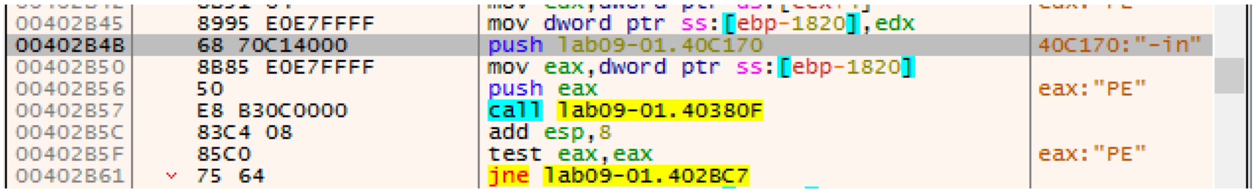
Professor Jonathan S. Weissman CSEC 466 PMA 9-1, 9-2, 9-3

Name: Smayan Daruka
Date: 11/07/18

LAB 9-1

1. How can you get this malware to install itself?

The malware installs itself when run with the “-in” flag/command line argument. We also need to provide the password in order to successfully install the malware.

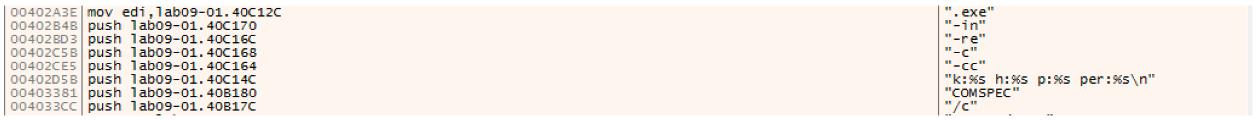


```
00402B45    8995 E0E7FFFF    mov dword ptr ss:[ebp-1820],edx
00402B4B    68 70C14000      push lab09-01.40C170                40C170:"-in"
00402B50    8B85 E0E7FFFF    mov eax,dword ptr ss:[ebp-1820]
00402B56    50               push eax                            eax:"PE"
00402B57    E8 B30C0000      call lab09-01.40380F
00402B5C    83C4 08          add esp,8
00402B5F    85C0             test eax,eax                        eax:"PE"
00402B61    75 64            jne lab09-01.402BC7
```

As can be seen in the above image, the highlighted command checks whether the “-in” command line argument is present or not.

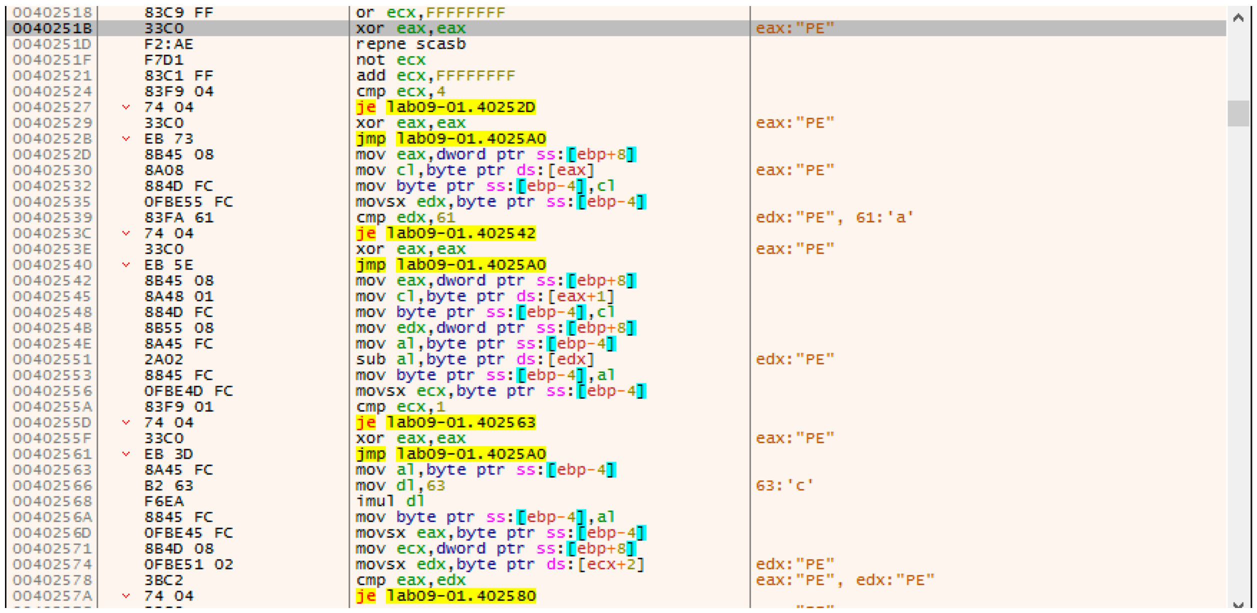
2. What are the command-line options for this program? What is the password requirement?

The image below shows the different command-line options for this program:



```
00402A3E mov edi,lab09-01.40C12C     ".exe"
00402B4B push lab09-01.40C170        "-in"
00402BD3 push lab09-01.40C16C        "-re"
00402C5B push lab09-01.40C168        "-c"
00402CE5 push lab09-01.40C164        "-cc"
00402D5B push lab09-01.40C14C        "k:%s h:%s p:%s per:%s\n"
00403381 push lab09-01.40B180        "COMSPEC"
004033CC push lab09-01.40B17C        "/c"
```

As can be seen above, “-in”, “-re”, “-c”, and “-cc” are the 4 different command-line options. The “-in” argument installs the malware, “-re” argument removes the malware, “-c” argument updates the malware’s configuration, and “-cc” argument prints the malware’s current configuration to the screen.

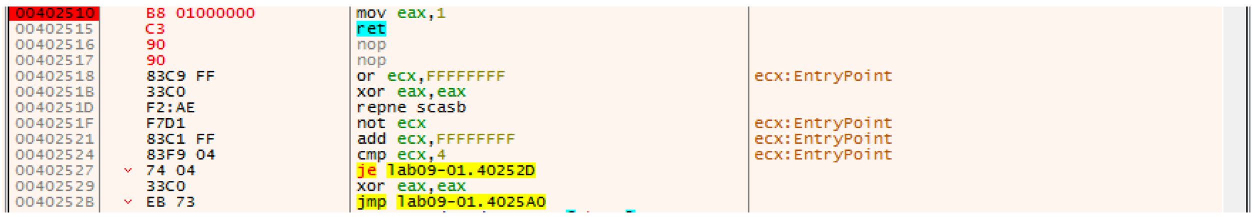


```
00402518      83C9 FF        or ecx,FFFFFFFF
0040251B      33C0           xor eax,eax                        eax:"PE"
0040251D      F2:AE          repne scasb
0040251F      F7D1           not ecx
00402521      83C1 FF        add ecx,FFFFFFFF
00402524      83F9 04        cmp ecx,4
00402527    ⌄ 74 04          je lab09-01.40252D
00402529      33C0           xor eax,eax                        eax:"PE"
0040252B    ⌄ EB 73          jmp lab09-01.4025A0
0040252D      8B45 08        mov eax,dword ptr ss:[ebp+8]
00402530      8A08           mov cl,byte ptr ds:[eax]           eax:"PE"
00402532      884D FC        mov byte ptr ss:[ebp-4],cl
00402535      0FBE55 FC      movsx edx,byte ptr ss:[ebp-4]
00402539      83FA 61        cmp edx,61                         edx:"PE", 61:'a'
0040253C    ⌄ 74 04          je lab09-01.402542
0040253E      33C0           xor eax,eax                        eax:"PE"
00402540    ⌄ EB 5E          jmp lab09-01.4025A0
00402542      8B45 08        mov eax,dword ptr ss:[ebp+8]
00402545      8A48 01        mov cl,byte ptr ds:[eax+1]
00402548      884D FC        mov byte ptr ss:[ebp-4],cl
0040254B      8B55 08        mov edx,dword ptr ss:[ebp+8]
0040254E      8A45 FC        mov al,byte ptr ss:[ebp-4]
00402551      2A02           sub al,byte ptr ds:[edx]           edx:"PE"
00402553      8845 FC        mov byte ptr ss:[ebp-4],al
00402556      0FBE4D FC      movsx ecx,byte ptr ss:[ebp-4]
0040255A      83F9 01        cmp ecx,1
0040255D    ⌄ 74 04          je lab09-01.402563
0040255F      33C0           xor eax,eax                        eax:"PE"
00402561    ⌄ EB 3D          jmp lab09-01.4025A0
00402563      8A45 FC        mov al,byte ptr ss:[ebp-4]
00402566      B2 63          mov dl,63                          63:'c'
00402568      F6EA           imul dl
0040256A      8845 FC        mov byte ptr ss:[ebp-4],al
0040256D      0FBE45 FC      movsx eax,byte ptr ss:[ebp-4]
00402571      8B4D 08        mov ecx,dword ptr ss:[ebp+8]
00402574      0FBE51 02      movsx edx,byte ptr ds:[ecx+2]      edx:"PE"
00402578      3BC2           cmp eax,edx                        eax:"PE", edx:"PE"
0040257A    ⌄ 74 04          je lab09-01.402580
```

As can be seen in the above image, the first character of the password is ‘a’. Then, it performs mathematical operations to go to the second ASCII character ‘b’. ‘c’ is explicitly tested for and ‘d’ is mathematically computed as with ‘b’. In the end, the password turns out to be ‘abcd’.

3. How can you use OllyDbg to permanently patch this malware, so that it doesn’t require the special command-line password?

The malware can be patched as seen in the screenshot below:

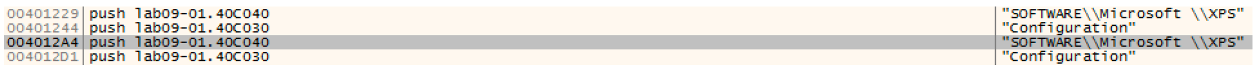


```
00402510      B8 01000000    mov eax,1
00402515      C3             ret
00402516      90             nop
00402517      90             nop
00402518      83C9 FF        or ecx,FFFFFFFF                    ecx:EntryPoint
0040251B      33C0           xor eax,eax
0040251D      F2:AE          repne scasb
0040251F      F7D1           not ecx                            ecx:EntryPoint
00402521      83C1 FF        add ecx,FFFFFFFF                   ecx:EntryPoint
00402524      83F9 04        cmp ecx,4                          ecx:EntryPoint
00402527    ⌄ 74 04          je lab09-01.40252D
00402529      33C0           xor eax,eax
0040252B    ⌄ EB 73          jmp lab09-01.4025A0
```

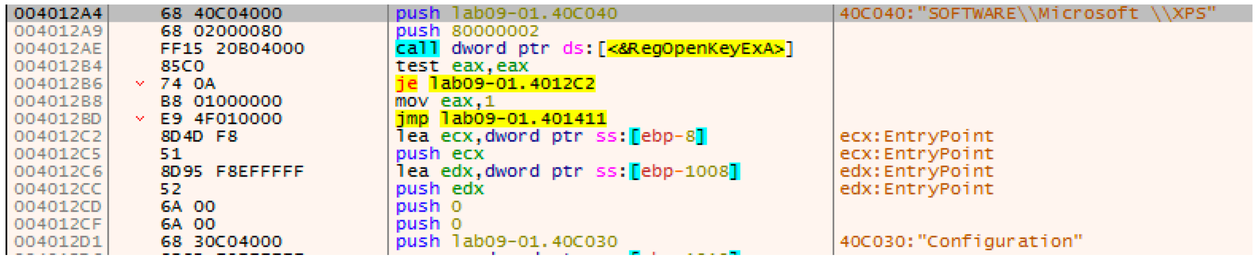
I changed the password checking function’s first few bytes to always return 1 no matter what password is provided or entered. I first moved 1 into eax and returned true to skip the password verification process.

4. What are the host-based indicators of this malware?

This malware creates a registry key at “HKLM\Software\Microsoft \XPS\Configuration”.



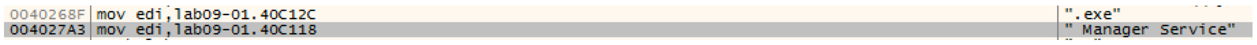
```
00401229 push lab09-01.40C040        "SOFTWARE\\Microsoft \\XPS"
00401244 push lab09-01.40C030        "Configuration"
004012A4 push lab09-01.40C040        "SOFTWARE\\Microsoft \\XPS"
004012D1 push lab09-01.40C030        "Configuration"
```



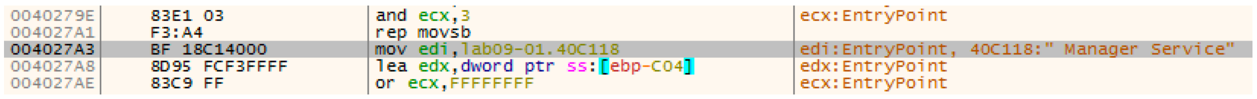
```
004012A4   68 40C04000     push lab09-01.40C040               40C040:"SOFTWARE\\Microsoft \\XPS"
004012A9   68 02000080     push 80000002
004012AE   FF15 20B04000   call dword ptr ds:[<&RegOpenKeyExA>]
004012B4   85C0            test eax,eax
004012B6   74 0A           je lab09-01.4012C2
004012B8   B8 01000000     mov eax,1
004012BD   E9 4F010000     jmp lab09-01.401411
004012C2   8D4D F8         lea ecx,dword ptr ss:[ebp-8]       ecx:EntryPoint
004012C5   51              push ecx                           ecx:EntryPoint
004012C6   8D95 F8EFFFFF   lea edx,dword ptr ss:[ebp-1008]    edx:EntryPoint
004012CC   52              push edx                           edx:EntryPoint
004012CD   6A 00           push 0
004012CF   6A 00           push 0
004012D1   68 30C04000     push lab09-01.40C030               40C030:"Configuration"
```

As can be seen in the above two screenshots, the registry key is being created.

This malware also creates a service called “XYZ Manager Service” and XYZ can be specified during run-time (installation). The screenshots below show the service being created.

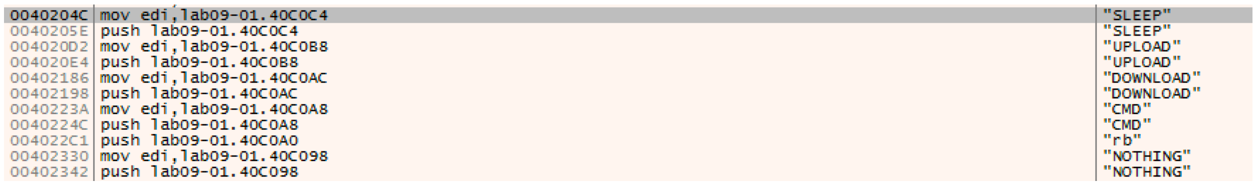


```
0040268F mov edi,lab09-01.40C12C        ".exe"
004027A3 mov edi,lab09-01.40C118        " Manager Service"
```



```
0040279E   83E1 03         and ecx,3                          ecx:EntryPoint
004027A1   F3:A4           rep movsb
004027A3   BF 18C14000     mov edi,lab09-01.40C118            edi:EntryPoint, 40C118:" Manager Service"
004027A8   8D95 FCF3FFFF   lea edx,dword ptr ss:[ebp-C04]     edx:EntryPoint
004027AE   83C9 FF         or ecx,FFFFFFFF                    ecx:EntryPoint
```

5. What are the different actions this malware can be instructed to take via the network?

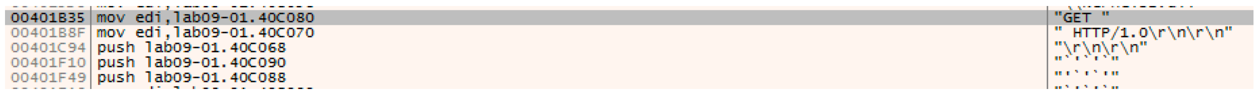


```
0040204C mov edi,lab09-01.40C0C4        "SLEEP"
0040205E push lab09-01.40C0C4           "SLEEP"
004020D2 mov edi,lab09-01.40C0B8        "UPLOAD"
004020E4 push lab09-01.40C0B8           "UPLOAD"
00402186 mov edi,lab09-01.40C0AC        "DOWNLOAD"
00402198 push lab09-01.40C0AC           "DOWNLOAD"
0040223A mov edi,lab09-01.40C0A8        "CMD"
0040224C push lab09-01.40C0A8           "CMD"
004022C1 push lab09-01.40C0A0           "rb"
00402330 mov edi,lab09-01.40C098        "NOTHING"
00402342 push lab09-01.40C098           "NOTHING"
```

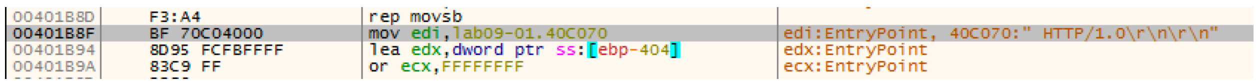
As can be seen in the above screenshot, this malware can take the following actions via the network: sleep, upload, download, cmd, and nothing.

The sleep command is used to instruct the malware not to perform any actions for a given period of time. The upload command is used to read a file from the network and write it to the local system at the path specified. The download command instructs the malware to send the contents of said file earlier over the network to a remote host. The cmd command instructs the malware to open a shell on the local system. Finally, the nothing command instructs the malware to do nothing by issuing a no-op command.

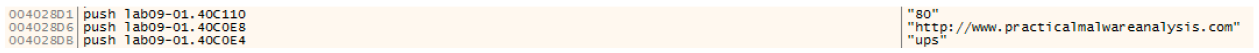
6. Are there any useful network-based signatures for this malware?



```
00401B35 mov edi,lab09-01.40C080                    "GET "
00401B8F mov edi,lab09-01.40C070                    " HTTP/1.0\r\n\r\n"
00401C94 push lab09-01.40C068                       "\r\n\r\n"
00401F10 push lab09-01.40C090                       "`'`'"
00401F49 push lab09-01.40C088                       "'`'`"
```



```
00401B8D   F3:A4          rep movsb
00401B8F   BF 70C04000    mov edi,lab09-01.40C070        edi:EntryPoint, 40C070:" HTTP/1.0\r\n\r\n"
00401B94   8D95 FCFBFFFF  lea edx,dword ptr ss:[ebp-404] edx:EntryPoint
00401B9A   83C9 FF        or ecx,FFFFFFFF                ecx:EntryPoint
```

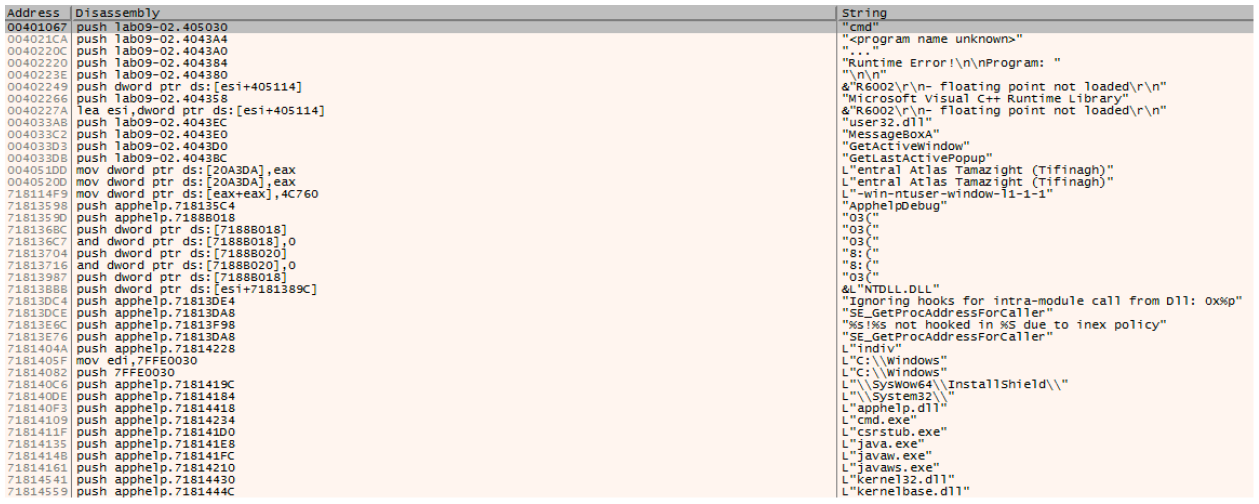


```
004028D1 push lab09-01.40C110              "80"
004028D6 push lab09-01.40C0E8              "http://www.practicalmalwareanalysis.com"
004028DB push lab09-01.40C0E4              "ups"
```

As can be seen in the above images, this malware uses HTTP 1.0 GET requests and beacons out to "http://www.practicalmalwareanalysis.com". One important thing to note is that the malware does not provide any HTTP headers with the requests. Further analysis revealed that this is an HTTP reverse backdoor.

LAB 9-2

1. What strings do you see statically in the binary?



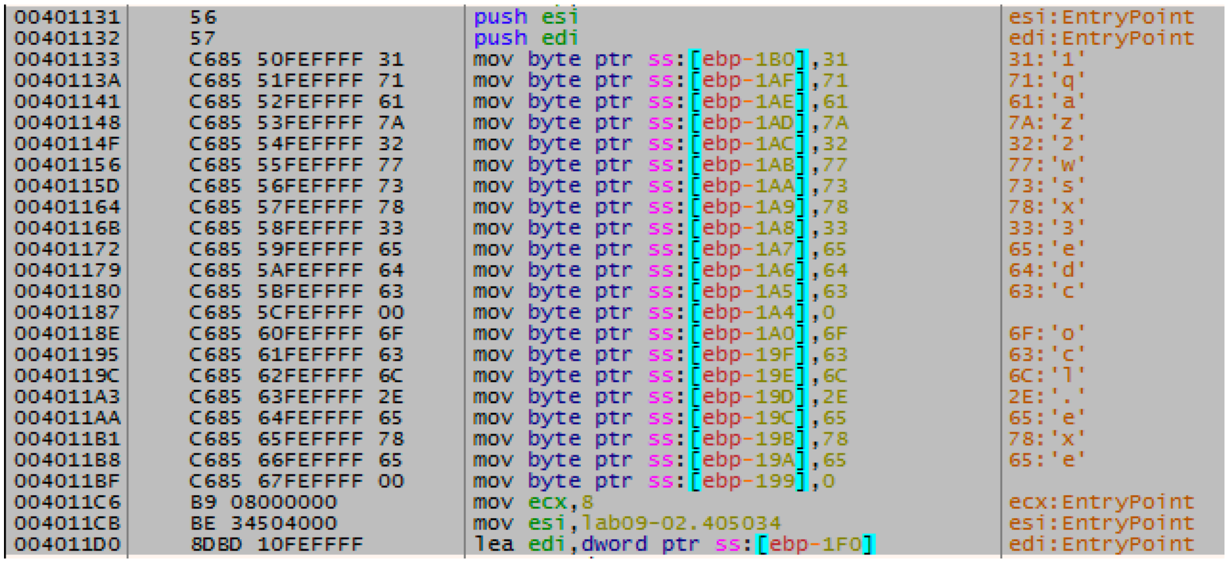
```
Address  Disassembly                                 String
00401067 push lab09-02.405030                        "cmd"
004021CA push lab09-02.4043A4                        "<program name unknown>"
0040220C push lab09-02.4043A0                        "..."
00402220 push lab09-02.404384                        "Runtime Error!\n\nProgram: "
0040223E push lab09-02.404380                        "\n\n"
00402249 push dword ptr ds:[esi+405114]              &"R6002\r\n- floating point not loaded\r\n"
00402266 push lab09-02.404358                        "Microsoft Visual C++ Runtime Library"
0040227A lea esi,dword ptr ds:[esi+405114]           &"R6002\r\n- floating point not loaded\r\n"
004033AB push lab09-02.4043EC                        "user32.dll"
004033C2 push lab09-02.4043E0                        "MessageBoxA"
004033D3 push lab09-02.4043D0                        "GetActiveWindow"
004033DB push lab09-02.4043BC                        "GetLastActivePopup"
004051DD mov dword ptr ds:[20A3DA],eax               L"entral Atlas Tamazight (Tifinagh)"
0040520D mov dword ptr ds:[20A3DA],eax               L"entral Atlas Tamazight (Tifinagh)"
718114F9 mov dword ptr ds:[eax+eax],4C760            L"-win-ntuser-window-l1-1-1"
71813598 push apphelp.718135C4                       "ApphelpDebug"
7181359D push apphelp.7188B018                       "03("
718136BC push dword ptr ds:[7188B018]                "03("
718136C7 and dword ptr ds:[7188B018],0               "03("
71813704 push dword ptr ds:[7188B020]                "8:("
71813716 and dword ptr ds:[7188B020],0               "8:("
71813987 push dword ptr ds:[7188B018]                "03("
71813BBB push dword ptr ds:[esi+7181389C]            &L"NTDLL.DLL"
71813DC4 push apphelp.71813DE4                       "Ignoring hooks for intra-module call from Dll: 0x%p"
71813DCE push apphelp.71813DA8                       "SE_GetProcAddressForCaller"
71813E6C push apphelp.71813F98                       "%s!%s not hooked in %S due to inex policy"
71813E76 push apphelp.71813DA8                       "SE_GetProcAddressForCaller"
7181404A push apphelp.71814228                       L"indiv"
7181405F mov edi,7FFE0030                            L"C:\\Windows"
71814082 push 7FFE0030                               L"C:\\Windows"
718140C6 push apphelp.7181419C                       L"\\SysWow64\\InstallShield\\"
718140DE push apphelp.71814184                       L"\\System32\\"
718140F3 push apphelp.71814418                       L"apphelp.dll"
71814109 push apphelp.71814234                       L"cmd.exe"
7181411F push apphelp.718141D0                       L"csrstub.exe"
71814135 push apphelp.718141E8                       L"java.exe"
7181414B push apphelp.718141FC                       L"javaw.exe"
71814161 push apphelp.71814210                       L"javaws.exe"
71814541 push apphelp.71814430                       L"kernel32.dll"
71814559 push apphelp.7181444C                       L"kernelbase.dll"
```

As can be seen above, 'cmd' is one of the statically appearing strings in this binary. Also, there are many imports like apphelp.dll or java.exe that are also statically appearing in the binary.

2. What happens when you run this binary?

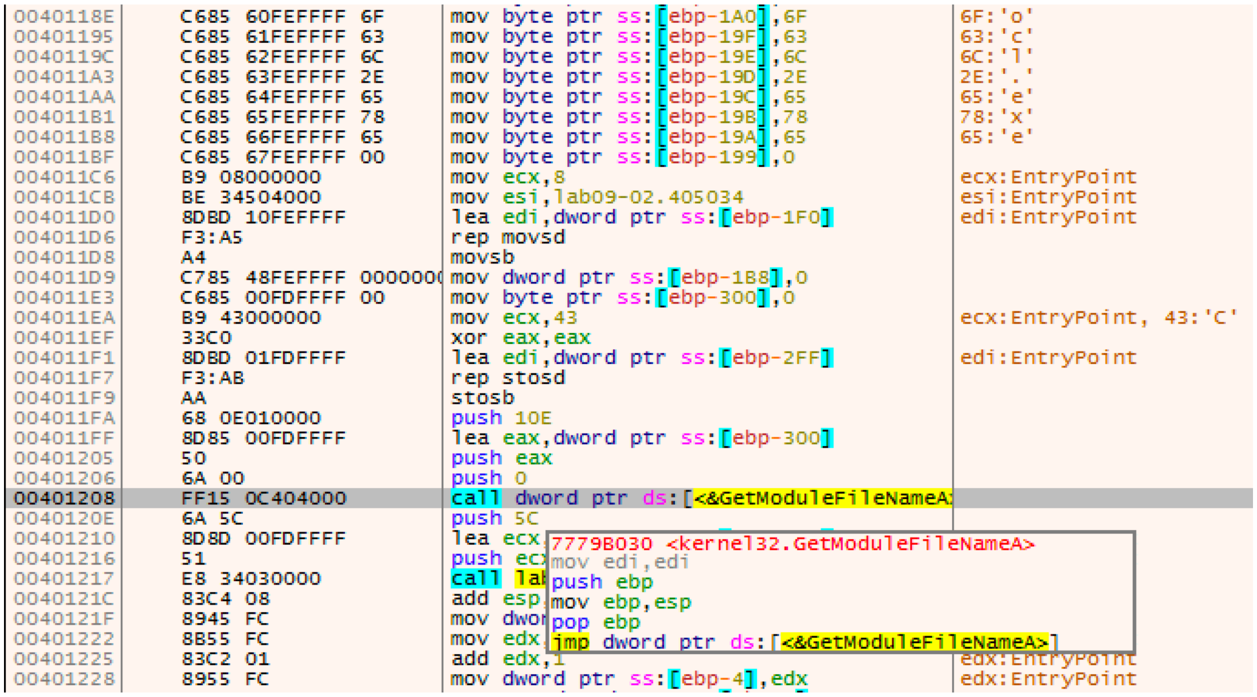
This binary only has one breakpoint and even after stepping over, the binary just exits within 5-8 step overs. This binary doesn't seem like it does a whole lot, if anything at all.

3. How can you get this sample to run its malicious payload?



```
00401131   56                 push esi                          esi:EntryPoint
00401132   57                 push edi                          edi:EntryPoint
00401133   C685 50FEFFFF 31   mov byte ptr ss:[ebp-1B0],31      31:'1'
0040113A   C685 51FEFFFF 71   mov byte ptr ss:[ebp-1AF],71      71:'q'
00401141   C685 52FEFFFF 61   mov byte ptr ss:[ebp-1AE],61      61:'a'
00401148   C685 53FEFFFF 7A   mov byte ptr ss:[ebp-1AD],7A      7A:'z'
0040114F   C685 54FEFFFF 32   mov byte ptr ss:[ebp-1AC],32      32:'2'
00401156   C685 55FEFFFF 77   mov byte ptr ss:[ebp-1AB],77      77:'w'
0040115D   C685 56FEFFFF 73   mov byte ptr ss:[ebp-1AA],73      73:'s'
00401164   C685 57FEFFFF 78   mov byte ptr ss:[ebp-1A9],78      78:'x'
0040116B   C685 58FEFFFF 33   mov byte ptr ss:[ebp-1A8],33      33:'3'
00401172   C685 59FEFFFF 65   mov byte ptr ss:[ebp-1A7],65      65:'e'
00401179   C685 5AFEFFFF 64   mov byte ptr ss:[ebp-1A6],64      64:'d'
00401180   C685 5BFEFFFF 63   mov byte ptr ss:[ebp-1A5],63      63:'c'
00401187   C685 5CFEFFFF 00   mov byte ptr ss:[ebp-1A4],0
0040118E   C685 60FEFFFF 6F   mov byte ptr ss:[ebp-1A0],6F      6F:'o'
00401195   C685 61FEFFFF 63   mov byte ptr ss:[ebp-19F],63      63:'c'
0040119C   C685 62FEFFFF 6C   mov byte ptr ss:[ebp-19E],6C      6C:'l'
004011A3   C685 63FEFFFF 2E   mov byte ptr ss:[ebp-19D],2E      2E:'.'
004011AA   C685 64FEFFFF 65   mov byte ptr ss:[ebp-19C],65      65:'e'
004011B1   C685 65FEFFFF 78   mov byte ptr ss:[ebp-19B],78      78:'x'
004011B8   C685 66FEFFFF 65   mov byte ptr ss:[ebp-19A],65      65:'e'
004011BF   C685 67FEFFFF 00   mov byte ptr ss:[ebp-199],0
004011C6   B9 08000000        mov ecx,8                         ecx:EntryPoint
004011CB   BE 34504000        mov esi,lab09-02.405034           esi:EntryPoint
004011D0   8DBD 10FEFFFF      lea edi,dword ptr ss:[ebp-1F0]    edi:EntryPoint
```

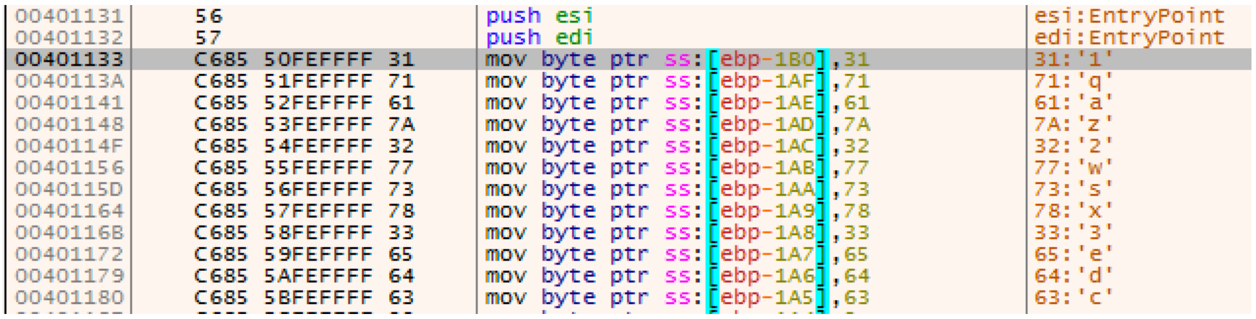
As can be seen above, there are two strings being created. The first string is "1qaz2wsx3edc" and the second string is "ocl.exe".



```
0040118E   C685 60FEFFFF 6F   mov byte ptr ss:[ebp-1A0],6F      6F:'o'
00401195   C685 61FEFFFF 63   mov byte ptr ss:[ebp-19F],63      63:'c'
0040119C   C685 62FEFFFF 6C   mov byte ptr ss:[ebp-19E],6C      6C:'l'
004011A3   C685 63FEFFFF 2E   mov byte ptr ss:[ebp-19D],2E      2E:'.'
004011AA   C685 64FEFFFF 65   mov byte ptr ss:[ebp-19C],65      65:'e'
004011B1   C685 65FEFFFF 78   mov byte ptr ss:[ebp-19B],78      78:'x'
004011B8   C685 66FEFFFF 65   mov byte ptr ss:[ebp-19A],65      65:'e'
004011BF   C685 67FEFFFF 00   mov byte ptr ss:[ebp-199],0
004011C6   B9 08000000        mov ecx,8                         ecx:EntryPoint
004011CB   BE 34504000        mov esi,lab09-02.405034           esi:EntryPoint
004011D0   8DBD 10FEFFFF      lea edi,dword ptr ss:[ebp-1F0]    edi:EntryPoint
004011D6   F3:A5              rep movsd
004011D8   A4                 movsb
004011D9   C785 48FEFFFF 000000( mov dword ptr ss:[ebp-1B8],0
004011E3   C685 00FDFFFF 00   mov byte ptr ss:[ebp-300],0
004011EA   B9 43000000        mov ecx,43                        ecx:EntryPoint, 43:'C'
004011EF   33C0               xor eax,eax
004011F1   8DBD 01FDFFFF      lea edi,dword ptr ss:[ebp-2FF]    edi:EntryPoint
004011F7   F3:AB              rep stosd
004011F9   AA                 stosb
004011FA   68 0E010000        push 10E
004011FF   8D85 00FDFFFF      lea eax,dword ptr ss:[ebp-300]
00401205   50                 push eax
00401206   6A 00              push 0
00401208   FF15 0C404000      call dword ptr ds:[<&GetModuleFileNameA:
0040120E   6A 5C              push 5C
00401210   8D8D 00FDFFFF      lea ecx,7779B030 <kernel32.GetModuleFileNameA>
00401216   51                 push ec mov edi,edi
00401217   E8 34030000        call lab push ebp
0040121C   83C4 08            add esp mov ebp,esp
0040121F   8945 FC            mov dwo pop ebp
00401222   8B55 FC            mov edx jmp dword ptr ds:[<&GetModuleFileNameA>]
00401225   83C2 01            add edx,1                         edx:EntryPoint
00401228   8955 FC            mov dword ptr ss:[ebp-4],edx      edx:EntryPoint
```

The malicious payload can be run if we rename the "ocl.exe" string before running it as can be seen in the above image where a call to 'GetModuleFileNameA' exists.

4. What is happening at 0x00401133?

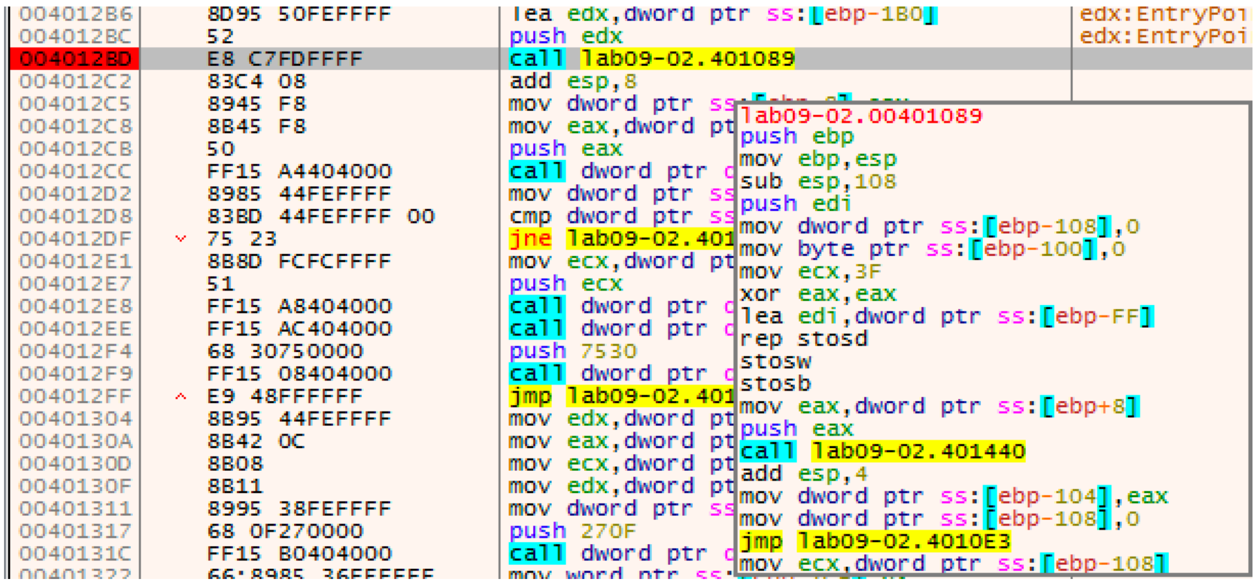


```
00401131   56                  push esi                          esi:EntryPoint
00401132   57                  push edi                          edi:EntryPoint
00401133   C685 50FEFFFF 31    mov byte ptr ss:[ebp-1B0],31      31:'1'
0040113A   C685 51FEFFFF 71    mov byte ptr ss:[ebp-1AF],71      71:'q'
00401141   C685 52FEFFFF 61    mov byte ptr ss:[ebp-1AE],61      61:'a'
00401148   C685 53FEFFFF 7A    mov byte ptr ss:[ebp-1AD],7A      7A:'z'
0040114F   C685 54FEFFFF 32    mov byte ptr ss:[ebp-1AC],32      32:'2'
00401156   C685 55FEFFFF 77    mov byte ptr ss:[ebp-1AB],77      77:'w'
0040115D   C685 56FEFFFF 73    mov byte ptr ss:[ebp-1AA],73      73:'s'
00401164   C685 57FEFFFF 78    mov byte ptr ss:[ebp-1A9],78      78:'x'
0040116B   C685 58FEFFFF 33    mov byte ptr ss:[ebp-1A8],33      33:'3'
00401172   C685 59FEFFFF 65    mov byte ptr ss:[ebp-1A7],65      65:'e'
00401179   C685 5AFEFFFF 64    mov byte ptr ss:[ebp-1A6],64      64:'d'
00401180   C685 5BFEFFFF 63    mov byte ptr ss:[ebp-1A5],63      63:'c'
```

As can be seen above, at the specified memory location, a string is being built on the stack. This is done by moving each character one at a time and thus obfuscating the string to prevent being found by simple string utilities.

5. What arguments are being passed to subroutine 0x00401089?

There are two arguments that are being passed into the subroutine at 0x00401089. The first is the string “1qaz2wsx3edc” that was created earlier and the second is a pointer to a buffer of data. This can be seen in the screenshot below:

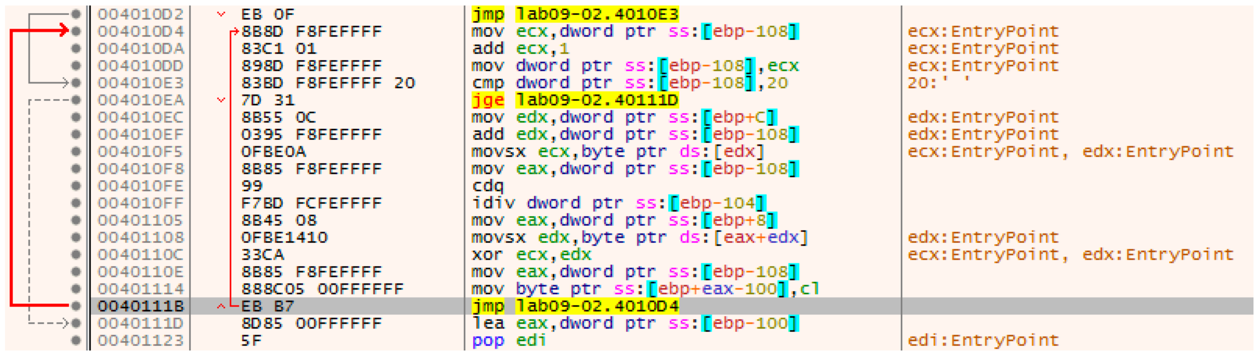


```
004012B6     8D95 50FEFFFF       lea edx,dword ptr ss:[ebp-1B0]    edx:EntryPoi
004012BC     52                  push edx                          edx:EntryPoi
004012BD     E8 C7FDFFFF         call lab09-02.401089
004012C2     83C4 08             add esp,8
004012C5     8945 F8             mov dword ptr ss
004012C8     8B45 F8             mov eax,dword pt
004012CB     50                  push eax
004012CC     FF15 A4404000       call dword ptr d
004012D2     8985 44FEFFFF       mov dword ptr ss
004012D8     83BD 44FEFFFF 00    cmp dword ptr ss
004012DF   v 75 23               jne lab09-02.401
004012E1     8B8D FCFCFFFF       mov ecx,dword pt
004012E7     51                  push ecx
004012E8     FF15 A8404000       call dword ptr d
004012EE     FF15 AC404000       call dword ptr d
004012F4     68 30750000         push 7530
004012F9     FF15 08404000       call dword ptr d
004012FF   ^ E9 48FFFFFF         jmp lab09-02.401
00401304     8B95 44FEFFFF       mov edx,dword pt
0040130A     8B42 0C             mov eax,dword pt
0040130D     8B08                mov ecx,dword pt
0040130F     8B11                mov edx,dword pt
00401311     8995 38FEFFFF       mov dword ptr ss
00401317     68 0F270000         push 270F
0040131C     FF15 B0404000       call dword ptr d
```

```
lab09-02.00401089
push ebp
mov ebp,esp
sub esp,108
push edi
mov dword ptr ss:[ebp-108],0
mov byte ptr ss:[ebp-100],0
mov ecx,3F
xor eax,eax
lea edi,dword ptr ss:[ebp-FF]
rep stosd
stosw
stosb
mov eax,dword ptr ss:[ebp+8]
push eax
call lab09-02.401440
add esp,4
mov dword ptr ss:[ebp-104],eax
mov dword ptr ss:[ebp-108],0
jmp lab09-02.4010E3
mov ecx,dword ptr ss:[ebp-108]
```

6. What domain name does this malware use?

The domain name being used is "practicalmalwareanalysis.com" as can be seen in the screenshot below:



```
004010D2   EB 0F               jmp lab09-02.4010E3
004010D4   8B8D F8FEFFFF       mov ecx,dword ptr ss:[ebp-108]          ecx:EntryPoint
004010DA   83C1 01             add ecx,1                               ecx:EntryPoint
004010DD   898D F8FEFFFF       mov dword ptr ss:[ebp-108],ecx          ecx:EntryPoint
004010E3   83BD F8FEFFFF 20    cmp dword ptr ss:[ebp-108],20           20:' '
004010EA   7D 31               jge lab09-02.40111D
004010EC   8B55 0C             mov edx,dword ptr ss:[ebp+C]            edx:EntryPoint
004010EF   0395 F8FEFFFF       add edx,dword ptr ss:[ebp-108]          edx:EntryPoint
004010F5   0FBE0A              movsx ecx,byte ptr ds:[edx]             ecx:EntryPoint, edx:EntryPoint
004010F8   8B85 F8FEFFFF       mov eax,dword ptr ss:[ebp-108]
004010FE   99                  cdq
004010FF   F7BD FCFEFFFF       idiv dword ptr ss:[ebp-104]
00401105   8B45 08             mov eax,dword ptr ss:[ebp+8]
00401108   0FBE1410            movsx edx,byte ptr ds:[eax+edx]         edx:EntryPoint
0040110C   33CA                xor ecx,edx                             ecx:EntryPoint, edx:EntryPoint
0040110E   8B85 F8FEFFFF       mov eax,dword ptr ss:[ebp-108]
00401114   888C05 00FFFFFF     mov byte ptr ss:[ebp+eax-100],cl
0040111B   EB B7               jmp lab09-02.4010D4
0040111D   8D85 00FFFFFF       lea eax,dword ptr ss:[ebp-100]
00401123   5F                  pop edi                                 edi:EntryPoint
```

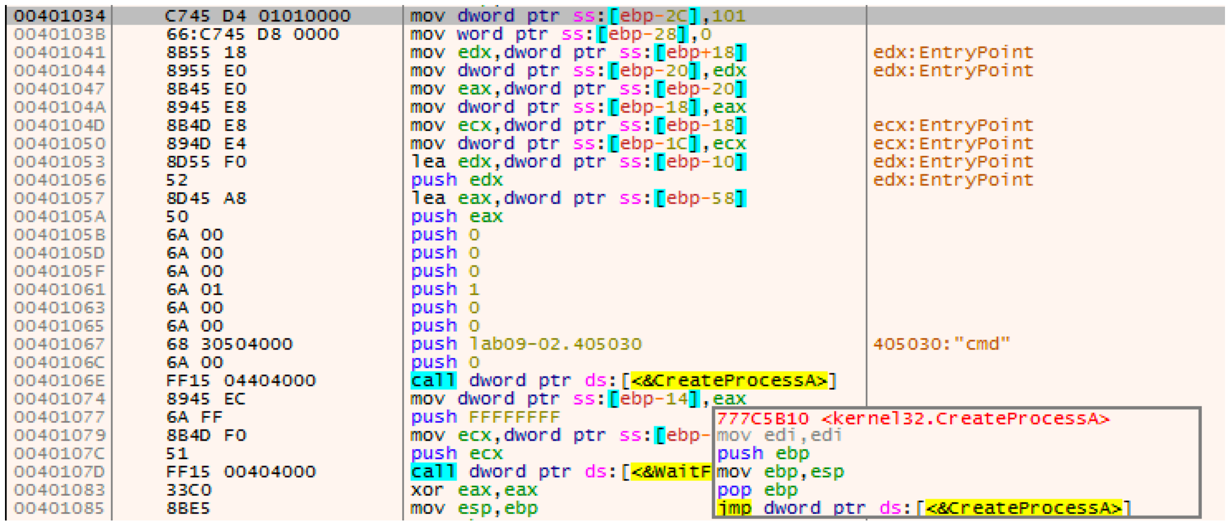
As can be seen above, the loop is executed multiple times until the domain name is finally decoded.

7. What encoding routine is being used to obfuscate the domain name?

This malware XORs the domain name with the string "1qaz2wsx3edc" to encode/decode it since XOR is reversible.

8. What is the significance of the CreateProcessA call at 0x0040106E?

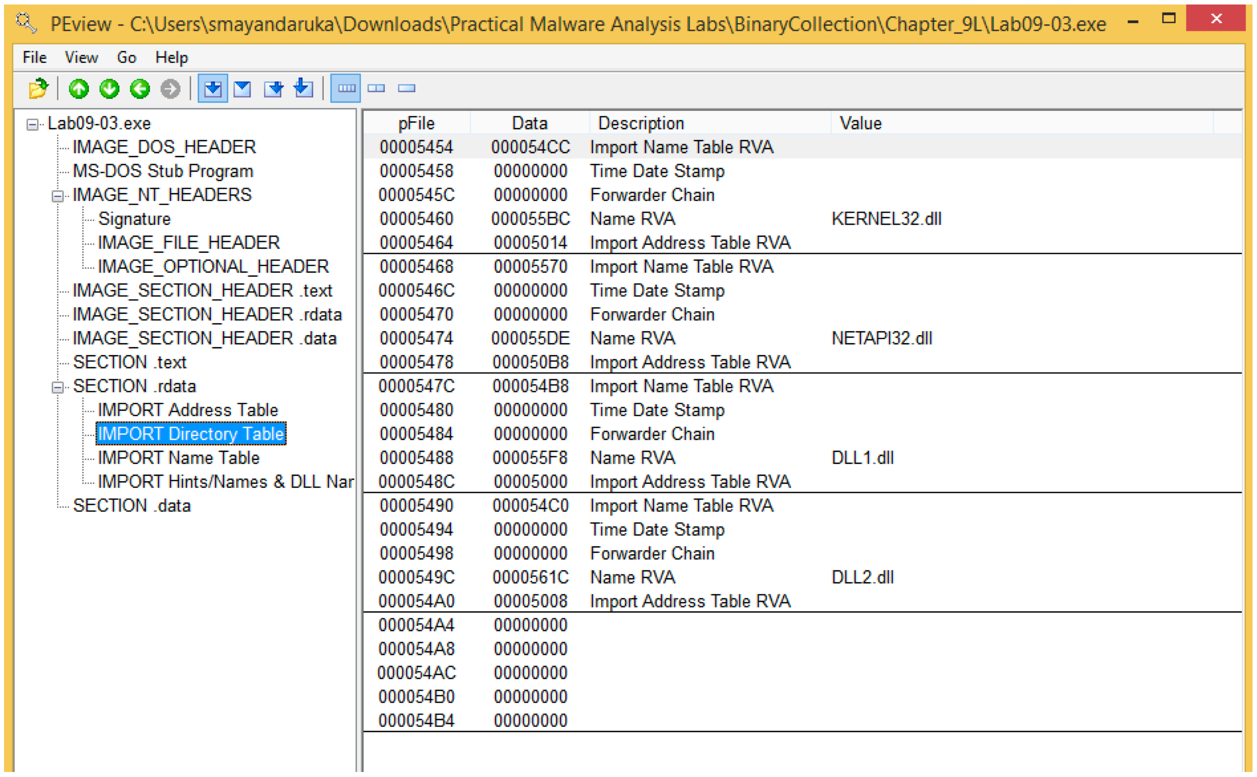
This is a really significant call since it actually creates the reverse shell and ties it back to the socket which is created at the beginning of the process call. It sets the different handles such as stdout, stderr, and stdin. One important thing to note is that the shell window is suppressed so the user doesn't see it at all.



```
00401034   C745 D4 01010000   mov dword ptr ss:[ebp-2C],101
0040103B   66:C745 D8 0000    mov word ptr ss:[ebp-28],0
00401041   8B55 18            mov edx,dword ptr ss:[ebp+18]           edx:EntryPoint
00401044   8955 E0            mov dword ptr ss:[ebp-20],edx           edx:EntryPoint
00401047   8B45 E0            mov eax,dword ptr ss:[ebp-20]
0040104A   8945 E8            mov dword ptr ss:[ebp-18],eax
0040104D   8B4D E8            mov ecx,dword ptr ss:[ebp-18]           ecx:EntryPoint
00401050   894D E4            mov dword ptr ss:[ebp-1C],ecx           ecx:EntryPoint
00401053   8D55 F0            lea edx,dword ptr ss:[ebp-10]           edx:EntryPoint
00401056   52                 push edx                                edx:EntryPoint
00401057   8D45 A8            lea eax,dword ptr ss:[ebp-58]
0040105A   50                 push eax
0040105B   6A 00              push 0
0040105D   6A 00              push 0
0040105F   6A 00              push 0
00401061   6A 01              push 1
00401063   6A 00              push 0
00401065   6A 00              push 0
00401067   68 30504000        push lab09-02.405030                    405030:"cmd"
0040106C   6A 00              push 0
0040106E   FF15 04404000      call dword ptr ds:[<&CreateProcessA>]
00401074   8945 EC            mov dword ptr ss:[ebp-14],eax
00401077   6A FF              push FFFFFFFF                  777C5B10 <kernel32.CreateProcessA>
00401079   8B4D F0            mov ecx,dword ptr ss:[ebp-     mov edi,edi
0040107C   51                 push ecx                       push ebp
0040107D   FF15 00404000      call dword ptr ds:[<&WaitF     mov ebp,esp
00401083   33C0               xor eax,eax                    pop ebp
00401085   8BE5               mov esp,ebp                    jmp dword ptr ds:[<&CreateProcessA>]
```

LAB 9-3

1. What DLLs are imported by Lab09-03.exe?



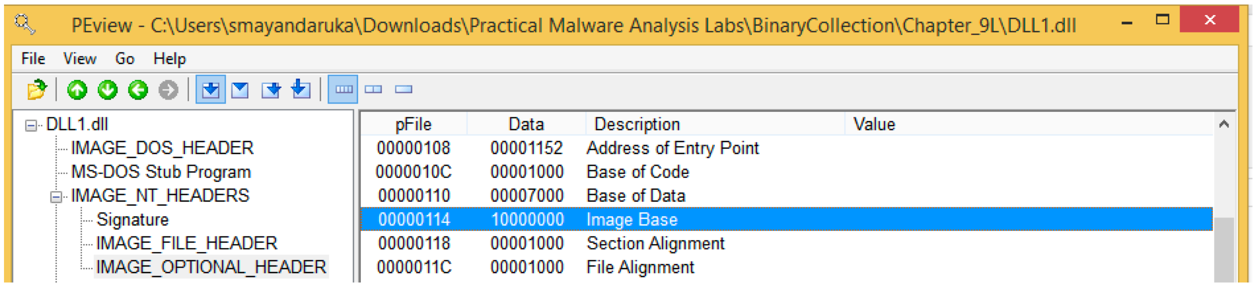
PEview - C:\Users\smayandaruka\Downloads\Practical Malware Analysis Labs\BinaryCollection\Chapter_9L\Lab09-03.exe

```
Lab09-03.exe
  IMAGE_DOS_HEADER
  MS-DOS Stub Program
  IMAGE_NT_HEADERS
    Signature
    IMAGE_FILE_HEADER
    IMAGE_OPTIONAL_HEADER
  IMAGE_SECTION_HEADER .text
  IMAGE_SECTION_HEADER .rdata
  IMAGE_SECTION_HEADER .data
  SECTION .text
  SECTION .rdata
    IMPORT Address Table
    IMPORT Directory Table
    IMPORT Name Table
    IMPORT Hints/Names & DLL Nar
  SECTION .data
```

pFile	Data	Description	Value
00005454	000054CC	Import Name Table RVA	
00005458	00000000	Time Date Stamp	
0000545C	00000000	Forwarder Chain	
00005460	000055BC	Name RVA	KERNEL32.dll
00005464	00005014	Import Address Table RVA	
00005468	00005570	Import Name Table RVA	
0000546C	00000000	Time Date Stamp	
00005470	00000000	Forwarder Chain	
00005474	000055DE	Name RVA	NETAPI32.dll
00005478	000050B8	Import Address Table RVA	
0000547C	000054B8	Import Name Table RVA	
00005480	00000000	Time Date Stamp	
00005484	00000000	Forwarder Chain	
00005488	000055F8	Name RVA	DLL1.dll
0000548C	00005000	Import Address Table RVA	
00005490	000054C0	Import Name Table RVA	
00005494	00000000	Time Date Stamp	
00005498	00000000	Forwarder Chain	
0000549C	0000561C	Name RVA	DLL2.dll
000054A0	00005008	Import Address Table RVA	
000054A4	00000000		
000054A8	00000000		
000054AC	00000000		
000054B0	00000000		
000054B4	00000000		

As can be seen above, KERNEL32.dll, NETAPI32.dll, DLL1.dll, and DLL2.dll are imported by Lab09-03.exe. USER32.dll and DLL3.dll are dynamically imported during runtime.

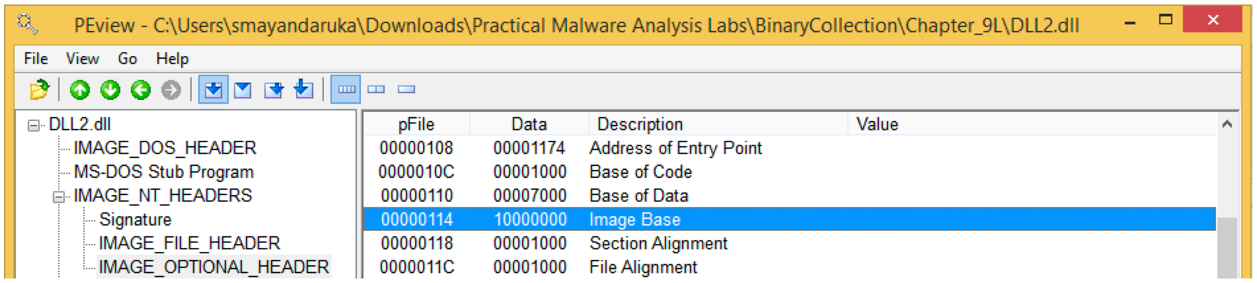
2. What is the base address requested by DLL1.dll, DLL2.dll, and DLL3.dll?


PEview - C:\Users\smayandaruka\Downloads\Practical Malware Analysis Labs\BinaryCollection\Chapter_9L\DLL1.dll

File View Go Help

DLL1.dll
- IMAGE_DOS_HEADER
- MS-DOS Stub Program
- IMAGE_NT_HEADERS
  - Signature
  - IMAGE_FILE_HEADER
  - IMAGE_OPTIONAL_HEADER

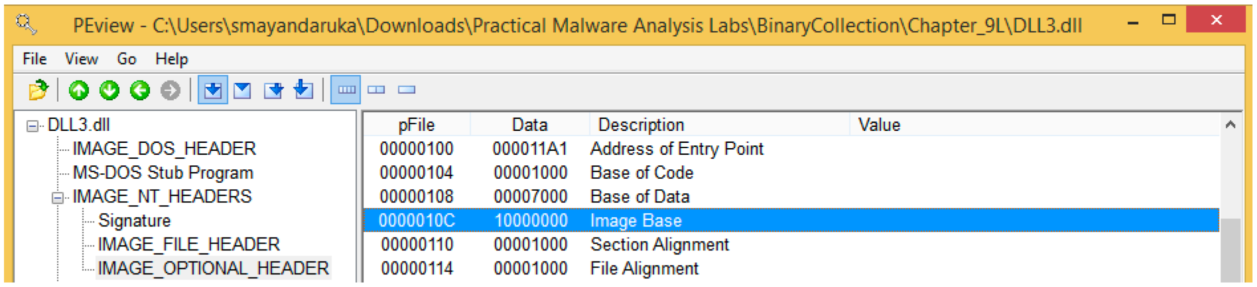
pFile	Data	Description	Value
00000108	00001152	Address of Entry Point	
0000010C	00001000	Base of Code	
00000110	00007000	Base of Data	
00000114	10000000	Image Base	
00000118	00001000	Section Alignment	
0000011C	00001000	File Alignment	


PEview - C:\Users\smayandaruka\Downloads\Practical Malware Analysis Labs\BinaryCollection\Chapter_9L\DLL2.dll

File View Go Help

DLL2.dll
- IMAGE_DOS_HEADER
- MS-DOS Stub Program
- IMAGE_NT_HEADERS
  - Signature
  - IMAGE_FILE_HEADER
  - IMAGE_OPTIONAL_HEADER

pFile	Data	Description	Value
00000108	00001174	Address of Entry Point	
0000010C	00001000	Base of Code	
00000110	00007000	Base of Data	
00000114	10000000	Image Base	
00000118	00001000	Section Alignment	
0000011C	00001000	File Alignment	


PEview - C:\Users\smayandaruka\Downloads\Practical Malware Analysis Labs\BinaryCollection\Chapter_9L\DLL3.dll

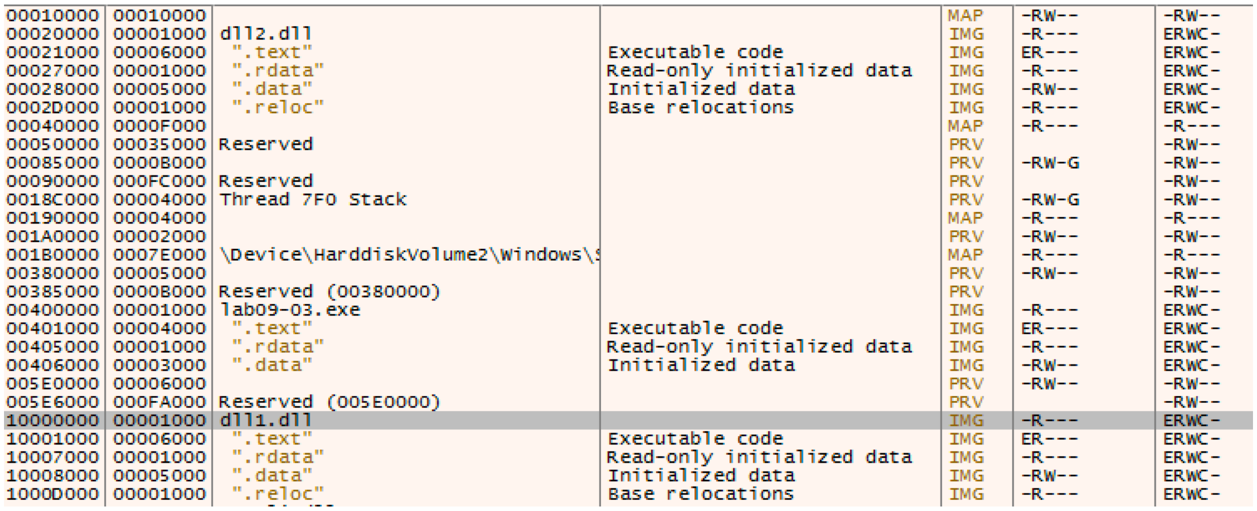
File View Go Help

DLL3.dll
- IMAGE_DOS_HEADER
- MS-DOS Stub Program
- IMAGE_NT_HEADERS
  - Signature
  - IMAGE_FILE_HEADER
  - IMAGE_OPTIONAL_HEADER

pFile	Data	Description	Value
00000100	000011A1	Address of Entry Point	
00000104	00001000	Base of Code	
00000108	00007000	Base of Data	
0000010C	10000000	Image Base	
00000110	00001000	Section Alignment	
00000114	00001000	File Alignment	

As can be seen in the screenshots above, each of the DLLs requests the same base address which is “0x10000000”.

3. When you use OllyDbg to debug Lab09-03.exe, what is the assigned based address for: DLL1.dll, DLL2.dll, and DLL3.dll?

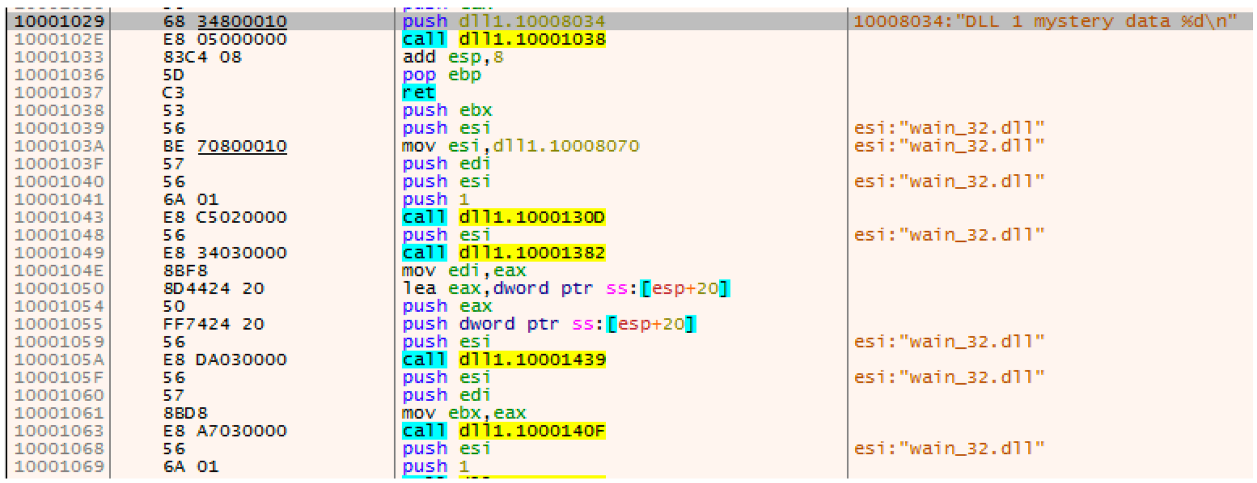


```
00010000 00010000                                                  MAP  -RW--   -RW--
00020000 00001000 dll2.dll                                         IMG  -R---   ERWC-
00021000 00006000  ".text"           Executable code               IMG  ER---   ERWC-
00027000 00001000  ".rdata"          Read-only initialized data    IMG  -R---   ERWC-
00028000 00005000  ".data"           Initialized data              IMG  -RW--   ERWC-
0002D000 00001000  ".reloc"          Base relocations              IMG  -R---   ERWC-
00040000 0000F000                                                  MAP  -R---   -R---
00050000 00035000 Reserved                                         PRV          -RW--
00085000 0000B000                                                  PRV  -RW-G   -RW--
00090000 000FC000 Reserved                                         PRV          -RW--
0018C000 00004000 Thread 7F0 Stack                                 PRV  -RW-G   -RW--
00190000 00004000                                                  MAP  -R---   -R---
001A0000 00002000                                                  PRV  -RW--   -RW--
001B0000 0007E000 \Device\HarddiskVolume2\Windows\S                MAP  -R---   -R---
00380000 00005000                                                  PRV  -RW--   -RW--
00385000 0000B000 Reserved (00380000)                              PRV          -RW--
00400000 00001000 lab09-03.exe                                     IMG  -R---   ERWC-
00401000 00004000  ".text"           Executable code               IMG  ER---   ERWC-
00405000 00001000  ".rdata"          Read-only initialized data    IMG  -R---   ERWC-
00406000 00003000  ".data"           Initialized data              IMG  -RW--   ERWC-
005E0000 00006000                                                  PRV  -RW--   -RW--
005E6000 000FA000 Reserved (005E0000)                              PRV          -RW--
10000000 00001000 dll1.dll                                         IMG  -R---   ERWC-
10001000 00006000  ".text"           Executable code               IMG  ER---   ERWC-
10007000 00001000  ".rdata"          Read-only initialized data    IMG  -R---   ERWC-
10008000 00005000  ".data"           Initialized data              IMG  -RW--   ERWC-
1000D000 00001000  ".reloc"          Base relocations              IMG  -R---   ERWC-
```

As can be seen above, the following base addresses are assigned:

- DLL1.dll – 0x10000000
- DLL2.dll – 0x00020000
- DLL3.dll – 0x00400000

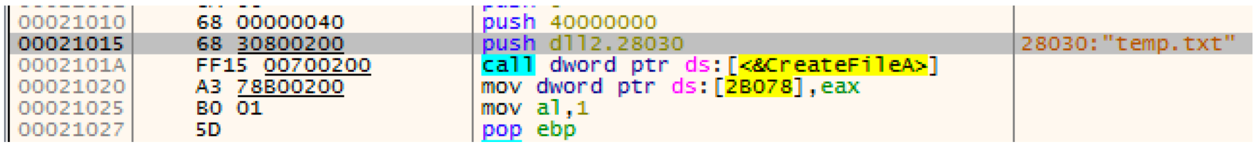
4. When Lab09-03.exe calls an import function from DLL1.dll, what does this import function do?



```
10001029  68 34800010   push dll1.10008034        10008034:"DLL 1 mystery data %d\n"
1000102E  E8 05000000   call dll1.10001038
10001033  83C4 08       add esp,8
10001036  5D            pop ebp
10001037  C3            ret
10001038  53            push ebx
10001039  56            push esi                  esi:"wain_32.dll"
1000103A  BE 70800010   mov esi,dll1.10008070     esi:"wain_32.dll"
1000103F  57            push edi
10001040  56            push esi                  esi:"wain_32.dll"
10001041  6A 01         push 1
10001043  E8 C5020000   call dll1.1000130D
10001048  56            push esi                  esi:"wain_32.dll"
10001049  E8 34030000   call dll1.10001382
1000104E  8BF8          mov edi,eax
10001050  8D4424 20     lea eax,dword ptr ss:[esp+20]
10001054  50            push eax
10001055  FF7424 20     push dword ptr ss:[esp+20]
10001059  56            push esi                  esi:"wain_32.dll"
1000105A  E8 DA030000   call dll1.10001439
1000105F  56            push esi                  esi:"wain_32.dll"
10001060  57            push edi
10001061  8BD8          mov ebx,eax
10001063  E8 A7030000   call dll1.1000140F
10001068  56            push esi                  esi:"wain_32.dll"
10001069  6A 01         push 1
```

As can be seen above, DLL1.dll calls an import function which prints “DLL 1 mystery data” to the screen and it also prints the current process ID.

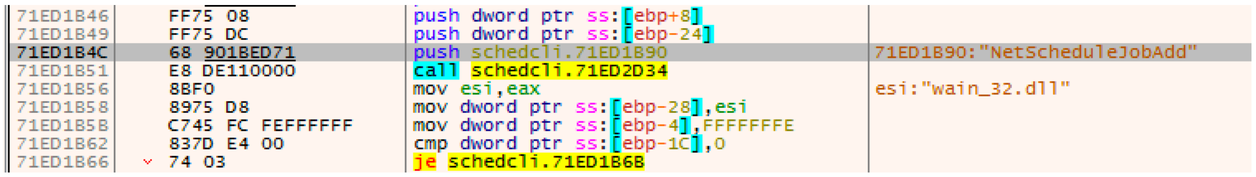
5. When Lab09-03.exe calls WriteFile, what is the filename it writes to?



```
00021010   68 00000040    push 40000000
00021015   68 30800200    push dll2.28030                      28030:"temp.txt"
0002101A   FF15 00700200  call dword ptr ds:[<&CreateFileA>]
00021020   A3 78B00200    mov dword ptr ds:[2B078],eax
00021025   B0 01          mov al,1
00021027   5D             pop ebp
```

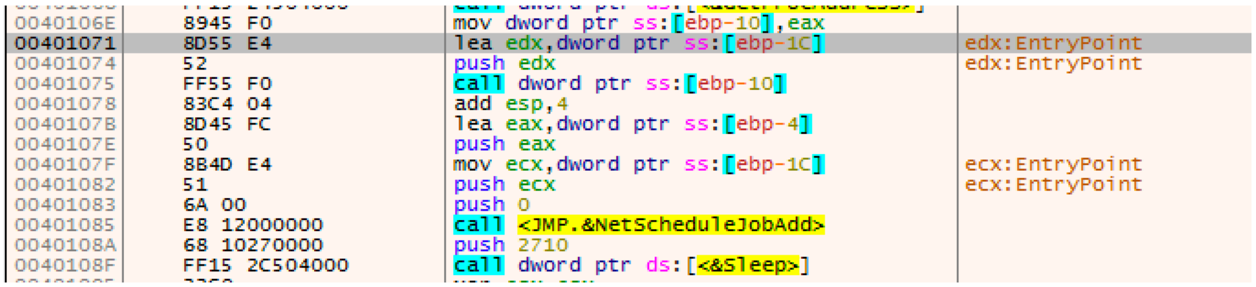
As can be seen above, WriteFile writes to “temp.txt”.

6. When Lab09-03 creates a job using NetScheduleJobAdd, where does it get the data for the second parameter?



```
71ED1B46     FF75 08             push dword ptr ss:[ebp+8]
71ED1B49     FF75 DC             push dword ptr ss:[ebp-24]
71ED1B4C     68 901BED71         push schedcli.71ED1B90        71ED1B90:"NetScheduleJobAdd"
71ED1B51     E8 DE110000         call schedcli.71ED2D34
71ED1B56     8BF0                mov esi,eax                   esi:"wain_32.dll"
71ED1B58     8975 D8             mov dword ptr ss:[ebp-28],esi
71ED1B5B     C745 FC FEFFFFFF    mov dword ptr ss:[ebp-4],FFFFFFFE
71ED1B62     837D E4 00          cmp dword ptr ss:[ebp-1C],0
71ED1B66   v 74 03               je schedcli.71ED1B6B
```

The data for the second parameter is received from DLL3GetStructure which is dynamically resolved for the call to NetScheduleJobAdd.



```
0040106E   8945 F0         mov dword ptr ss:[ebp-10],eax
00401071   8D55 E4         lea edx,dword ptr ss:[ebp-1C]       edx:EntryPoint
00401074   52              push edx                            edx:EntryPoint
00401075   FF55 F0         call dword ptr ss:[ebp-10]
00401078   83C4 04         add esp,4
0040107B   8D45 FC         lea eax,dword ptr ss:[ebp-4]
0040107E   50              push eax
0040107F   8B4D E4         mov ecx,dword ptr ss:[ebp-1C]       ecx:EntryPoint
00401082   51              push ecx                            ecx:EntryPoint
00401083   6A 00           push 0
00401085   E8 12000000     call <JMP.&NetScheduleJobAdd>
0040108A   68 10270000     push 2710
0040108F   FF15 2C504000   call dword ptr ds:[<&Sleep>]
```

7. While running or debugging the program, you will see that it prints out three pieces of mystery data. What are the following: DLL 1 mystery data 1, DLL 2 mystery data 2, and DLL 3 mystery data 3?

DLL1 mystery data 1 is the current process ID. DLL 2 mystery data 2 is a handle to the currently open temp.txt file. DLL 3 mystery data 3 is the location in memory of the string “ping www.malwareanalysisbook.com”.

8. How can you load DLL2.dll into IDA Pro so that it matches the load address used by OllyDbg?

IDA Pro gives us the ability to specify our own base address and everything is automatically offset based off of that. We can use the “Manual Load” box.