv1, buf, 48);// verify the result
  }
  __outbyte(3u, result);
  return result;
}
```

One more thing

- Easy to verify the base64 decode function.
- Let’s give a look to the decryption verification function.
- It returns the value 1 that we want.

```
/* Base64 encoder/decoder. Originally Apache file ap_base64.c
 */

/* aaaack but it's fast and const should make it shared text page. */
static const unsigned char pr2six[256] =
{
    /* ASCII table */
    64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64,
    64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 64,
    64, 64, 64, 64, 64, 64, 64, 64, 64, 64, 62, 64, 64, 64, 63,
    52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 64, 64, 64, 64, 64, 64,
    64,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
```

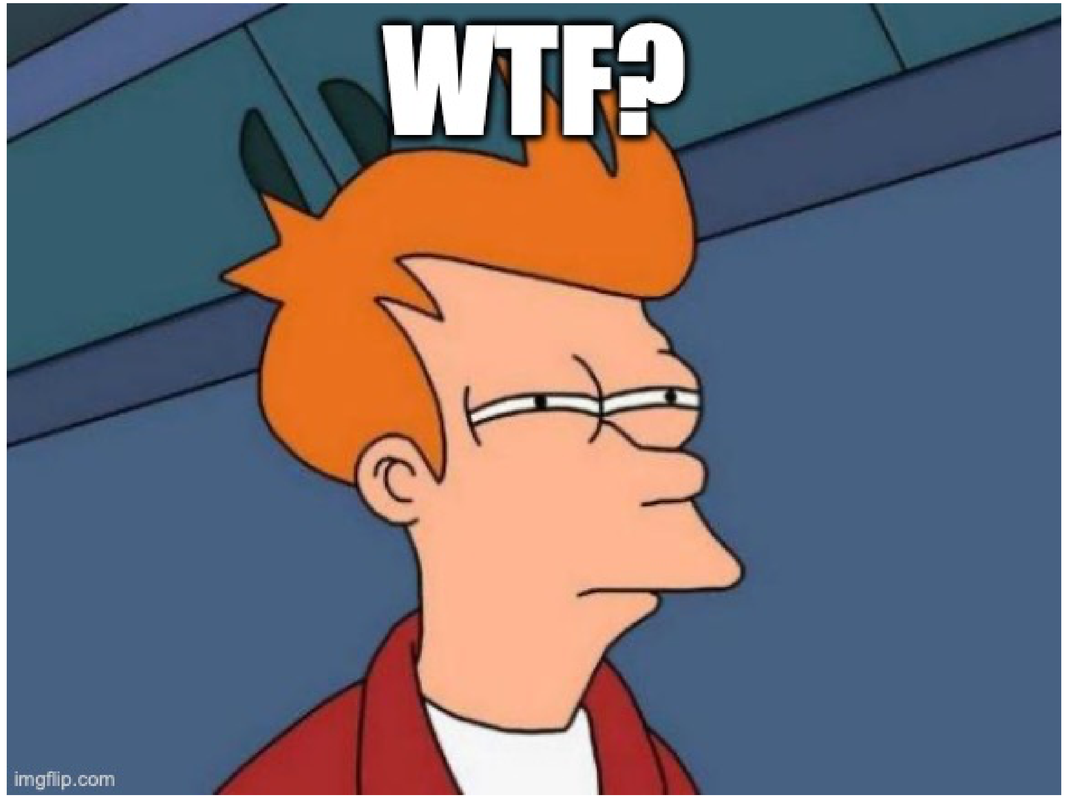
```
fg_00_entrypoint endp
qword_BF2 dq 4040404040404040h          ; DATA XREF: fg_decode_base64+BD↑o
                                        ; fg_decode_base64+108↑o ...
          dq 4040404040404040h
          dq 4040404040404040h
          dq 4040404040404040h
          dq 4040404040404040h
          dq 3F4040403E404040h
          dq 3B3A393837363534h
          dq 4040404040403D3Ch
          db  40h ; @
          db    0
```

One more thing

```
_BOOL8 __fastcall fg_verify_decryption(char *argv1, char *decoded_buf, int len)
{
  _BOOL8 result;
  __int64 v4;

  __inbyte(3u);
  *(&v4 - 3) = (__int64)argv1;
  *(&v4 - 4) = (__int64)decoded_buf;
  *((_DWORD *)&v4 - 9) = len;
  *((_DWORD *)&v4 - 1) = 0;
  for ( *((_DWORD *)&v4 - 2) = 0; *((_DWORD *)&v4 - 2) < *((_DWORD *)&v4 - 9); ++*((_DWORD *)&v4 - 2) )
  {
    if ( *(_BYTE *)(*((int *)&v4 - 2) + *(&v4 - 3)) == *(_BYTE *)(*((int *)&v4 - 2) + *(&v4 - 4)) )
      ++*((_DWORD *)&v4 - 1);
  }
  result = *((_DWORD *)&v4 - 1) == *((_DWORD *)&v4 - 9);
  __outbyte(3u, result);
  return result;
}
```

One more thing

- Decompiler output looks like hard garbage to read.



One more thing

```
                push    rbp
                mov     rbp, rsp
                mov     [rbp-18h], rdi        ; argv[1]
                mov     [rbp-20h], rsi        ; the base64 decoded buffer
                mov     [rbp-24h], edx        ; len to verify (48)
                mov     dword ptr [rbp-4], 0  ; validation counter
                mov     dword ptr [rbp-8], 0  ; loop counter
                jmp     short loc_8F0
loc_8C4:                                      ; CODE XREF: fg_verify_decryption+63↓j
                mov     eax, [rbp-8]          ; i
                movsxd  rdx, eax
                mov     rax, [rbp-18h]        ; arg1 pointer
                add     rax, rdx              ; move byte array one position ahead
                movzx   edx, byte ptr [rax]   ; read the byte from the arg1
                mov     eax, [rbp-8]          ; i
                movsxd  rcx, eax
                mov     rax, [rbp-20h]        ; decoded buffer
                add     rax, rcx              ; move decoded buffer ahead
                movzx   eax, byte ptr [rax]   ; read the byte from the decoded buffer
                cmp     dl, al                ; check if they match
                                              ; the decoded buffer must be the same as argv[1]
                jnz     short loc_8EC
                add     dword ptr [rbp-4], 1  ; byte is valid
loc_8EC:                                      ; CODE XREF: fg_verify_decryption+53↑j
                add     dword ptr [rbp-8], 1  ; advance counter
loc_8F0:                                      ; CODE XREF: fg_verify_decryption+2F↑j
                mov     eax, [rbp-8]
                cmp     eax, [rbp-24h]        ; check if we arrived to the end of the loop
                jl      short loc_8C4
                mov     eax, [rbp-4]
                cmp     eax, [rbp-24h]        ; compare with the input length
                                              ; 48 bytes need to be valid
                setz    al
                movzx   eax, al
                pop     rbp
```

One more thing

- Much easier to read and understand.
- Just comparing each byte and increasing counter when they match.
- All chars need to match.
- We found out that the first argument was 36 chars so we must pad to 48.
- Not a problem in the original code because enough space in RAM.

One more thing

```
__int64 __fastcall fg_decrypt(char *buf, int buf_len, int key)
{
  (...)
  memset(v6, 0, 64);
  // initialize the salsa20 key stream
  for ( i = 0; i <= 15; ++i ) {
    v5[i] = key;
  }
  sub_A7((__int64)v6, (__int64)v5);
  v8 = buf_len / 8;
  v7 = buf;
  for ( j = 0; ; j += 2 )
  {
    result = (unsigned int)j;
    if ( j >= v8 )
      break;
    // probably do the decryption
    sub_421((__int64 *)&v7[8 * j], (__int64 *)&v7[8 * j + 8], (__int64)v6);
  }
  return result;
}
```

One more thing

- We can identify hints of a possible Salsa20.
- But it’s not a straightforward implementation!
- I don’t want to reverse this stuff:
  - Too tired already.
  - Don’t like crypto that much (bad for CTFs).
  - Annoyed I spent too much time reimplementing Blowfish in a previous challenge.

One more thing

One more thing

- Unicorn Engine is great for these tasks.
- All the code is self contained so it is easy to setup and run.
- Learn and play with it. Be creative!
- Other solutions such as MIASM.

UNICORN
ENGINE

One more thing

- We just need to map the payload into the Unicorn VM.
- Copy the arguments to memory.
- Setup registers and start emulation at the function.
- Install code hooks to see what is going on.
- Dump memory when it ends.

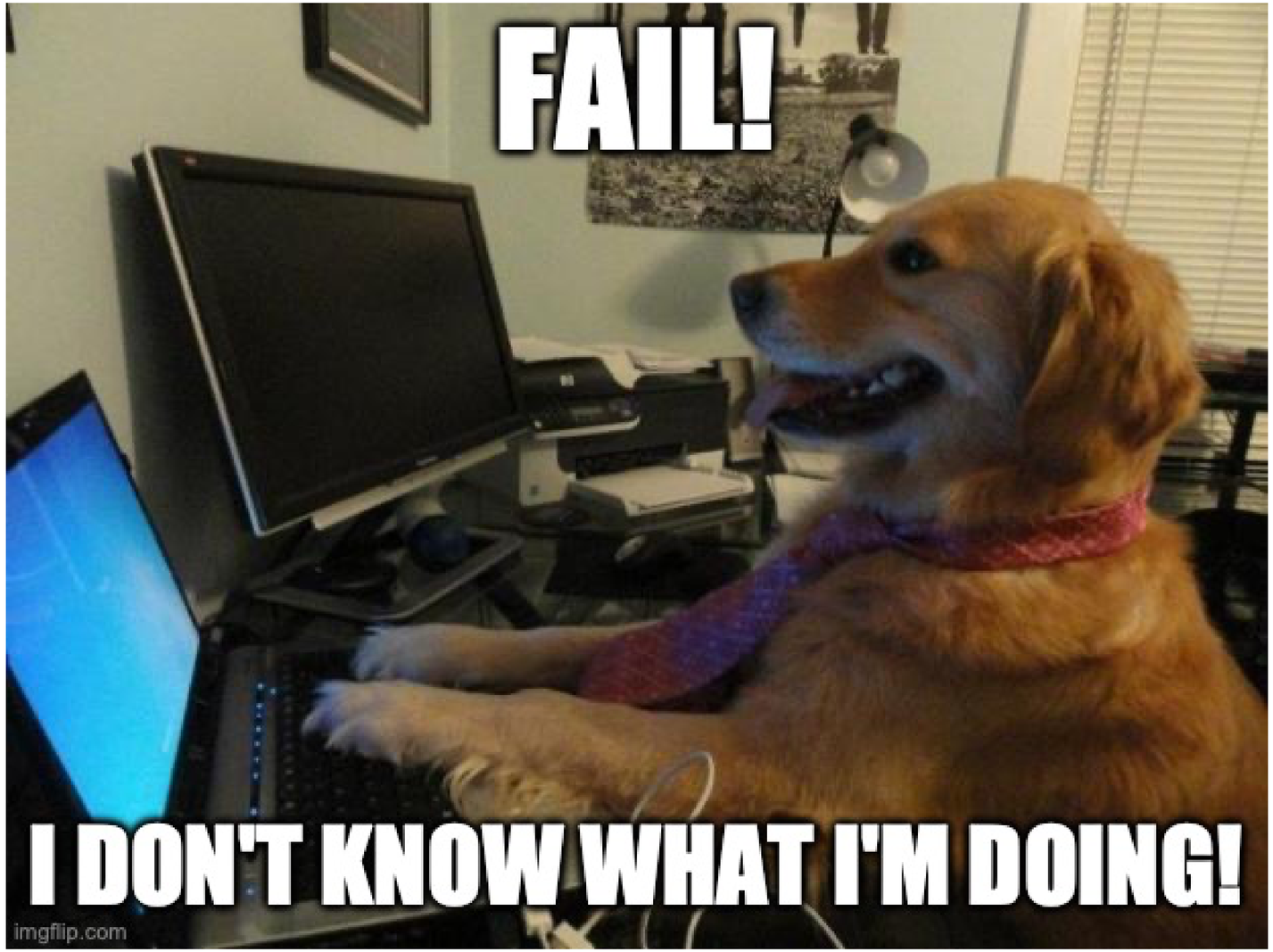
One more thing

- From the verification function we know that the decrypted contents must be equal to the first argument.
- The second argument is base64 encoded.
- We want to find the valid encrypted value.
- We can call the decryption routine with Unicorn to encrypt everything, which is our goal.

One more thing

- Start with base64(argv[1]), padded to 48 bytes.
- Run the emulated code and extract the result.
- Check what is going on inside the verification function.
    - All bytes should be equal and return 1.

One more thing



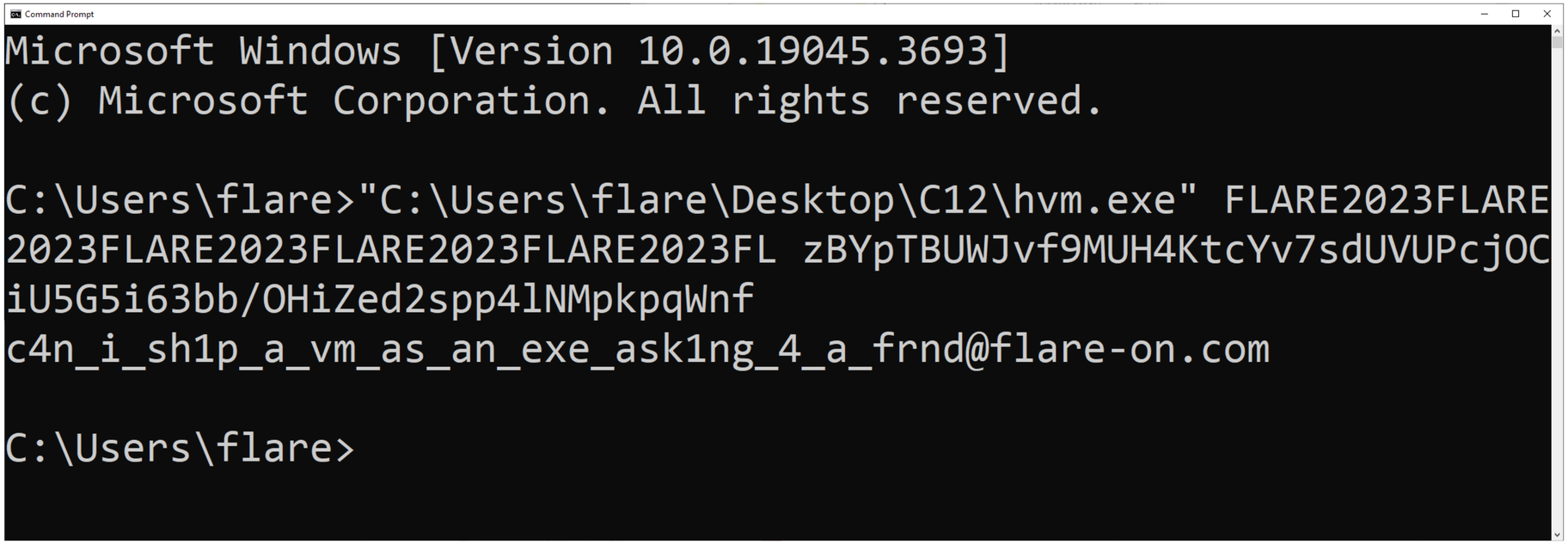
One more thing

- It should work in theory!
  - Very dangerous state because now I get obsessed to make it work or prove it doesn’t.
- Something is wrong.
- I still don’t want to reverse the crypto. It’s 4am or something.
- All of a sudden I have a stupid idea.
- Encrypt again the result!

One more thing

One more thing

- It works, don’t care.
- Submit the flag and move on. Still 2 spots available for top 50.
- Why was it failing?



Conclusions

Conclusions

- Significant amount of work but not that hard.
- Lots of details and general knowledge.
- Nothing that RTFM and some patience doesn’t solve.
- Great learning experience. Practice makes perfection!
- Hope to see you there next year. Goal is top 25!

Contacts, etc

- Blog: https://reverse.put.as
- Code: https://github.com/gdbinit
- Email: reverser@put.as
- IRC: #osxre @ irc.libera.chat
- Slack: 0xmadlabs.slack.com
- OpoSec: www.meetup.com/0xoposec/
- PGP key: https://reverse.put.as/E7CD23FD.asc

References

- Images from the internet. Credit due to their authors.