Malware Analysis
Fall 2015 — Project 1

Maxfield Chen

October 12, 2015

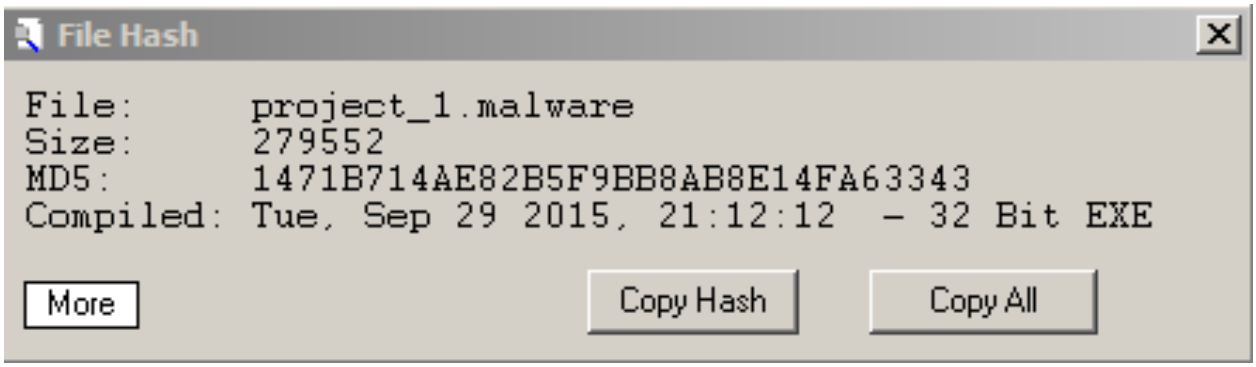
Executive Summary

1. This is a sophisticated piece of malware, capable of downloading files to the host, running arbitrary commands, deleting files and sending various system information to the attacker. In addition, this malware has sent all saved google chrome passwords to the attacker, be sure to enact appropriate security measures as soon as possible.

Basic Static Analysis

1. Purpose: To scope out the file for any anomolies which might be present, odd strings, packed files, dropped resources / executables. It's also used to suess out any previous knowledge about the file from databases or prior sources ect. All this information will be used to inform our basic dynamic analysis and futher advanced analysis.

2. File Information:

(a) 

3. Virus Total:

   (a) Detection Ratio: 3/56

   (b) Packers Identified: UPX (F-Prot)

   (c) Timestamp: 2015-09-29 21:12:12

   (d) Sections:

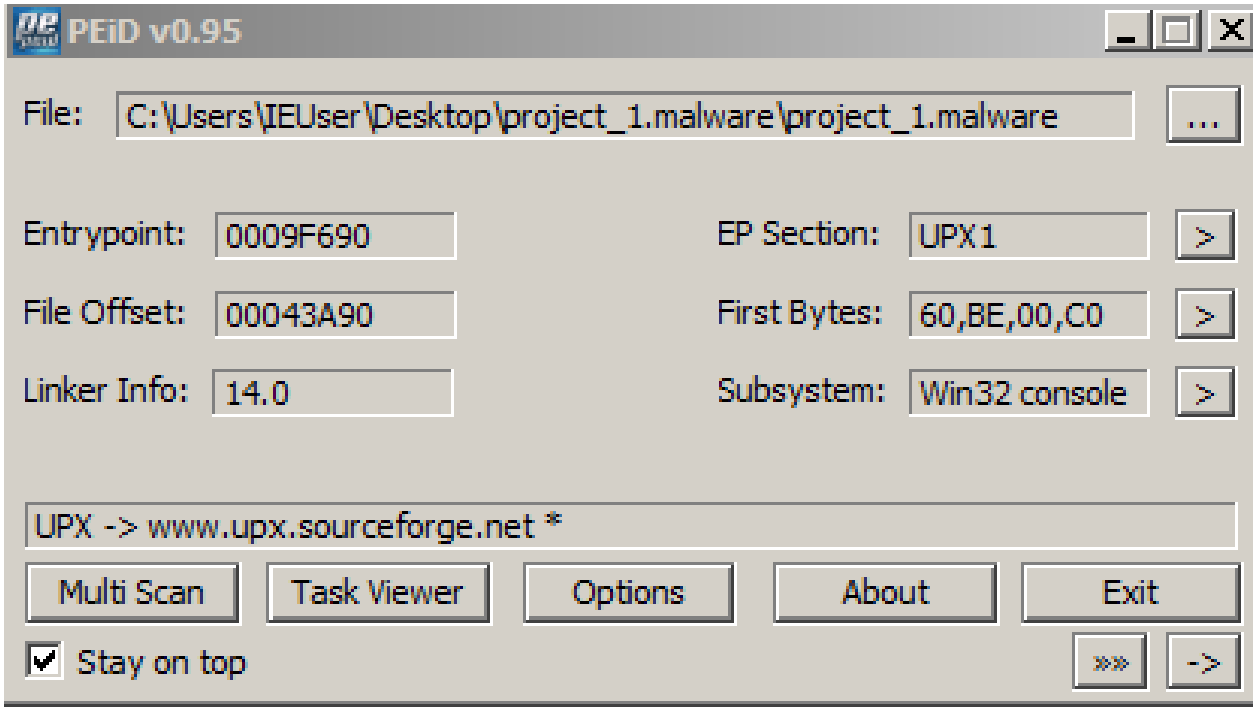
i.

PE sections

Name	Virtual address	Virtual size	Raw size	Entropy	MD5
UPY0	4096	372736	0	0.00	d41d8cd98f00b204e9800998ecf8427e
UPX1	376832	278528	276992	7.94	7aadeeacfd7d3f36971ebe522583bf1b
.rsrc	655360	4096	1536	4.06	40a002d5787a5c7af0636a4b4af3f937

ii. UPX1 has a high entropy value. Also noted that the .rsrc section is being used and that we'll want to double check for resources during the rest of analysis.

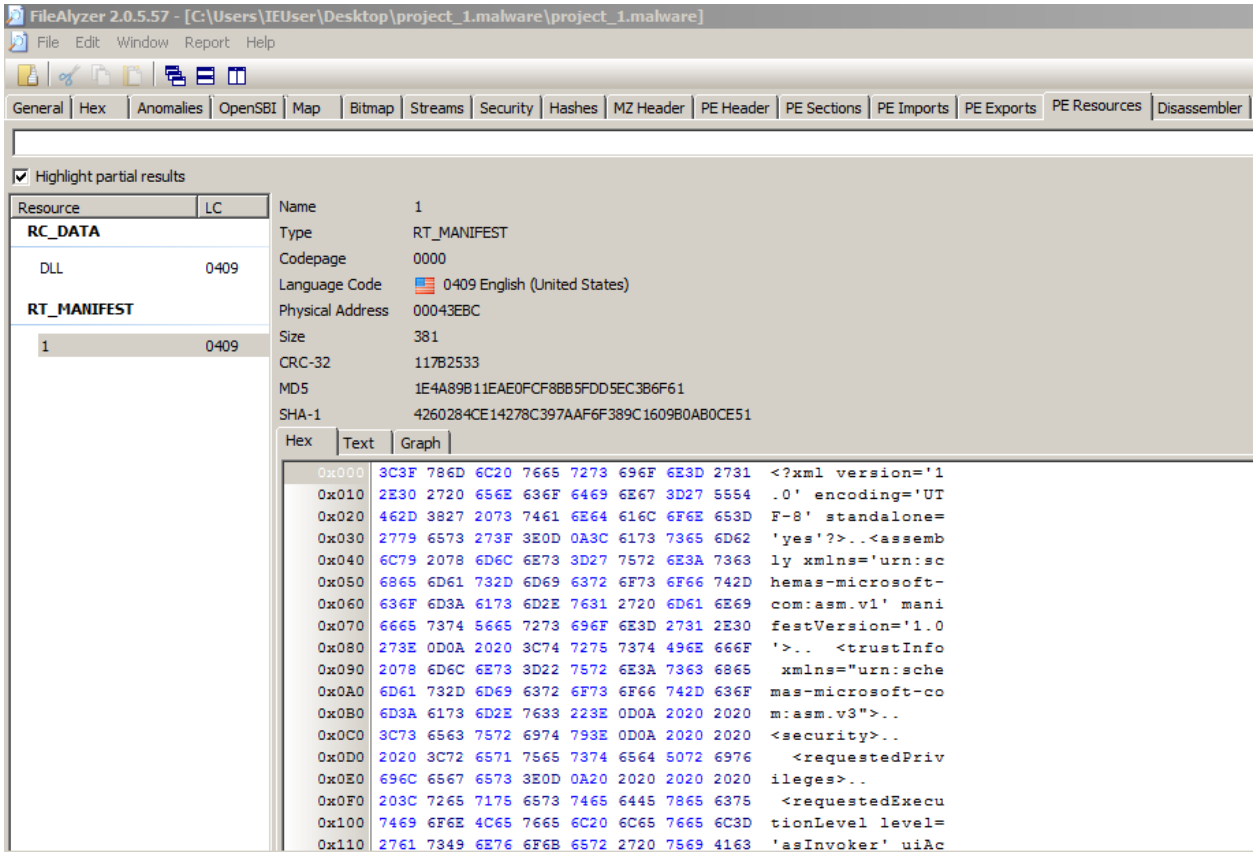
4. PEID:



(a)

(b) UPX was identified as the method of packing.

5. FileAlyzer:

(a) 

(b) It appears that this file contains a dll resource.

6. Dependency Walker:

   (a) Kernal32.dll

   (b) LoadLibraryA

   (c) GetProcAddress

7. Strings: Although this is of limited use due to the packing / resource dropping, here are a few strings which stuck out to me.

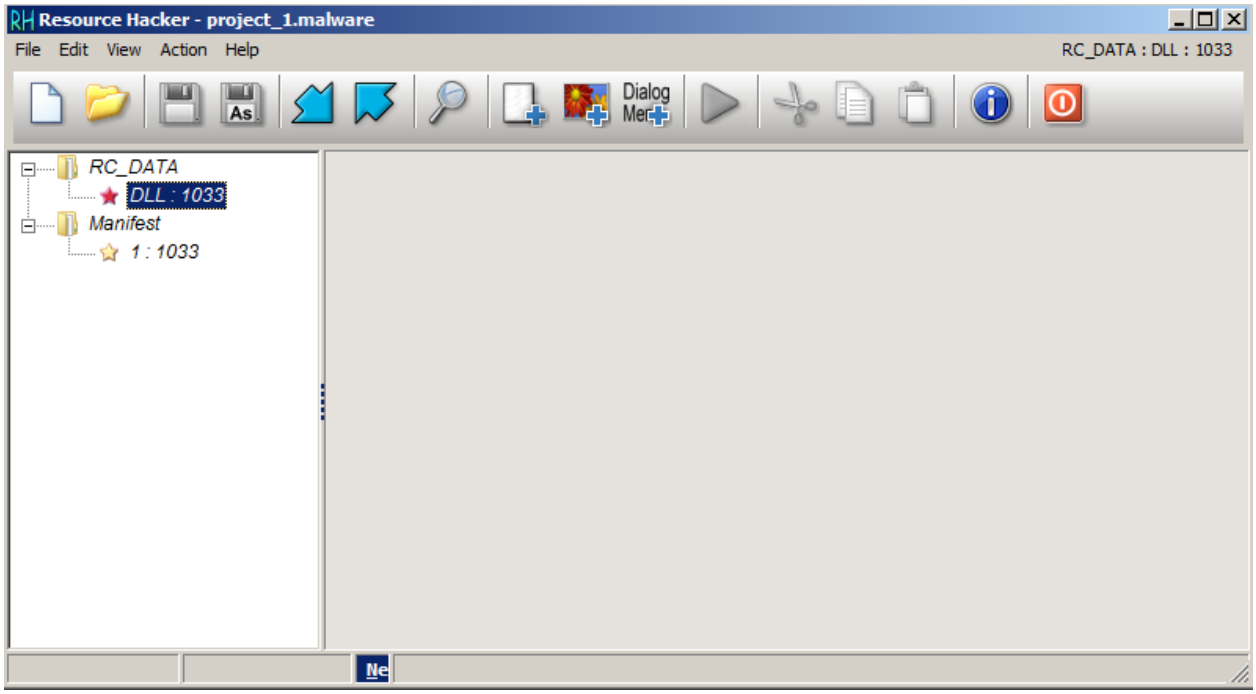
```
00043EBC  <?xml version='1.0' encoding='UTF-8' standalone='yes'?>
00043EF5  <assembly xmlns='urn:schemas-microsoft-com:asm.v1' manifestVersion='1.0'>
00043F40    <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
00043F78      <security>
00043F88        <requestedPrivileges>
00043FA5          <requestedExecutionLevel level='asInvoker' uiAccess='false' />
00043FED        </requestedPrivileges>
0004400B      </security>
0004401C    </trustInfo>
0004402C  </assembly>
00044128  KERNEL32.DLL
00044135  api-ms-win-crt-heap-l1-1-0.dll
00044154  api-ms-win-crt-locale-l1-1-0.dll
00044175  api-ms-win-crt-math-l1-1-0.dll
00044194  api-ms-win-crt-runtime-l1-1-0.dll
000441B6  api-ms-win-crt-stdio-l1-1-0.dll
000441D6  VCRUNTIME140.dll
000441E8  LoadLibraryA
000441F7  etProcAddress
00044206  VirtualProtect
00044216  VirtualAlloc
00044224  VirtualFree
00044232  ExitProcess
00044240  _set_new_mode
00044250  _configthreadlocale
00044266  __setusermatherr
00044278  exit
0004427E  _set_fmode
0004428A  memset
```

(a) i.

(b) Here we can see a few imports which were included in the import list and the xml. In this instance the xml doesn't reveal any further information to us.

8. Resource Hacker:

(a)



(b) This is the same information we found using fileAlyzer.

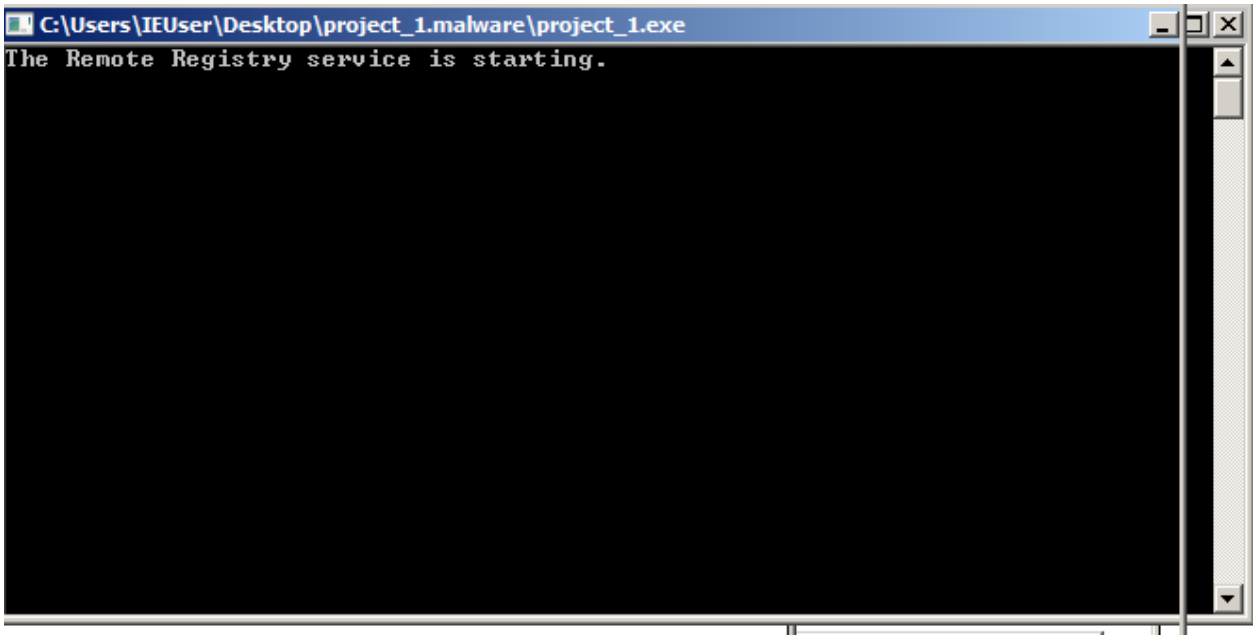
9. Conclusion: It's clear from the peid and lack of clean strings that this file has been packed at least once using standard UPX. Additionally, when checking through resources we found a dll file which will likely need to be dropped after we unpack the malware.

Basic Dynamic Analysis

10. Purpose: We can run the program and observe the actions it takes and the effects it has on our system in order to get a better idea of its functionality and scope. This will also give us context when we proceed to more advanced static analysis. Finally, by analyzing the effects it has on our system, we can start to develop host based signatures.

11. Program:

    (a) There is a quick command prompt which flashes onto the screen with the following text:

    (b) 

12. FakeNet:

    (a) Fakenet detected the malware reaching out to malcode.rpis.ec

    (b) 

13. Regshot:

    (a) Regshot detected 22 key modifications and 1 new value. We will look to confirm if these changes are harmless windows operations or malware actions as we filter through the Proccess Monitor logs.

(b)

```
Regshot 1.9.0 x86 ANSI
Comments:
Datetime: 2015/10/12 15:38:07  ,  2015/10/12 15:38:50
Computer: MALWARE , MALWARE
Username: User , User

----------------------------------
Values added: 1
----------------------------------
HKU\S-1-5-21-3463664321-2923530833-3546627382-1000\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\{CEBFF5CD-
ACE2-4F4F-9178-9926F41749EA}\Count\P:\Hfref\VRHfre\Qrfxgbc\cebwrpg_1.znyjner\cebwrpg_1.rkr:  00 00 00 00 01 00 00 00 00 00 00 00 AB 08 00 00 00 00 80 BF 00 00 80 BF 00 00 80
BF 00 00 80 BF 00 00 80 BF 00 00 80 BF 00 00 80 BF 00 00 80 BF 00 00 80 BF FF FF FF FF 30 BF 10 0B 04 05 D1 01 00 00 00 00

----------------------------------
Values modified: 22
----------------------------------
```

(c)

```
HKLM\SYSTEM\ControlSet001\services\RemoteRegistry\DisplayName: "@regsvc.dll,-1"
HKLM\SYSTEM\ControlSet001\services\RemoteRegistry\DisplayName: "VT-x Extension"
HKLM\SYSTEM\ControlSet001\services\RemoteRegistry\ImagePath: "%SystemRoot%\system32\svchost.exe -k regsvc"
HKLM\SYSTEM\ControlSet001\services\RemoteRegistry\ImagePath: "C:\Windows\System32\svchost.exe -k regsvc"
HKLM\SYSTEM\ControlSet001\services\RemoteRegistry\Description: "@regsvc.dll,-2"
HKLM\SYSTEM\ControlSet001\services\RemoteRegistry\Description: "Maintains the virtual comaptability interface extension."
HKLM\SYSTEM\ControlSet001\services\RemoteRegistry\ObjectName: "NT AUTHORITY\LocalService"
HKLM\SYSTEM\ControlSet001\services\RemoteRegistry\ObjectName: "LocalSystem"
HKLM\SYSTEM\ControlSet001\services\RemoteRegistry\Start: 0x00000003
HKLM\SYSTEM\ControlSet001\services\RemoteRegistry\Start: 0x00000002
HKLM\SYSTEM\ControlSet001\services\RemoteRegistry\DependOnService:  52 50 43 53 53 00 00
HKLM\SYSTEM\ControlSet001\services\RemoteRegistry\DependOnService: "rpcss"
HKLM\SYSTEM\ControlSet001\services\RemoteRegistry\Parameters\ServiceDll: "%SystemRoot%\system32\regsvc.dll"
HKLM\SYSTEM\ControlSet001\services\RemoteRegistry\Parameters\ServiceDll: "C:\Windows\VTxExt.dll"
HKLM\SYSTEM\CurrentControlSet\services\RemoteRegistry\DisplayName: "@regsvc.dll,-1"
HKLM\SYSTEM\CurrentControlSet\services\RemoteRegistry\DisplayName: "VT-x Extension"
HKLM\SYSTEM\CurrentControlSet\services\RemoteRegistry\ImagePath: "%SystemRoot%\system32\svchost.exe -k regsvc"
HKLM\SYSTEM\CurrentControlSet\services\RemoteRegistry\ImagePath: "C:\Windows\System32\svchost.exe -k regsvc"
HKLM\SYSTEM\CurrentControlSet\services\RemoteRegistry\Description: "@regsvc.dll,-2"
HKLM\SYSTEM\CurrentControlSet\services\RemoteRegistry\Description: "Maintains the virtual comaptability interface extension."
HKLM\SYSTEM\CurrentControlSet\services\RemoteRegistry\ObjectName: "NT AUTHORITY\LocalService"
HKLM\SYSTEM\CurrentControlSet\services\RemoteRegistry\ObjectName: "LocalSystem"
HKLM\SYSTEM\CurrentControlSet\services\RemoteRegistry\Start: 0x00000003
HKLM\SYSTEM\CurrentControlSet\services\RemoteRegistry\Start: 0x00000002
HKLM\SYSTEM\CurrentControlSet\services\RemoteRegistry\DependOnService:  52 50 43 53 53 00 00
HKLM\SYSTEM\CurrentControlSet\services\RemoteRegistry\DependOnService: "rpcss"
HKLM\SYSTEM\CurrentControlSet\services\RemoteRegistry\Parameters\ServiceDll: "%SystemRoot%\system32\regsvc.dll"
HKLM\SYSTEM\CurrentControlSet\services\RemoteRegistry\Parameters\ServiceDll: "C:\Windows\VTxExt.dll"
```

(d) Here we see some of the suspicious key values changed. Again, we will look to confirm these in the next step.

14. Proccess Monitor:

(a)

⊟ System32	0.0023538	171	44	38	0	0	0	0	4	0	85
KernelBase.dll	0.0000077	1	0	0	0	0	0	0	0	0	1
advapi32.dll	0.0001197	1	0	0	0	0	0	0	0	0	1
api-ms-win-appmodel-runtime-l1-1-1.DLL	0.0000150	1	1	0	0	0	0	0	0	0	0
api-ms-win-core-fibers-l1-1-1.DLL	0.0001482	3	3	0	0	0	0	0	0	0	0
api-ms-win-core-file-l1-2-0.dll	0.0000134	8	2	2	0	0	0	0	0	0	4
api-ms-win-core-file-l2-1-0.dll	0.0000572	8	2	2	0	0	0	0	0	0	4
api-ms-win-core-localization-l1-2-0.dll	0.0000672	8	2	2	0	0	0	0	0	0	4
api-ms-win-core-localization-l1-2-1.DLL	0.0000150	1	1	0	0	0	0	0	0	0	0
api-ms-win-core-processthreads-l1-1-1.dll	0.0000448	8	2	2	0	0	0	0	0	0	4
api-ms-win-core-synch-l1-2-0.dll	0.0000702	8	2	2	0	0	0	0	0	0	4
api-ms-win-core-timezone-l1-1-0.dll	0.0000823	8	2	2	0	0	0	0	0	0	4
api-ms-win-crt-convert-l1-1-0.dll	0.0000473	8	2	2	0	0	0	0	0	0	4
api-ms-win-crt-heap-l1-1-0.dll	0.0004998	8	2	2	0	0	0	0	0	0	4
api-ms-win-crt-locale-l1-1-0.dll	0.0000513	8	2	2	0	0	0	0	0	0	4
api-ms-win-crt-math-l1-1-0.dll	0.0000502	8	2	2	0	0	0	0	0	0	4
api-ms-win-crt-runtime-l1-1-0.dll	0.0000591	8	2	2	0	0	0	0	0	0	4
api-ms-win-crt-stdio-l1-1-0.dll	0.0000853	8	2	2	0	0	0	0	0	0	4
api-ms-win-crt-string-l1-1-0.dll	0.0000909	8	2	2	0	0	0	0	0	0	4
apisetschema.dll	0.0000069	1	0	0	0	0	0	0	0	0	1
apphelp.dll	0.0000469	8	2	2	0	0	0	0	0	0	4
cmd.exe	0.0003538	22	4	4	0	0	0	0	4	0	10
ext-ms-win-kernel32-package-current-l1-1-0.l	0.0002060	1	1	0	0	0	0	0	0	0	0
kernel32.dll	0.0000065	1	0	0	0	0	0	0	0	0	1
msvcrt.dll	0.0000064	1	0	0	0	0	0	0	0	0	1
ntdll.dll	0.0000065	1	0	0	0	0	0	0	0	0	1
rpcrt4.dll	0.0000065	1	0	0	0	0	0	0	0	0	1
sechost.dll	0.0000480	8	2	2	0	0	0	0	0	0	4
ucrtbase.dll	0.0000567	8	2	2	0	0	0	0	0	0	4
vcruntime140.dll	0.0000850	8	2	2	0	0	0	0	0	0	4
VTxExt.dll	0.0005289	3	1	1	0	1	0	622,592	0	0	0

(b) Here we can see the unpacked libraries the program uses. Lots of core windows libraries, but also of note is the VTxExt.dll. It has an enormous number of writebacks; a quick google search confrims that this is not a standard windows dll. This is likely the resource we saw being dropped in our basic static analysis.

(c)

8:38:2...	project_1.exe	3548	Process Create	C:\Windows\system32\cmd.exe	SUCCESS	PID: 2332, Comma...

(d)

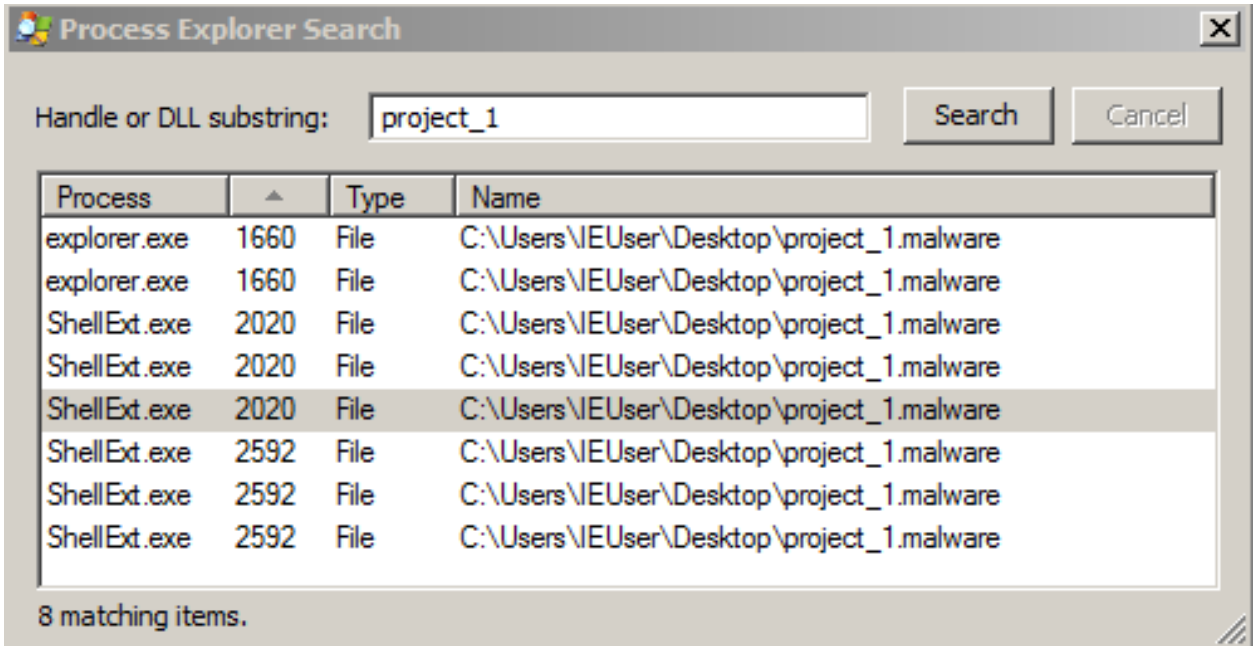
8:38:2...	cmd.exe	2332	RegOpenKey	HKCU\Control Panel\Desktop\Languag...	SUCCESS	Desired Access: R...
8:38:2...	cmd.exe	2332	RegEnumValue	HKCU\Control Panel\Desktop\Languag...	NO MORE ENTRI...	Index: 0, Length: 5...
8:38:2...	cmd.exe	2332	RegCloseKey	HKCU\Control Panel\Desktop\Languag...	SUCCESS	

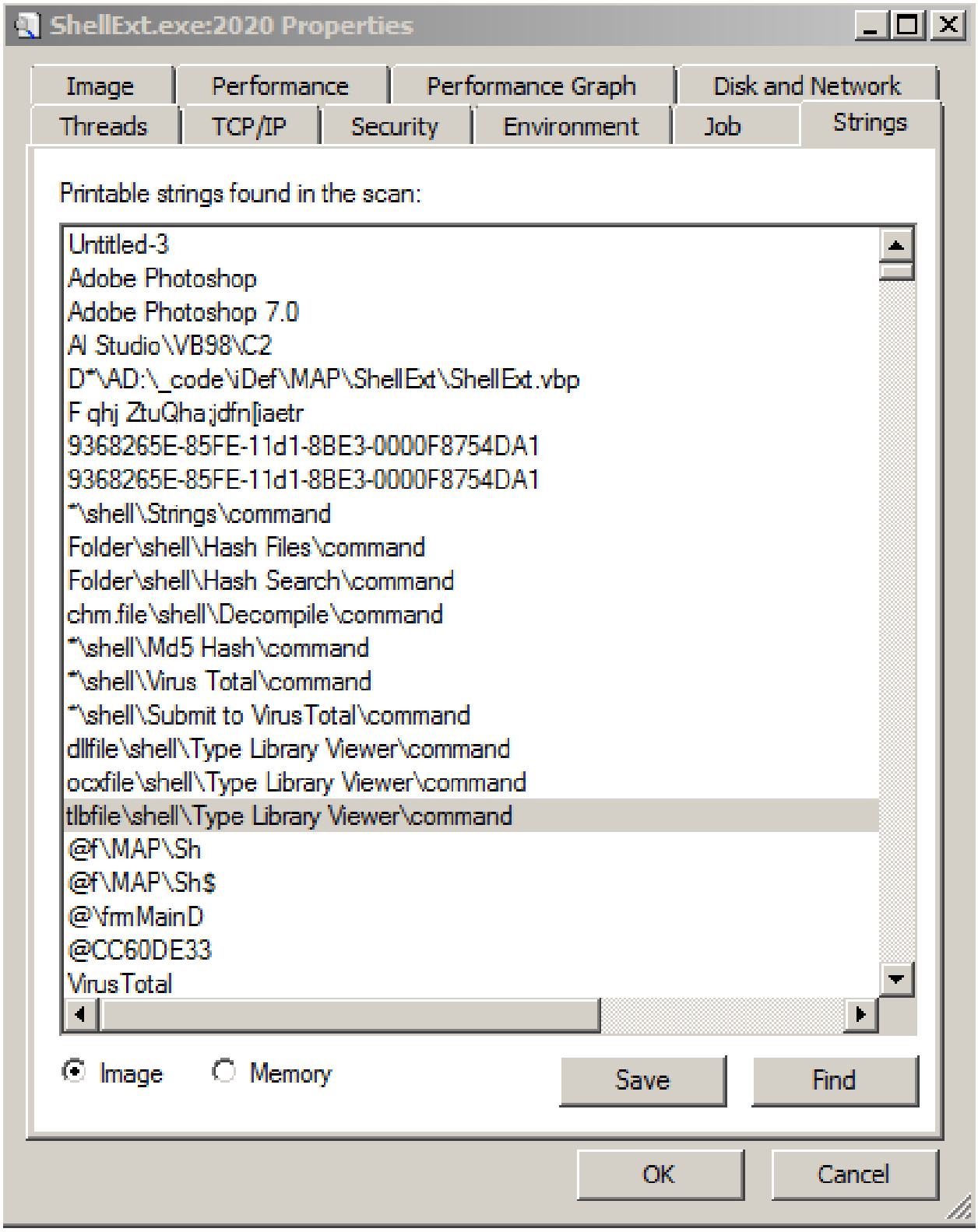
(e) Another interesting point to note is that this program spawns a command line which performs a number of other registry key checks. Among them, it appears to be checking the language of the machine. Perhaps this malware takes different actions depending on the default language present?

(f) The last thing to note is that as we look to confirm the registry changes we saw in Regshot, that an operations filter for setRegValue turns up no results. It appears as though this malware does not employ a registry based peristance scheme and those change we saw were harmless.

15. Proccess Explorer

(a)





(b)

(c) Here we have some very interesting data, it seems like our malware has spawned off

additional sub processes. It has made some trivial effort to hide them by using the company name blah.com, but clearly this is suspicious. Upon investigating the strings for these spawned processes, we find a whole host of interesting elements. It appears that it is scraping some information from the current setup, it contains some of the dialog boxes I had open in other windows. This suggests that the program is gathering system information to send remotely to an attacker. It is possible that this program is some sort of sophisticated remote shell / reconissance system. ShellExt is a subprocess of explorer.exe which suggests a potential persistance method.

16. Conclusion: Although there is still some uncertainty, we now know that this is a sophisticated piece of malware. We know it has a network component, we know that it spawns a command line process which checks a plethora of registry keys (some of which relate to language), we know that it drops a dll(VTxExt.dll), and finally, we know that it spawns a sub-process(1 per run; no mutex) which hides as ShellExt.exe. This spawned process may be related to the dll as they seem to share the same name.

Unpacking

17. Purpose: The file is packed, in order to continue with advanced static analysis, we have to unpack it.

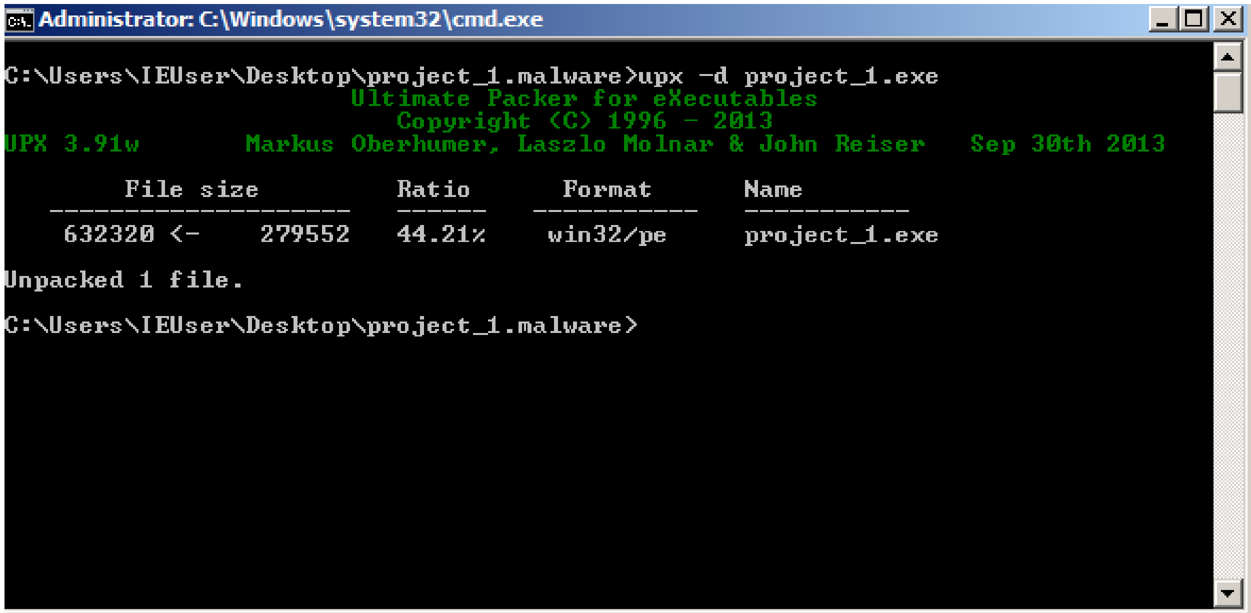
18. UPX Command Line Auto Unpacker:

    (a) Something has been done to make this file unpackable with the standard tool, we'll have to examine the file to see if there's a way to fix it and if not, then we'll manually unpack it.

    (b)

```
000001B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001D0  6C F8 09 00 5C 00 00 00 00 00 00 00 00 00 00 00  lø..\...........
000001E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000001F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000200  55 50 58 30 00 00 00 00 00 B0 05 00 00 10 00 00  UPX0.....°......
00000210  00 00 00 00 00 04 00 00 00 00 00 00 00 00 00 00  ................
00000220  00 00 00 00 80 00 00 E0 55 50 58 31 00 00 00 00  ....€..àUPX1....
00000230  00 40 04 00 00 C0 05 00 00 3A 04 00 00 04 00 00  .@...À...:......
00000240  00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 E0  ............@..à
```

    (c) When examining the section headers, we see that one of them is UYX instead of UPX. A quick change in hXD remedies this change and allows us to continue unpacking.

(d) 

(e) Success!

Advanced Static Analysis

19. Purpose: To further explore the functionalities of the program at the code level. We will use IDA to discover all the major components of this program. This will greatly inform the work we need to complete in advanced dynamic analysis.

20. Strings: Unpacked Pass

    (a) I felt obligated to take a second look at our unpacked strings.

    (b)
```
0008BF20  CREATE TEMP TABLE sqlite_temp_master(
0008BF46    type text,
0008BF53    name text,
0008BF60    tbl_name text,
0008BF71    rootpage integer,
0008BF85    sql text
0008C008  onoffalseyestruefull
0008C8C9  !5Ng
0008C9C0  0123456789ABCDEF0123456789abcdef
0008CAA1  ******""""""""""""""""""""""""
0008CAC0
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@3.8.11.1
0008D208   !"#$%&'()*+,-./0123456789:;<=>?@abcdefghijklmnopqrstuvwxyz[\]
```

    (c) Here are some indicators that this program will be storing data, the SQL above means that it will likely be interfacing with an external database.

```
0008B3CE  EC:\Windows\System32\svchost.exe -k regsvc
0008B424  VT-x Extension
0008B444  RemoteRegistry
0008B468  SYSTEM\CurrentControlSet\Services\RemoteRegistry
0008B4CC  ImagePath
0008B4E0  C:\Windows\System32\svchost.exe -k regsvc
0008B534  ErrorControl
0008B550  C:\Windows\System32\svchost.exe -k regsvc
```

(d)

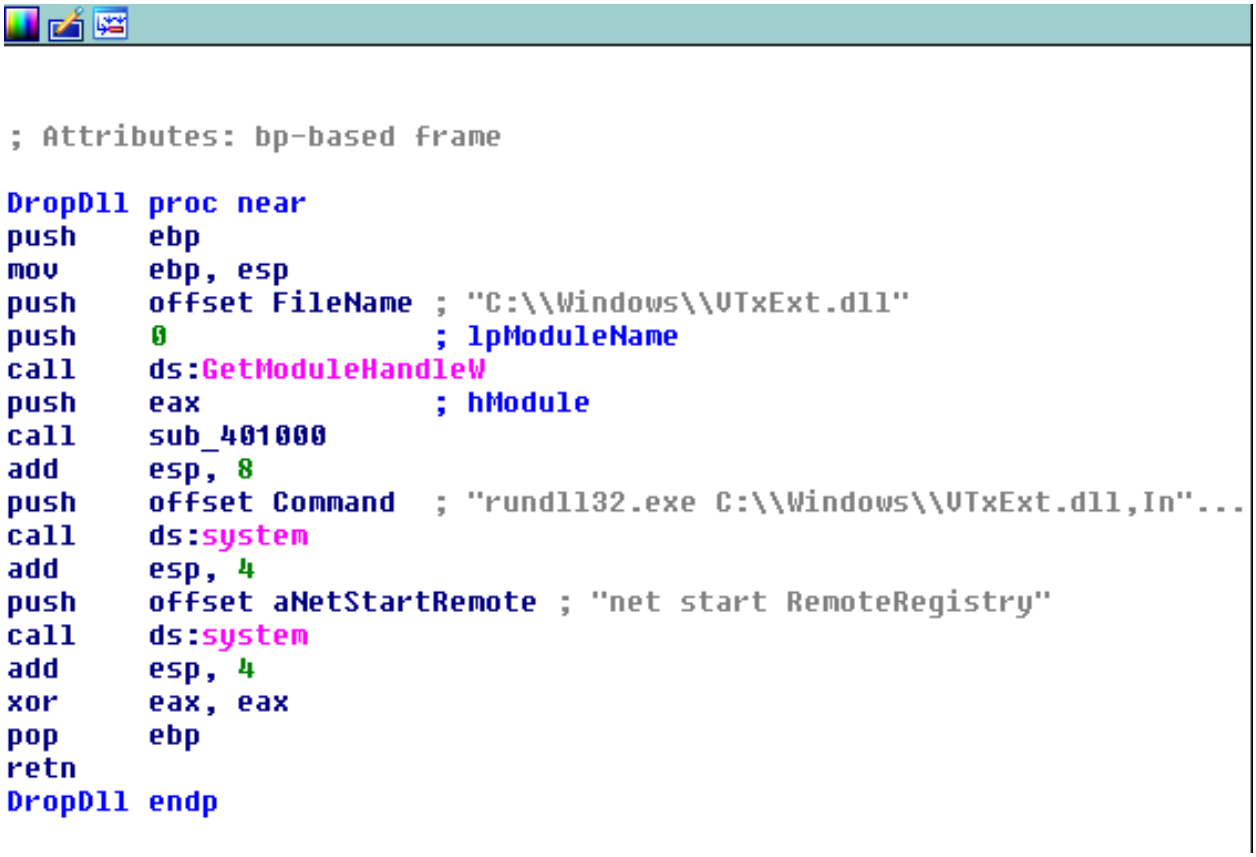
(e) There are also more references to Windows Remote Registry. Googling this reveals that this service allows registry keys to be changed remotely!

(f) Further interesting Strings:

i. rundll32.exe C:/Windows/VTxExt.dll,InstallService

ii. net start RemoteRegistry

21. IDA Code Coverage:



```
; Attributes: bp-based frame

DropDll proc near
push    ebp
mov     ebp, esp
push    offset FileName ; "C:\\Windows\\VTxExt.dll"
push    0               ; lpModuleName
call    ds:GetModuleHandleW
push    eax             ; hModule
call    sub_401000
add     esp, 8
push    offset Command  ; "rundll32.exe C:\\Windows\\VTxExt.dll,In"...
call    ds:system
add     esp, 4
push    offset aNetStartRemote ; "net start RemoteRegistry"
call    ds:system
add     esp, 4
xor     eax, eax
pop     ebp
retn
DropDll endp
```

(a)

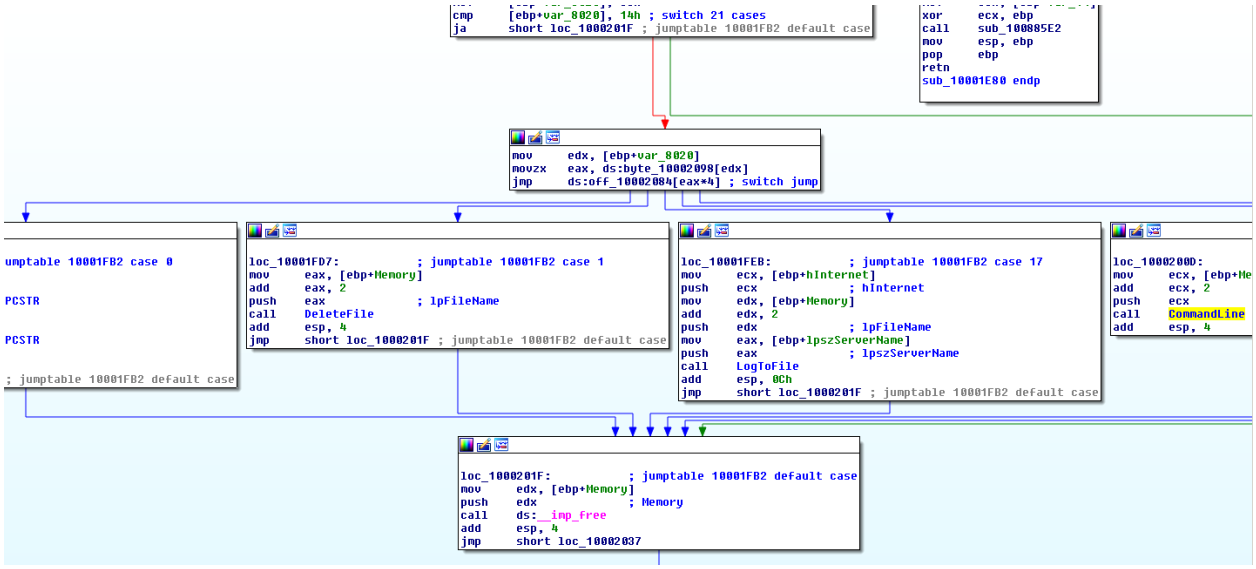
(b) This section of the code shows the malware running the dll it previously dropped using rundll32.exe and then starting the remote registry service. From here we will investigate the malware through the VTxExt.dll

(c)

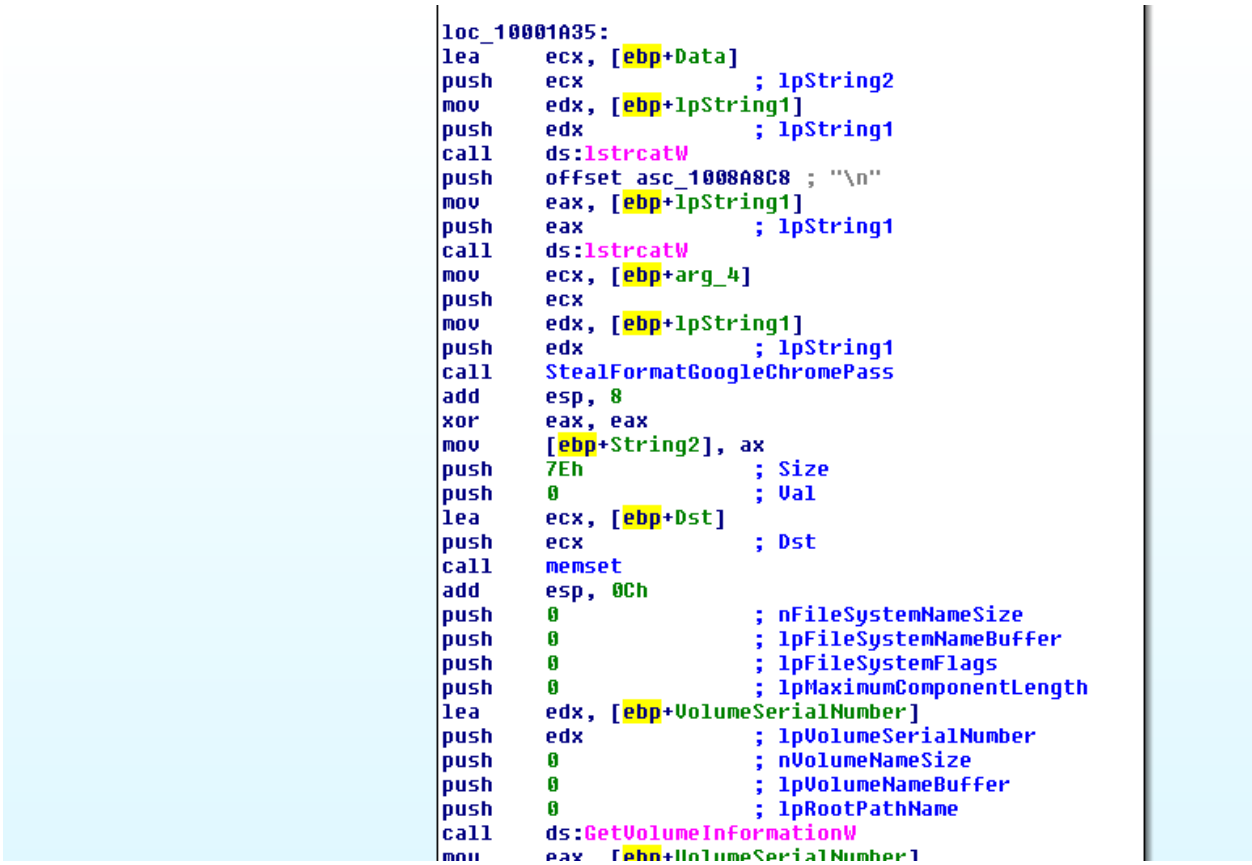
(d) Here we can see the two primary exports of this dll, InstallService, and ServiceMain.

(e) When examining ServiceMain, it becomes clear that there is one major function responsible for most of the functionality.

(f)



(g) Here we find the meat of the program. We have a loop which checks to see if it could make connection to the malicious site, queries with a GET request for which command should be executed, a small function to extract system information(drive space / language / time) and extracts passwords from the google chrome SQL database. All of that information is passed onto the attacker on each iteration. The previously mentioned get request corresponds to a switch statement with four different commands. DeleteFile, DownloadFile, SendDataToAttacker, and a command line. This provides the attacker with a whole host of convenient functionality and a discreet way to access it.

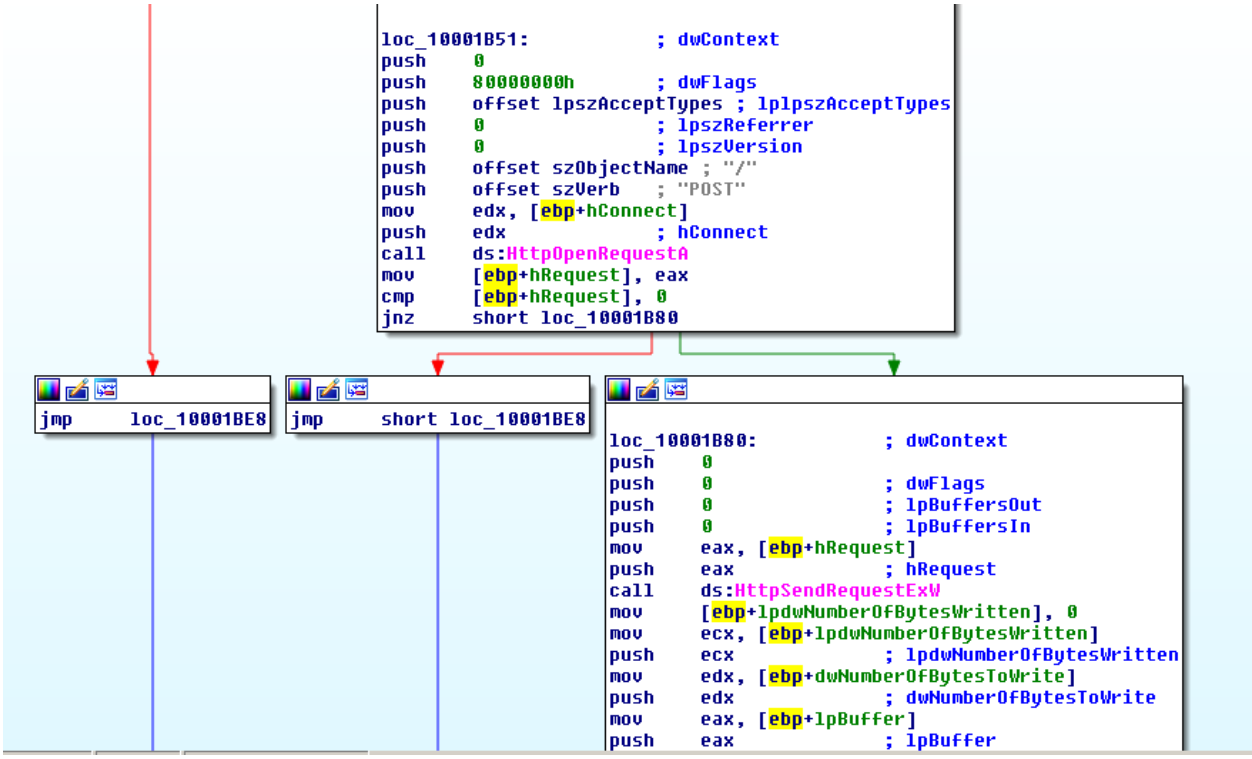
i. 

```
loc_10001A35:
lea     ecx, [ebp+Data]
push    ecx             ; lpString2
mov     edx, [ebp+lpString1]
push    edx             ; lpString1
call    ds:lstrcatW
push    offset asc_1008A8C8 ; "\n"
mov     eax, [ebp+lpString1]
push    eax             ; lpString1
call    ds:lstrcatW
mov     ecx, [ebp+arg_4]
push    ecx
mov     edx, [ebp+lpString1]
push    edx             ; lpString1
call    StealFormatGoogleChromePass
add     esp, 8
xor     eax, eax
mov     [ebp+String2], ax
push    7Eh             ; Size
push    0               ; Val
lea     ecx, [ebp+Dst]
push    ecx             ; Dst
call    memset
add     esp, 0Ch
push    0               ; nFileSystemNameSize
push    0               ; lpFileSystemNameBuffer
push    0               ; lpFileSystemFlags
push    0               ; lpMaximumComponentLength
lea     edx, [ebp+VolumeSerialNumber]
push    edx             ; lpVolumeSerialNumber
push    0               ; nVolumeNameSize
push    0               ; lpVolumeNameBuffer
push    0               ; lpRootPathName
call    ds:GetVolumeInformationW
mov     eax, [ebp+VolumeSerialNumber]
```

ii. This is the system data and google chrome logging function. It sends all this information on every iteration of the loop.

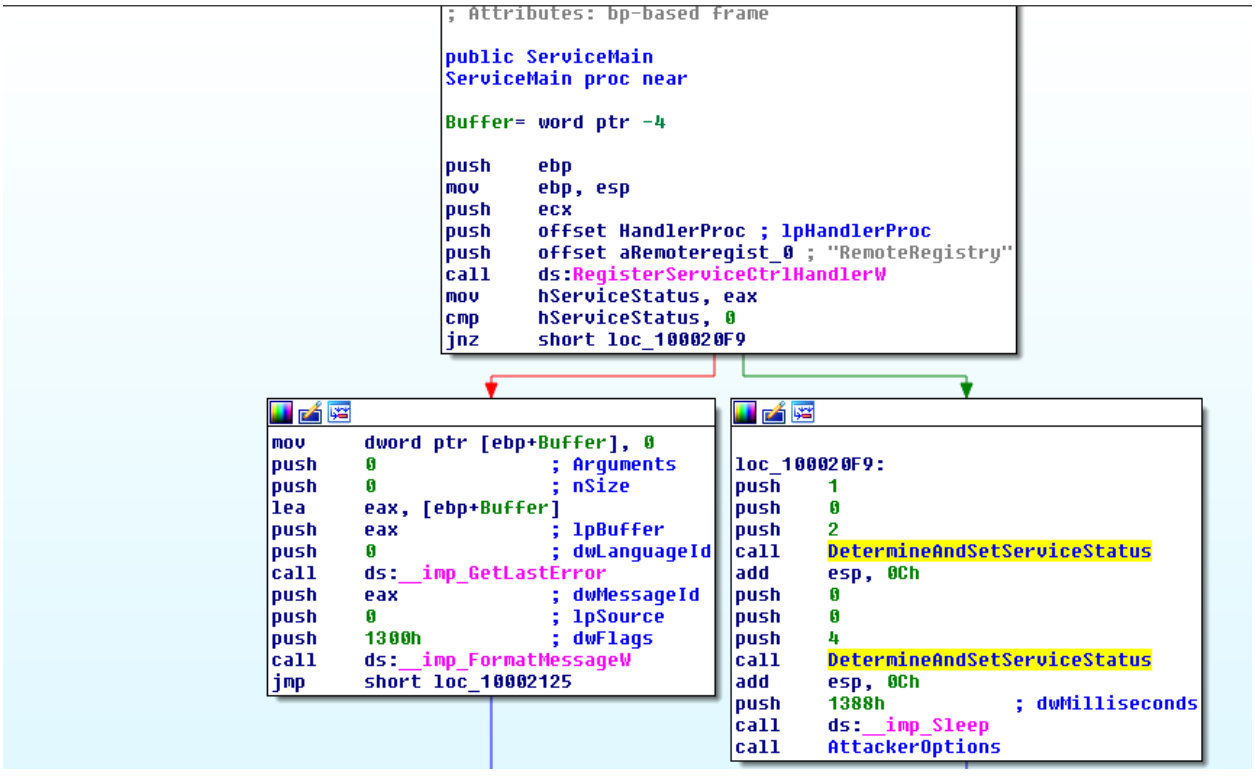
iii. DeleteFile, DownloadFile and CommandLine do exactly what you would expect.

iv.



v. SendDataToAttacker takes a file name and sends that file to the attacker using POST data.

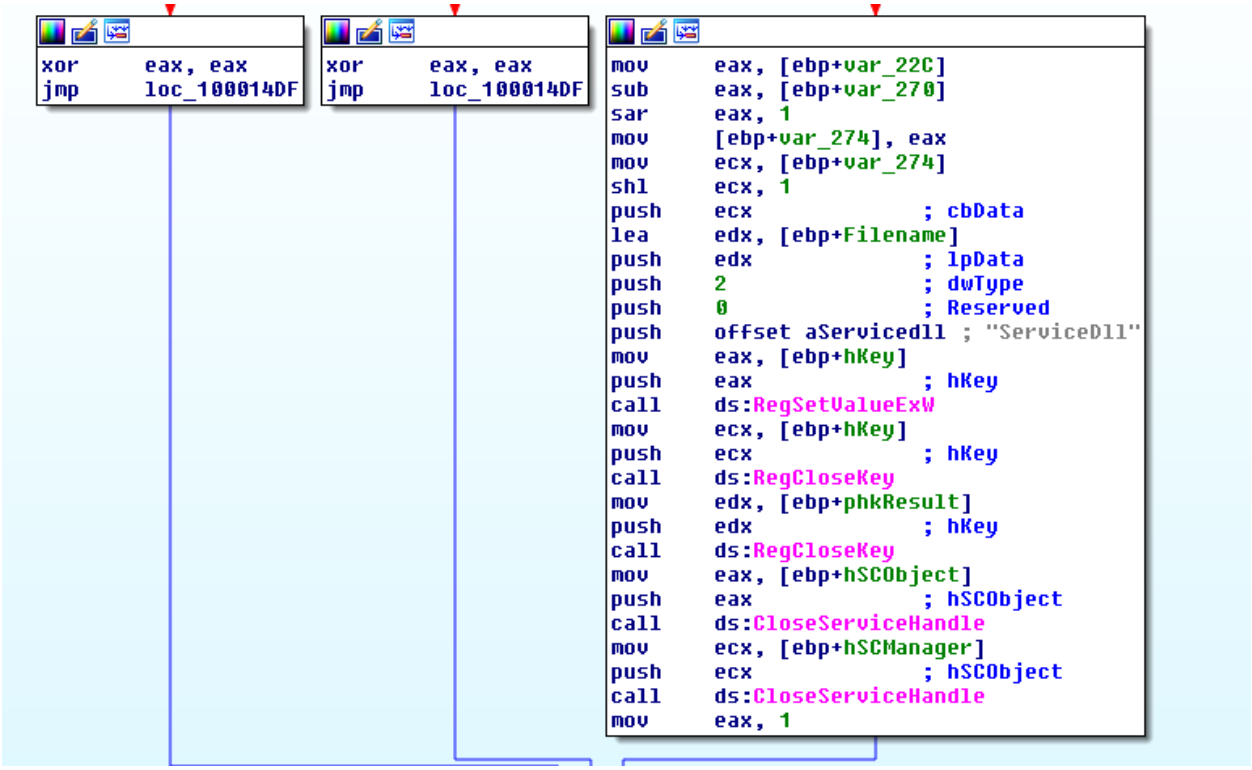
(h)



(i) This is one more way the attacker subjucates the remoteRegistry requests with this

program. It uses RegisterServiceCtrlHandlerW to register the above function to handle incoming requests for the remote registry service. Using this, the attacker can discreetly execute the above functionality through a seemingly legitimate front.

(j)



(k) When examining the second export, InstallService, we see that this function is used to set a host of registry keys with the intention of using this new dropped dll whenever someone attempts to start the Remote Registry Service. They do this by hijacking the ServiceDLL value which determines which dll to load when a service is started.

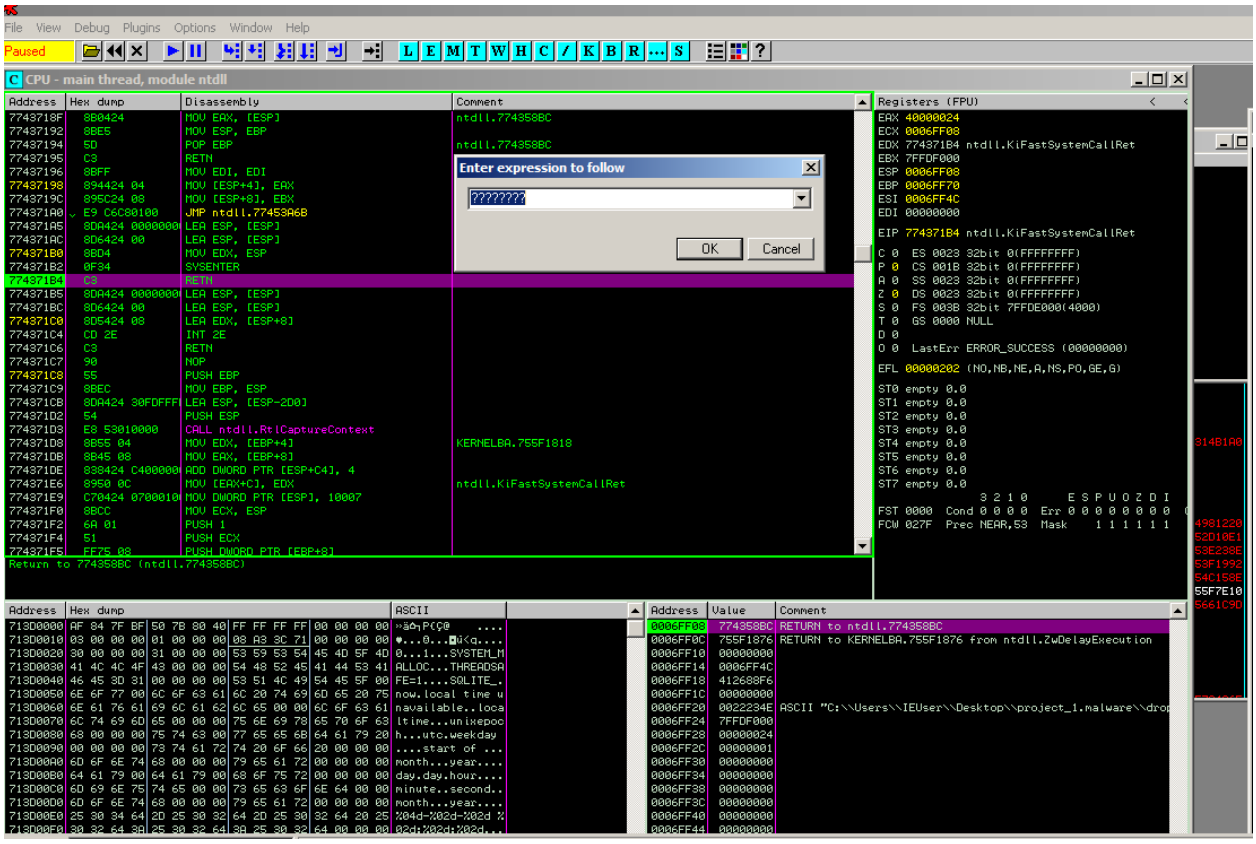
(l) Conclusion: Further static analysis proved difficult. There are a host of suspicious includes such as CryptUnProtectData which likely are used in conjunction with the google chrome password logging which don't seem to be able to be cross referenced through the rest of the code. Perhaps as an anti-analysis technique they are only calling these functions indirectly at runtime. Further advanced dynamic analysis should determine if this is the case and reveal exactly how these functions are being used.

Advanced Dynamic Analysis

22. Purpose: We've now covered all the main functionality of the program, we want to see how it's used. We're going to place breakpoints in each of the four major functionalities provided by the dropped dll and examine the variables they're called with. This should provide us with a complete picture of how the malware is used and perhaps allow us to glean some information as to what the attacker wants.

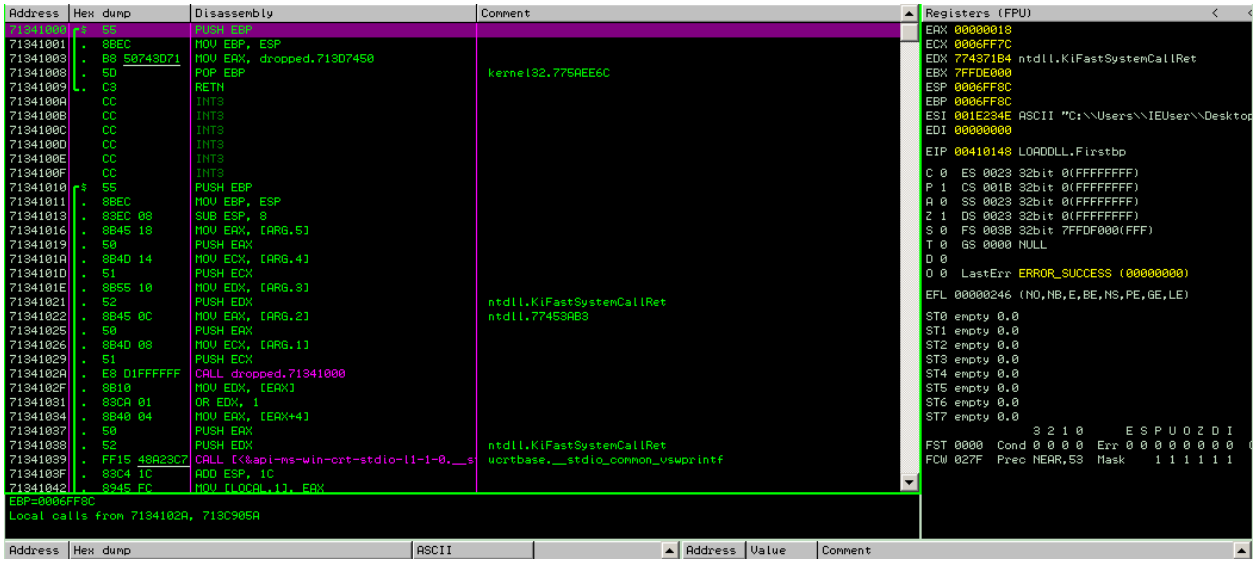
23. IDA Debugging: I chose to use the OllyDBG to debug the malware.

(a)

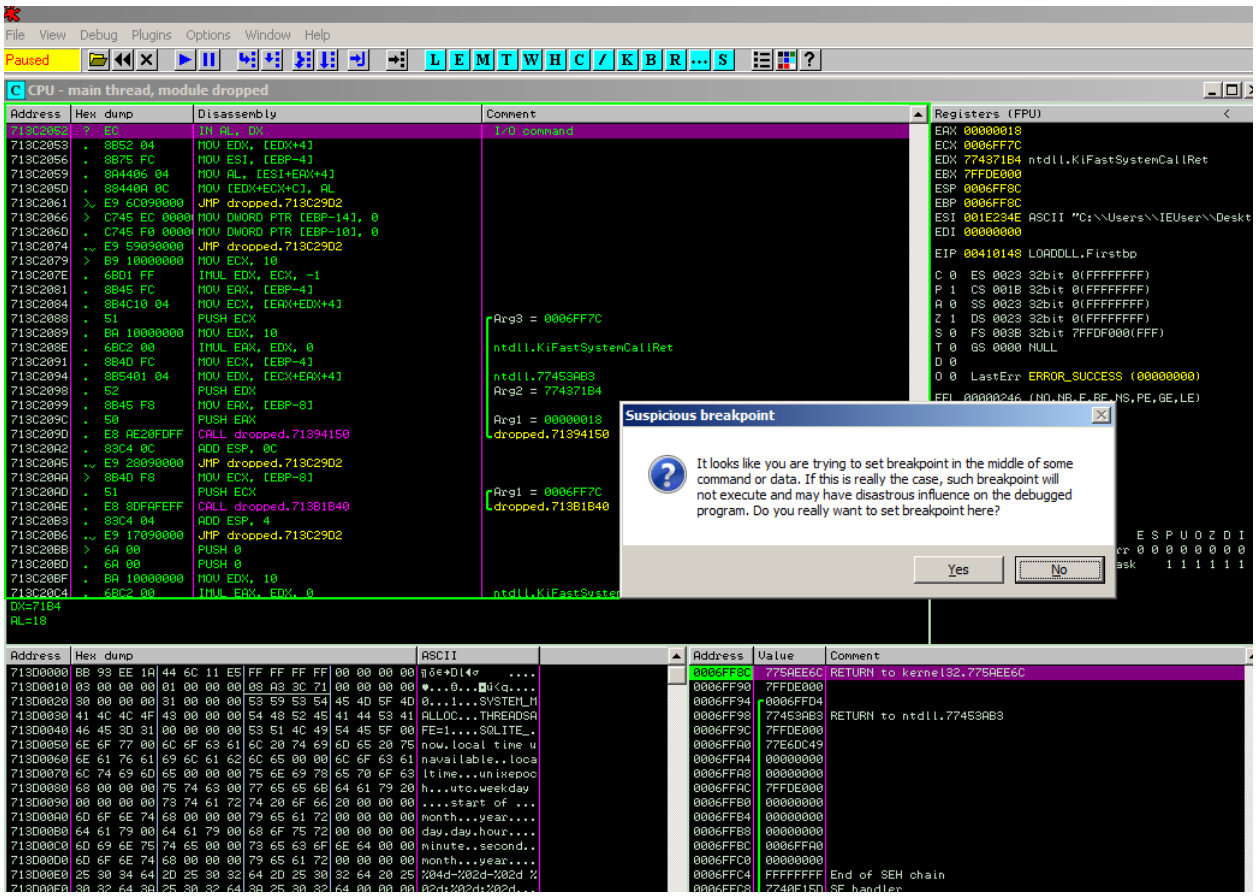


(b) When I ran the debugger initially, I was greeted with ASLR issues and had to search through olly with hex to find my given breakpoints using the addresses I retrieved from IDA.

(c)



(d) When I finally found my breakpoints, I was able to use olly to break when the program began to decrypt the Google Chrome passwords. It looks similar to the previous malware we examined which confirms that it is decrypting these passwords.

(e) 

(f) Unfortunately, it appears my struggles with olly were not yet over. Upon attempting to place the rest of the breakpoints, I repeatedly ran into this issue. It appears that I'm attempting to place my breakpoint inside of an instruction.

(g) Further investigation revealed that the unpacking progress caused the executable to segfault. The resulting dll is able to run however. Examining only the dll, I was able to confirm the conclusions I came to in advanced static debugging: Namely that the dll uses the GET instruction to check for new actions from the attacker and that it is able to utilize all the above functionality as a result of that GET request.

(h) Conclusion: All of the threats stated above are valid. The file hijacks the dynamic requests service and uses it to provide an attacker with delete / download / run command abilities on the host in addition to passing along any valid google chrome saved passwords.

Summary

(a) This is a sophisticated piece of malware. It starts with an executable which immediately launches the remote registry service and drops a DLL named VXtExt.dll. This dll has two major exports, ServiceInstall and ServiceMain. ServiceInstall employs a load order hijacking and a registry key change to SessionDLL in order to call this newly dropped dll instead of the system standard. ServiceMain has a command center which sends

off decrypted google chrome passwords to malcode.rpis.ec via a POST command, and then waits for further instructions via a GET query. The options for the GET query involve DeleteFile which Deletes a given file, Download file, which allows the attacker to download further resources to the hosts computer, Commandline which allows the attacker access to a terminal on the hosts machine. Finally there is SendDataToAttacker which will post any arbitrary file back to the attacker. All in all this is a complex malware with a unique persistance method and ongoing remote control over the host machine.