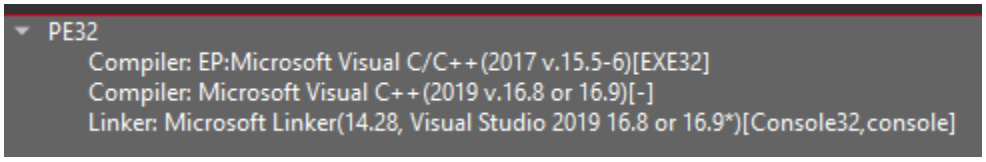
Blue Team Labs Online: Reverse Engineering – A Classic Injection

The following writeup is for Reverse Engineering -A classic Injection on Blue Team Labs Online, it's an easy lab that involves reverse engineering a binary using IDA among other tools. This was one of the most enjoyable challenges I have done in a while. If you have just started reverse engineering and basic malware analysis, this is a great way to practice your skills.

What is the name of the compiler used to generate the EXE?

If we open up the binary using a tool like PEStudio or DIE, we can see that this particular file was compiled using Microsoft Visual C++:

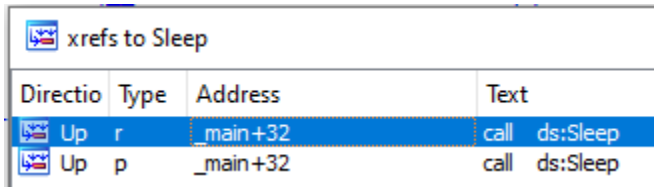
Microsoft Visual C++



The malware, when executed, sleeps for some time. What is the sleep time in minutes?

Start off by opening up the binary in a disassembler like IDA, or a tool like Cutter, Binary Ninja, etc. I typically like to pivot from strings, but in this case ill go to the imports table where we will see that sleep has been imported:

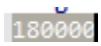
If we double click this, we will be taken to the .idata section. If you select Sleep and press CTRL + X we can see that sleep is called twice:



Let's check out the first cross reference:

```
push    2BF20h          ; dwMilliseconds
call    ds:Sleep
```

Here we can see that right before Sleep is called, the hex value 2BF20 is pushed to the stack. IDA is also kind enough to comment what this value is, so it's easy to determine that this hex value is simply how long the Sleep function goes for in milliseconds. If you select the value and press H, you can see its decimal value:



This converted to minutes is 3. Therefore, the program will sleep for 3 minutes once this sleep function executes.

After the sleep time, it prompts for user password, what is the correct password?

Right after the call to sleep, we can see that a string is moved into the edx register:

```
mov     edx, offset aBtlo ; "btlo"
```

This is the password.

What is the size of the shellcode?

If you scroll through the code, just past the sleep function, we come across this subroutine:

```
call    ds:CreateProcessW ; Indirect Call Near Procedure
push    3E8h              ; dwMilliseconds
push    [ebp+ProcessInformation.hProcess] ; hHandle
call    ds:WaitForSingleObject ; Indirect Call Near Procedure
push    40h ; '@'         ; flProtect
push    12288             ; flAllocationType
push    473               ; dwSize
push    0                 ; lpAddress
push    [ebp+ProcessInformation.hProcess] ; hProcess
call    ds:VirtualAllocEx ; Indirect Call Near Procedure
push    0                 ; lpNumberOfBytesWritten
mov     esi, eax
lea     eax, [ebp+Buffer] ; Load Effective Address
push    1D9h              ; nSize
push    eax               ; lpBuffer
push    esi               ; lpBaseAddress
push    [ebp+ProcessInformation.hProcess] ; hProcess
call    ds:WriteProcessMemory ; Indirect Call Near Procedure
push    0                 ; lpThreadId
push    0                 ; dwCreationFlags
push    0                 ; lpParameter
push    esi               ; lpStartAddress
push    0                 ; dwStackSize
push    0                 ; lpThreadAttributes
push    [ebp+ProcessInformation.hProcess] ; hProcess
call    ds:CreateRemoteThread ; Indirect Call Near Procedure
mov     edi, [ebp+Block]
jmp     short loc_40143F ; Jump
```

Based on all the function calls, this is clearly where the shellcode is written to memory. If you look at the VirutalAllocEx function's documentation, we can see that dwSize is the size of the region of memory to allocate in bytes. Therefore, we can assume that this parameter is the size of the shellcode being injected, which in this instance is 473 bytes (answer is 473).

Shellcode injection involves three important windows API. What is the name of the API call used?

As seen in the above image, shellcode injection requires VirtualAllocEx, WriteProcessMemory, and CreateRemoteThread. I believe that this question was meant to ask what the name of the third API call is, but due to a grammatical error this was overlooked. The answer is however, CreateRemoteThread.

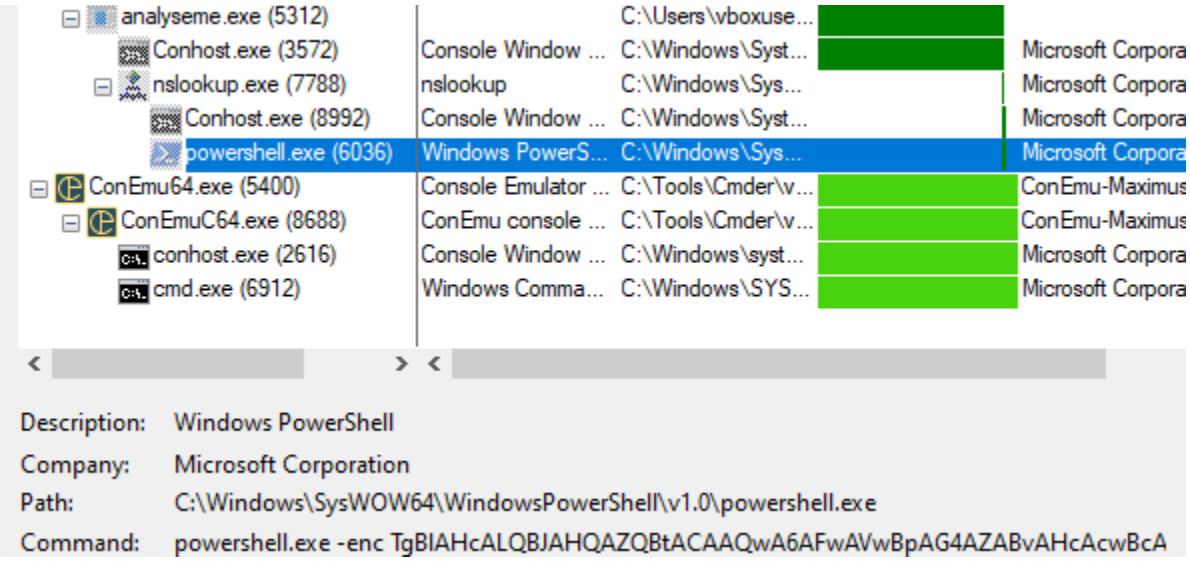
What is the name of the victim process?

Based on the parameters fed into the call to CreateProcessW, we can determine that the name of the victim process is nslookup.exe:

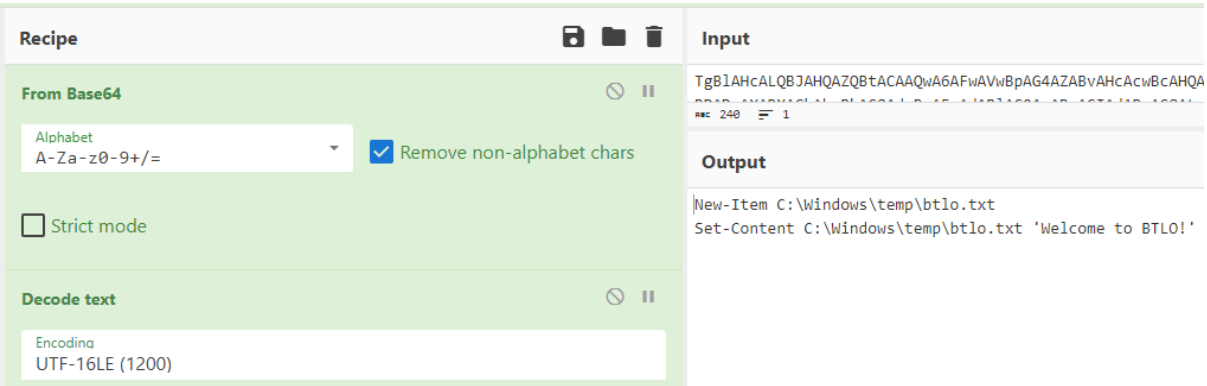
```
push    offset ApplicationName ; "C:\\Windows\\System32\\nslookup.exe"
movsb                   ; Move Byte(s) from String to String
movlpd  qword ptr [ebp+StartupInfo.lpTitle], xmm0 ; Move Low Packed Double-Precision Floating-Point Values
movlpd  qword ptr [ebp+StartupInfo.dwY], xmm0 ; Move Low Packed Double-Precision Floating-Point Values
movlpd  qword ptr [ebp+StartupInfo.dwYSize], xmm0 ; Move Low Packed Double-Precision Floating-Point Values
movlpd  qword ptr [ebp+StartupInfo.dwYCountChars], xmm0 ; Move Low Packed Double-Precision Floating-Point Values
movlpd  qword ptr [ebp+StartupInfo.dwFlags], xmm0 ; Move Low Packed Double-Precision Floating-Point Values
movlpd  qword ptr [ebp+StartupInfo.lpReserved2], xmm0 ; Move Low Packed Double-Precision Floating-Point Values
movlpd  qword ptr [ebp+StartupInfo.hStdOutput], xmm0 ; Move Low Packed Double-Precision Floating-Point Values
movups  xmmword ptr [ebp+ProcessInformation.hProcess], xmm0 ; Move Unaligned Four Packed Single-FP
call    ds:CreateProcessW ; Indirect Call Near Procedure
```

What is the file created by the sample

To answer this question, we need to perform some dynamic analysis. Therefore, I opened up Procmon and set the filter to the name of the binary and executed it. Remember, we need to wait 3 minutes due to the sleep function, a ? will appear and then we need to enter the password (btlo). If we look at the process tree in Procmon, we can see that powershell.exe was executed by nslookup.exe:



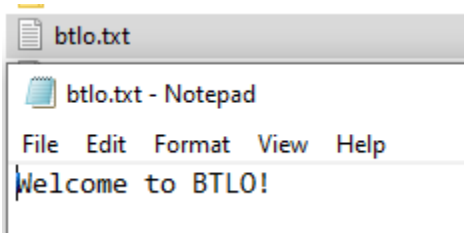
Here we can see a base64 encoded command, so let's decode it using Cyberchef:



As you can see, I used the From Base64 and Decode text recipes to decode the base64 string. Therefore, the answer is C:\Windows\temp\btlo.txt

What is the message in the created file

As you can see in the decode text, Welcome to BTLO! Was written to the btlo.txt file. You can also navigate to this file in the temp directory to find the answer:



What is the program that the shellcode used to create and write this file?

powershell.exe