nani

IDA 打開從 main 開始分析，可以看到第三個呼叫的函數 4019A3 裡面用 ida 的反編譯結果只會看到一個 jumpout。直接看那部分的 assembly 可以看到大致上是如下:

```
4019B7 call $+5
4019BC pop rax
4019BD add rax, (offset loc_4019C9 - offset loc_4019BC)
4019C1 jmp rax
4019C3 db 0E9h, 80h, 87h, 55h, 66h
4019C8 1
4019C9 mov rax, cs:IsDebuggerPresent
```

它在 `call $+5` 的時候會 jump 到 4019BC 並同時 push 4019BC 到 stack 上面，所以經過 `pop rax` 就會使 `rax = 4019BC`，之後的 add 和 jmp 就讓它直接跳過一些 garbage 到 4019C9 繼續執行。所以剩下就能看出它是在利用 `IsDebuggerPresent` 檢查是否正在被 debug 著，這部分用在 x64dbg 中安裝 ScyllaHide 就可以解決。

通過 debugger 檢查後會呼叫 401869 的一個函數，裡面會利用 `cpuid` 這個 instruction 得到 cpu 資訊，然後檢查是否是常用的幾個 VM 軟體的特徵(KVM, Hyper-V, VMWare, VirtualBox etc)，是的話就一樣輸出訊息然後 exit。這邊我是用 CPUID Spoofer 去繞，或是可以直接 patch 掉那部分的 instructions。

通過 VM 檢查之後會呼叫 4017DF 的一個函數，裡面在 40181D 左右的地方會產生 exception。對照 PE-bear 可以找到它對應的 unwind info 在 BF0A4 (RVA)，跳過 5 個 unwind code 和 padding 可以知道 handler 的位置是 16FB (RVA)，直接在 x64dbg 中下斷點到 4016FB 可以之後它確實會呼叫到該函數。

handler 中會從把 401000~402000 這塊用 `VirtualProtect` 設為 rwx，之後將 4015AF 開始的 256 個 bytes 和 0x87 做 xor 之後再改回 r-x。之後把 handler 的第三個參數 pointer 的某個 offset 設定為 4015AF。

從簡報中知道第三個參數是 exception 的 context，用 IDA 改它的類型為 `_CONTEXT*` 之後就能看到那個 offset 是 `Rip`，所以代表它 return 之後會回到 4015AF 上面，所以可以知道 4015AF 上的應該是可以執行的 shellcode。

之後 x64dbg 繼續追蹤可以看到它會到 4015AF，裡面就大概是 xor 一些東西，然後可以在 stack 上面找到 flag。

Flag: `FLAG{r3v3rs3_Ma5T3R}`