Winmgr Malware Analysis

**COMP 4970: Software Re-Engineering**

**Summer 2018 - Team Project**

**Authors: Achille Heraud, Trisha Sengupta, Jordan Sosnowski, and John David Watts**

**Table of Contents**

[**Black Box Phase Notes**](#_5ph4qj7a1on) **4**

[Initial Execution](#_44qsv5ftj4jd) 4

[Process Monitor](#_cp61h5i4spaa) 4

[File Explorer: Locating the malware](#_3nglz9tsefnt) 5

[**Static Analysis Phase Notes**](#_lnzgnrf3w8rr) **7**

[UnPacking the Malware](#_csa36wc1uopw) 7

[Odd Imports and Strings](#_d7frcpf3t8pl) 8

[Imports](#_yjwlrpxmr7me) 8

[Strings](#_ux5t72u33u9g) 8

[Delving into Assembly using IDA](#_gkmhmsp49rtt) 8

[Secure Environment Check (sub\_4014B0)](#_68oj879d9dx9) 9

[Hiding the malware](#_c8zt049wrmc6) 12

[Setting Attributes](#_go85c5cs0dic) 15

[Setting Registry Keys](#_pcd9xhna8a86) 16

[1st Key](#_jh70z0hvfw6v) 16

[2nd Key](#_9pu5wxxbeiwf) 16

[3rd Key](#_wdbyyszce6wm) 16

[4th Key](#_ygihb3h5ili7) 17

[Virus Cleanup (sub\_40340)](#_i69zd3jlp3fy) 17

[Concurrent Threads](#_jom32mz5t0kz) 18

[**Report**](#_z88e7al92ekx) **20**

[What is the general functionality of the sample?](#_qmkzdi1dfj7h) 20

[What are the indicators that this sample is malicious?](#_6gucl0s3olg8) 20

[How does this sample interact with the local system (e.g., system DLLs, files, etc.)?](#_yujtosal0w7n) 20

[What files and registry keys does this sample create, modify and access?](#_pzfhztaizih5) 21

[Update.exe creates/modifies/accesses the following registry keys:](#_psn6jtsg445j) 21

[Winmgr.exe creates/modifies/accesses the following registry keys:](#_i4a7mxpdiwo0) 21

[Update.exe creates/modifies/accesses the following files:](#_xqka6nlwzlws) 21

[Cmd.exe creates/modifies/accesses the following files:](#_veq4xvbh9eso) 22

[Winmgr.exe creates/modifies/accesses the following files:](#_b9nhhe2g3f99) 22

[What is the network behavior (including hosts, domains and IP addresses accessed)?](#_tnfjdrw8vlnu) 23

[Introduction](#_9nfjq5o8iicb) 23

[Nickname](#_6u9cvmq0rjlr) 24

[Authentication](#_42ioj56b2578) 26

[IRC Commands](#_f6uxmm41vmm7) 28

[What are the time and local system dependent features?](#_rth4r9lytquc) 33

[What is method and means by which this sample communicates to the external environment?](#_6malovhazq1b) 33

[What is the original infection vector and propagation methodology?](#_3n70ta5qak8d) 34

[What use does this sample make of encryption for storage, communication?](#_mnudoanx1wg1) 34

[What self-modifying or encrypted code does this sample employ?](#_2e8ij99s1t3) 35

[What ancillary information is available concerning the development of this sample compiler type, country of origin, author names/handles, etc.)](#_a4ct83wrox5p) 35

# Black Box Phase Notes:

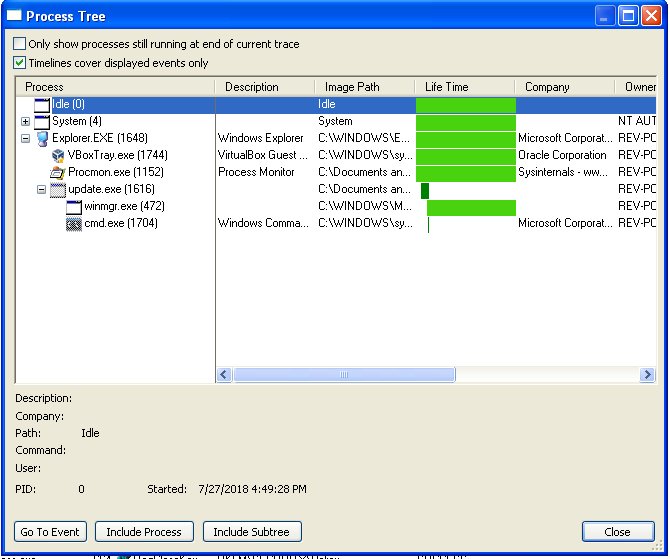
## Initial Execution

* + 1. After executing the program nothing visually happens on the screen other than update.exe disappearing. To a normal user this may seem typical, so they may not suspect something malicious has happened. Shortly after that, we are given a warning that tells us system files have been changed. This should throw some red flags to the user that something has gone wrong.



## Process Monitor

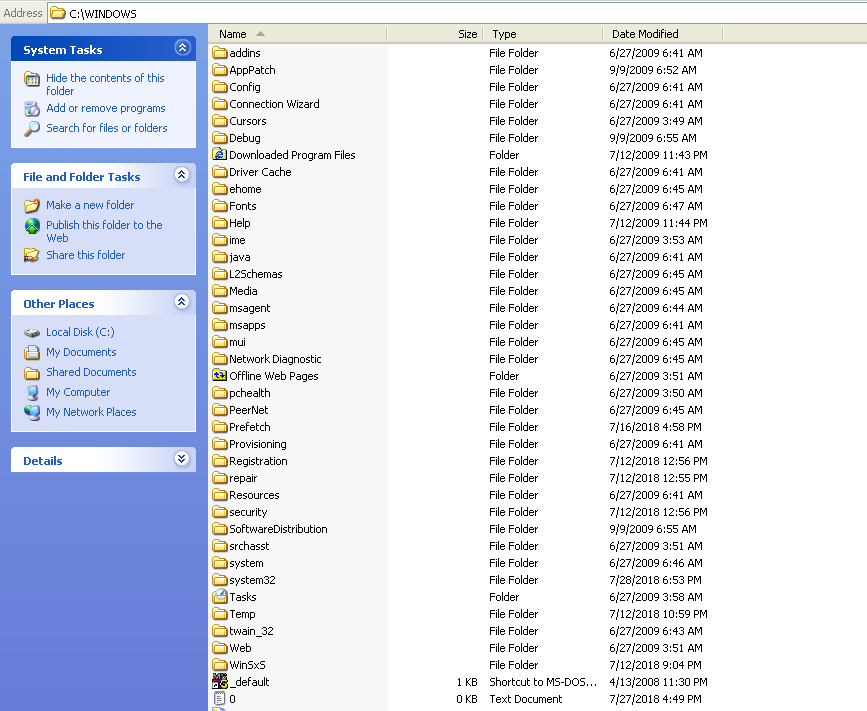
* + 1. If one were to run process monitor during this execution and activate the process tree functionality and then look under Explorer.EXE, one would find update.exe. One would also find that update.exe starts two additional processes “winmgr.exe” and “cmd.exe.” It also tells us that “winmgr.exe” has a path of C:\\WINDOWS\\M-50504508848879876012050406030\winmgr.exe.

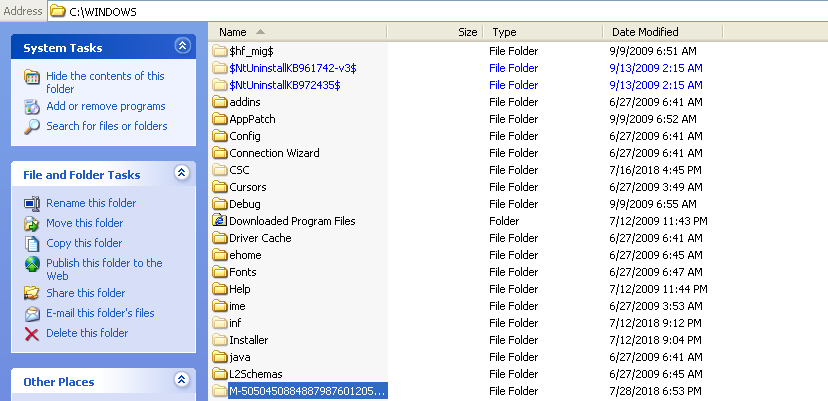


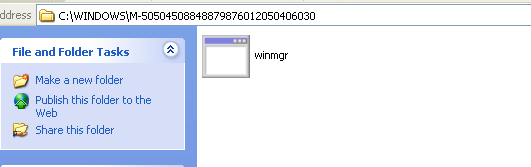
* + 1. Filtering out all processes that are not winmgr.exe. We can see that winmgr.exe does quite a few things. Specifically, with the registry. Which are discussed more in detail later in this report.

## File Explorer: Locating the malware

* + 1. If we try to go to the path listed in process monitor, using File Explorer, it initially looks like it does not exist. This is most likely because the malware has tried to hide the folder “M-505…”. After selecting the option to show hidden folders and system folders in the file options folder “M-505…” appears and inside it is “winmgr.exe”.



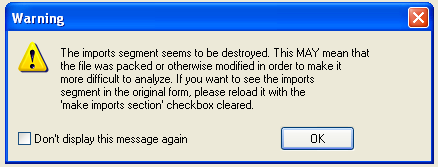




# Static Analysis Phase Notes:

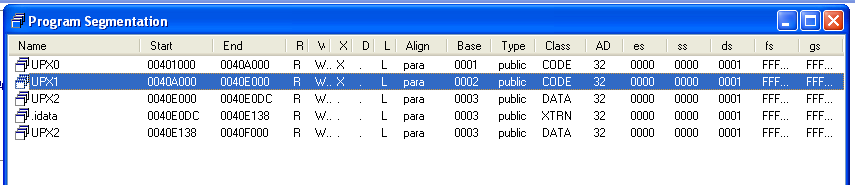
## UnPacking the Malware

* + 1. After booting up “update.exe” or “winmgr.exe” with IDA, the first warning flag thrown shows a warning telling the user that the import segment seems to be destroyed.



* + 1. Clicking okay and inspecting the code, one can see that the data portion of the file is extremely small and also that the only segments listed in the segment view are UPX0, UPX1, UPX2, idata, and another UPX2.





* + 1. After doing a quick google search, one can find out that UPX is an executable packer. So most likely the author of the malware packed the said malware to one, shrink the size of the code, and to also make it harder to reverse.
    2. After unpacking with the command UPX -d update.exe / winmgr.exe we can now correctly analyze the malware.

## Odd Imports and Strings

### Imports:

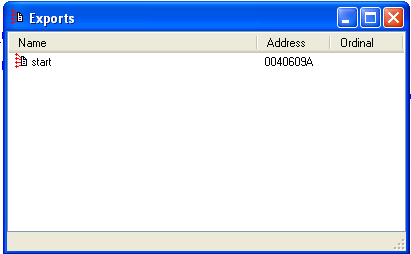
* + - 1. GetUserNameA, RegOpenKeyExA, RegSetValueExA, DeleteFileA, IsDebuggerPresent, TerminateProcess, ShellExecuteA, InternetOpenA, Socket, and Connect to name a few.

### Strings:

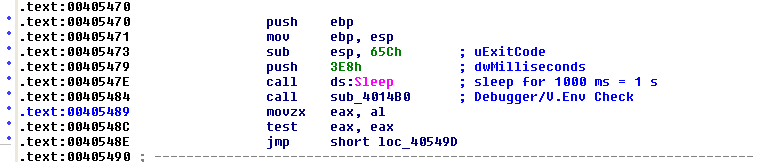
* + - 1. NICK, JOIN, PING, PONG, USER, PRIVMSG
         1. These are used primarily for IRC servers. Which used to be popular but are now used for hackers to control botnets.
      2. MSSECES.EXE, MSASCUI.EXE, MRT.EXE, RSTRUI.EXE, WUAUCLT.EXE
         1. Which are all Microsoft windows applications, primarily dealing with Virus Removal/Security
      3. qemu, wine\_get\_unix\_file\_name, \\\\.\\PhysicalDrive0
         1. Which all have to do with Secure/Virtual/Sandboxed Environments
      4. C:\\Users\\s\\Desktop\\Home\\Code\\Trik v2.5\\Release\\Trik.pdb
         1. Which is a known spam bot virus

## Delving into Assembly using IDA

* 1. After unpacking winmgr and booting up IDA we see that the program starts at .text:004060A9



* 1. Not much happens initially at the start point. The call to \_WinMain@16 at line .text:004061C9 is where the assembly gets interesting.

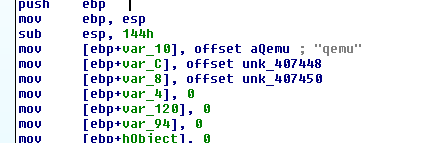


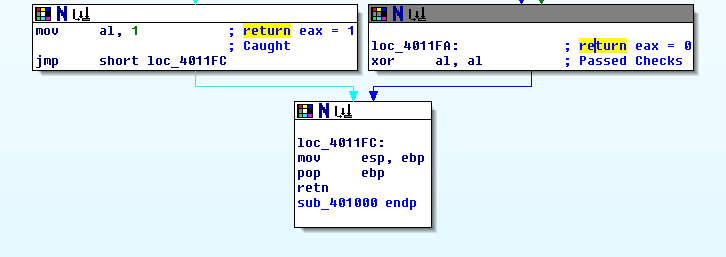
* 1. We first see that the malware will sleep for 1 second (parameter 3E8h in decimal is 1,000ms is a second). This is most likely to give a pause before executing the rest of the malware just in case some values are not set yet.
  2. After running the Sleep function, we have a call to sub\_4014B0. This call contains checks to make sure the malware is not running in a secure environment.

### Secure Environment Check (sub\_4014B0)

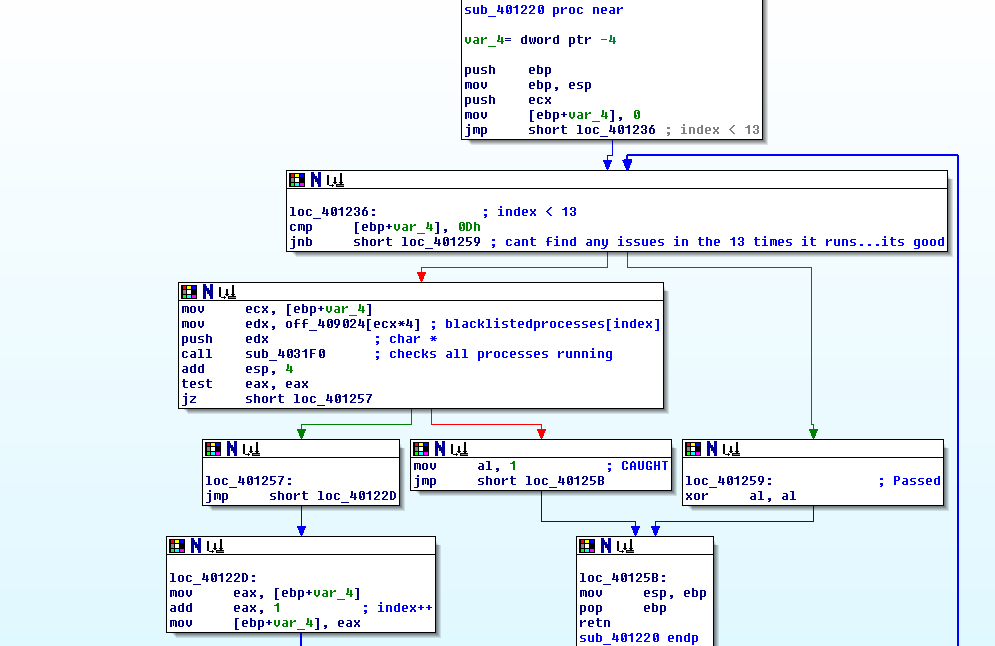
*What follows is a high-level overview of these functions. A more detailed explanation will be at the end of the report.*

* + 1. The first check the is performed is sub\_401000. Which contains code that queries the PhysicalDrive and checks to see if its name matches any values that are in a list of blacklisted hard drives. One of these hard drives is “qemu” which was discussed early with the string imports. Qemu is an open source hardware virtualizer. There are two other strings that are checked but they are unknown at the point in time. A good assumption to make here is that they also contain common hardware virtualizers. If the hard drive of the user matches these then the program will immediately return 1. If it passes, then it will return 0.

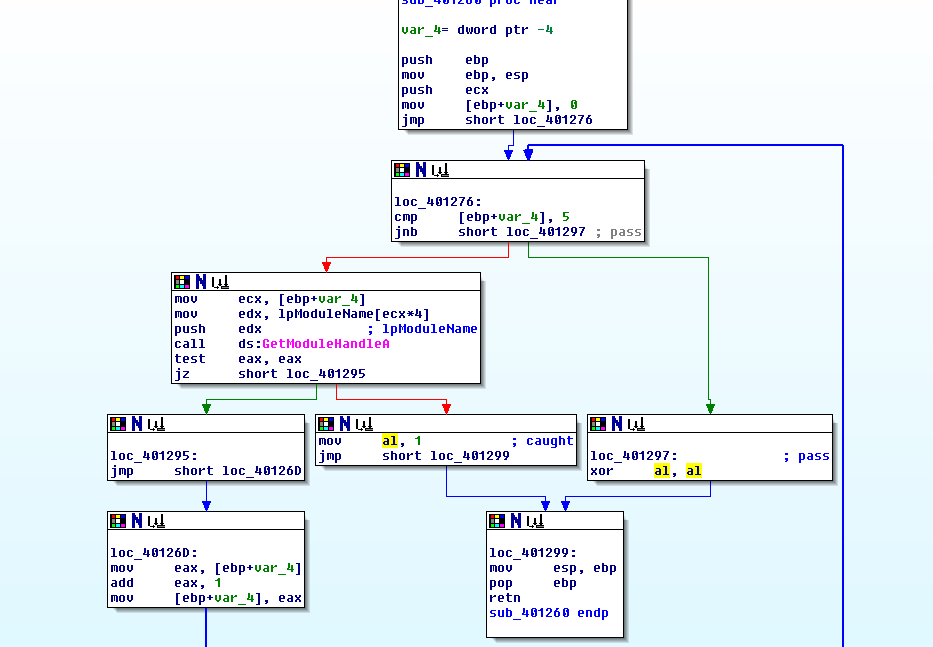




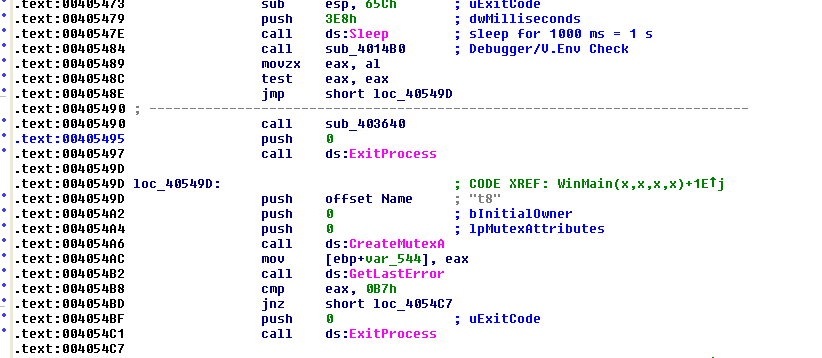
* + 1. Following the check of partitions, we will call sub\_401220. Which will enter a loop that will iterate through an array of blacklisted processes and see if they are currently running on the system with the help of function CreateToolhelp32Snapshot found in sub\_4031F0. If a blacklisted process is found running, this function will return 1 and if not, it will return 0.



* + - 1. Processes that are contained in the blacklist are probably MSSECES.EXE and the other applications listed with it found earlier in the report. My logic behind this is that MSSECES is a windows security program that will check for viruses. So, if it is running the malware it will want to exit itself before being found by virus protection programs.
    1. Following this malware will make a call to sub\_401260 which checks the current dlls running against a blacklist of dlls. Similar to the above functions if they are found, the program will return 1 and if not, it will return 0. We also see that it runs 5 times, therefore there are probably 5 dlls it checks against. However, at this point of the static analysis the names are unknown.

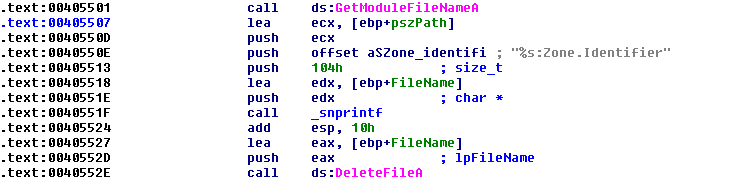


* + 1. Following the check of dlls we will check file names (sub\_4012E0) created by .dlls or .exe. If these are in existence just like before, the program will return 1 and if not, it will return 0. List unknown at this point.
    2. Following this, the program will check the current user’s username (sub\_4013F0). If the username hits one in the blacklist, it will return 1 and if not, it will return 0. List unknown at this point.
    3. After the check of usernames, the malware will check to see if wine is present in the system(sub\_401470). It will return 1 if it is, and it will return 0 if it is not.
    4. Following the wine check, the malware will check to see if a debugger is present (IsDebuggerPresent) and return 1 if it is, and 0 if it is not.
    5. If any of these had returned 1, the malware would immediately return 1 from sub\_4014B0. If not, it would return 0. The interesting thing is that even if the malware returns 1 or 0 nothing happens. Following the checks of a secure environment at .text:0040548C, there is a test to see if the value returned was 1 or 0 but after that, there is a normal jump and not a conditional. So even if 1 was returned the code still executes.
       1. Assumption: This was done intentionally by the instructor or whoever had the code prior to make reversing easier.
  1. At this point any reverse engineer should be convinced that this file might be malicious. Programs should not need defense tactics against anti-virus or virtual environments unless that have something they want to keep hidden
  2. Following the secure environment checks the malware will create a mutex at .text:004054A6 with the name “t8.” Then it will perform a check to see if that creation returned an error 0B7h which after searching it on google means “ERROR\_ALREADY\_EXISTS.” If this error appeared, that means the mutex is already locked by a prior iteration of the malware and at this point the malware will exit because it does not want to infect the host multiple times. Following this, we will start the process of trying to hide the malware from the user.

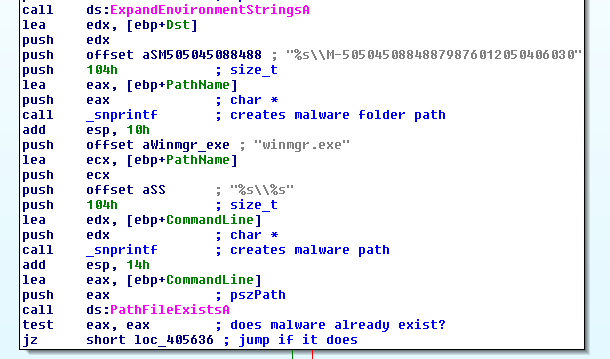


### Hiding the malware

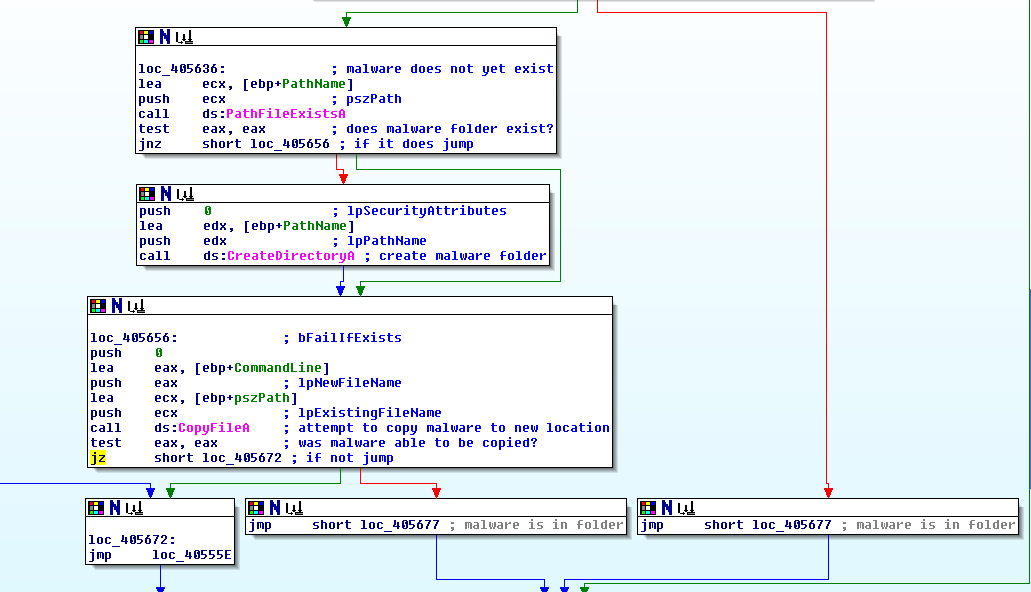
* + 1. First, we perform a deletion of a zone identifier. From google we can find that zone identifier files are known as “alternative data stream files” which are used to describe the data of other files. They have the same filename as the file they are downloaded, plus a suffix “:Zone.Identifier.” (i.e. you download “test.exe” the zone identifier that downloads with it is “text.exe:Zone.Identifier”)
    2. Based on this, we can assume that the malware is deleting the zone identifier that was downloaded with itself to try to obscure itself from detection.

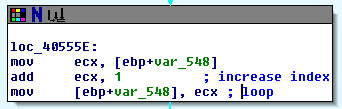


* + 1. Following this we create a few strings: “%windir%”, “%userprofile%”, “%temp%.” These are all environment strings which will be used later. (.text:00405534)
    2. After this, we enter a for loop that iterates 3 times (for each environment string). Inside the for loop the malware will expand the current environment string with a call to ExpandEnvironmentStringA (.text:004055D6).
       1. %windir% will become C:\WINDOWS
       2. %userprofile% becomes C:\Document and Settings\<username>
       3. %temp% becomes C:\Document and Settings\<username>\Local Settings\Temp
    3. Following this, the malware will concat \M-50504508848879876012050406030 to the current expanded environment path. We will refer to this as the malware folder. After this, we will create the malware path by concating \winmgr.exe to the malware folder (\_snprintf) this will be referred to as the malware location. Then we will check to see if the malware is already there (PathFileExistA). If it is, we will skip down to loc\_405677. This will be covered later.

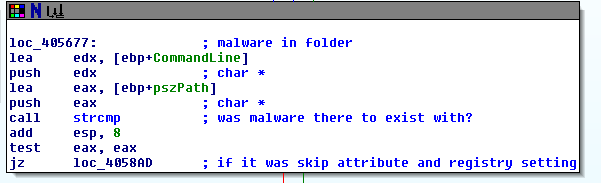


* + 1. If the malware does not currently exist, then the malware will check to see if the folder exists (PathFileExistA). If it does not, it will create it (CreateDirectoryA). Following this it will attempt to copy the file with CopyFileA. The Copy FileA will return a 0 if it the copy failed.
       1. If the copy fails, the malware will go to the next iteration of the for loop and try a different environment variable path.
       2. If the copy succeeds it will go to loc\_405677.



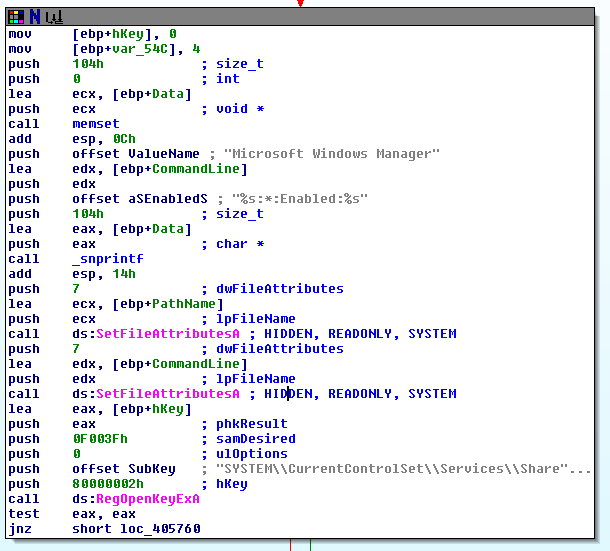


* + 1. At loc\_405677 the malware will check to see if the original malware location is equal to the new location.
       1. If it is, that means the malware is being ran for a second time and it does not need to mess with the file attributes or registry keys and it will jump to loc\_4058AD.
       2. If the original file location does not equal the new location that means it was recently copied over and this is the first time the malware is running. So, it goes to .text:00405695.

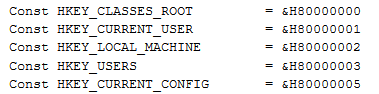


### Setting Attributes

* + 1. Here, the file attributes for the folder C:\<whichever environment path worked>\M-50504508848879876012050406030. It will be set as Hidden, Read Only, and System by a call to SetFileAttributesA. Following this the malware will set winmgr.exe to Hidden, Read only, and System in a similar way.
       1. This matches up with the analysis that was made during the black box phase.

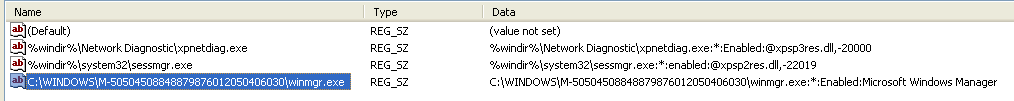


### Setting Registry Keys

* + 1. The malware will attempt to open four registry keys here and if it can open them, it will set values by using RegOpenKeyExA and RegSetValueExA.
       1. 

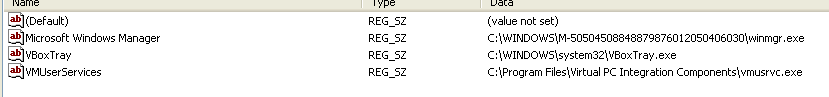
#### 1st Key

* + - 1. Hkey: 80000002h (local machine)
      2. Subkey: “SYSTEM\CurrentcontrolSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\AuthorizedApplication\\List\”
      3. Value Set: malware path
      4. Data Set: malware location + ":\*:Enabled:Microsoft Windows Manager"
         1. Allows malware to bypass the firewall



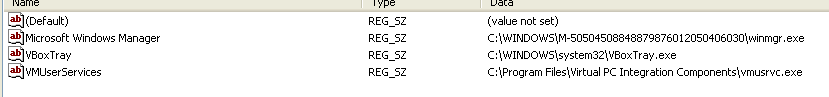
#### 2nd Key

* + - 1. Hkey: 80000002h (local machine)
      2. Subkey: “SOFTWARE\Microsoft\Windows\CurrentVersion\Run\”
      3. Value Set: “Microsoft Windows Manager”
      4. Data Set: malware path
         1. This will auto run the malware at computer boot up.



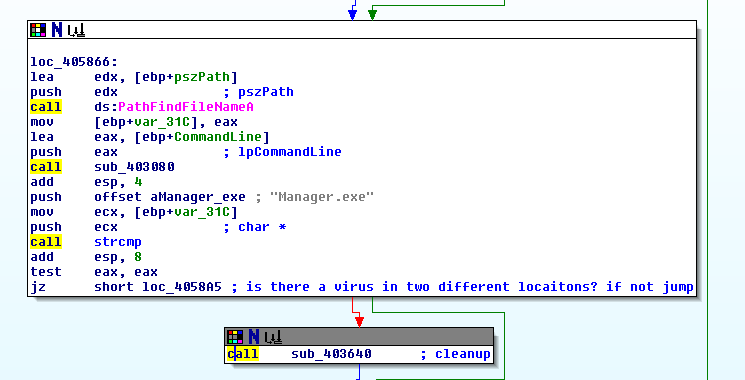
#### 3rd Key

* + - 1. Hkey: 80000001h (current user)
      2. Subkey: “SOFTWARE\Microsoft\Windows\CurrentVersion\Run\”
      3. Value Set: “Microsoft Windows Manager”
      4. Data Set: malware path
         1. This will auto run the malware at computer boot up.
         2. This is similar to the above registry but the effects specifically the current user. While the above one sets it for any user



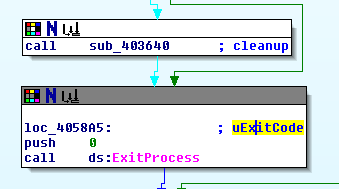
#### 4th Key

* + - 1. Hkey: 80000002h (local machine)
      2. Subkey: “SYSTEM\CurrentControlSet\Services\WinDefend\”
      3. Value Set: “Start”
      4. Data Set: 4
         1. This will disable Windows Defender
         2. Since our version of XP does not have Windows Defender the malware actually won’t be able to open this key
  1. Following the registry and attribute setting we perform some clean up. We check to see if the path of the virus is equal to where it where it originated. Similar to above. If it is, we will skip the call to sub\_403640. If it is not that means we need to delete the original virus.



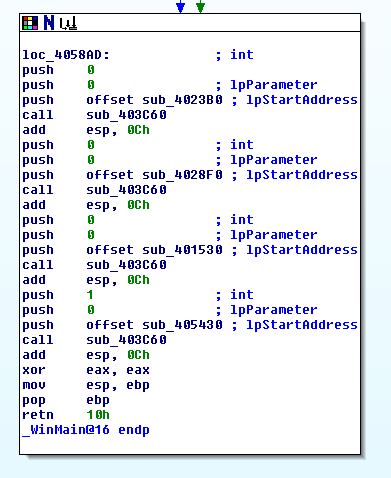
### Virus Cleanup (sub\_40340)

* + 1. Here a batch file will be created. The contents of this file will run a script that will delete the original virus file and also delete the batch file. This is extremely similar to ?dontuseme in hacarmy. However, it will also start a new instance of the virus. The original virus at this point will end since after sub\_403640 is executed ExitProcess is called.
    2. Patching out the call to sub\_40340 will prevent the original malware from being deleted. This was tested and confirmed by filling that line with NOP in immunity.



* 1. Picking up with the new instance of the virus, at this point we would skip down to loc\_4058AD as discussed earlier. Here we will spawn 4 threads to execute which is where the main part of the virus will execute.

### Concurrent Threads



* + 1. At sub\_4023B0 there are calls to GetLogicalDriveStringsA to find the removable drive and then checks the drive type. It also gets the volume information of the removable drive. If the drive is of type ‘a’ or ‘b’ it will leave. Depending on the type of drive, it will create the following files:
       1. Autorun.inf
       2. DeviceManager.bat
       3. Manager.bat (network drive)
       4. Manager.js (removable)
       5. .lnk (shortcut for Manager file)

Autorun.inf will act as an autostart that will execute the malware when the drive is opened. It will activate Manager.bat/js which activates DeviceManager.bat.

* + 1. Beginning at sub\_402630() the malware will attempt to copy of itself into specific folders in fixed drives. It targets these folders:
       1. public\_html (.rdata:0040780C)
       2. htdocs (.rdata:0040781C)
       3. httpdocs (.rdata:00407824)
       4. wwwroot (.rdata:00407830)
       5. ftproot (.rdata:0040783C)
       6. share (.rdata:00407848)
       7. income (.rdata:00407850)
       8. upload (.rdata:00407858)
       9. warez (.rdata:00407850)

If found, for every .exe file in that folder, it will replace it with a

copy of itself. Similarly, for every .zip or .rar in that folder it will add a copy of itself as README.txt.scr.

* + 1. At line .text:004058D8 it will create another thread to execute at sub\_403C60.
       1. This function will loop through a list of blacklisted processes to check if they are running. This is similar to when we were checking for a secure environment, using the same processes too. This is most likely done as a last scan to make sure it will not be discovered before starting the 4th thread which initiates the IRC connection.
    2. At sub\_405430() the IRC server connection is setup. This is described later in the report when discussing network behavior.

# 

# Report

## What is the general functionality of the sample?

The general functionality of this sample is a to establish a backdoor in the host machine to allow itself to download updates of itself, any files that the attacker wishes to put on the host and functionality that allows the attacker to use the host machine as a DDOS tool. Since the malware can spread itself through the network drives, removable drives, and fixed drives, it could also be classified as a worm. Since it also tries to keep itself from being removed, it can also be classified as a sticky malware.

## What are the indicators that this sample is malicious?

The malware being packed with UPX is the first indicator that this may be malicious.

After the signs of maliciousness are the odd imports that aren’t usually used in most programs. Such as IsDebuggerPresent, ShellExecuteA and the other functions listed above.

There are also odd strings that were discussed above such as NICK, PONG, PING, JOIN, the odd URLs which seem to point to IRC servers which are now a days used for botnets and the URLs that after doing quick google searches, are malicious.

The IRC commands that are discussed later on in the report are extremely malicious and would not be used in normal programs.

Other indicators that this is malicious are checks against if the program is running in a secure environment, the hiding of itself, and setting of registry keys.

The most glaring indicator is the string that references the trik spam bot.

## How does this sample interact with the local system (e.g., system DLLs, files, etc.)?

Update.exe executes and changes different settings on the infected computer. It accesses and manipulates internet settings and then creates a file called winmgr.exe. It manipulates the firewall and authorized apps so that winmgr.exe is not detected on your computer. This way it can run peacefully in the background while also running each time the computer starts up. The sample accesses user and administrator files and then reads through all of the documents along with the settings for each. It also accesses configuration information of each Windows users. It accesses and reads through different dlls such as kernel32.dll, cmd.exe, user32.dll, and ole32.dll.

## What files and registry keys does this sample create, modify and access?

### Update.exe creates/modifies/accesses the following registry keys:

1. A key is created in Internet Settings with the desired access to read and write. It then opens a key in SharedAccess\Parameters\FirewallPolicy\StandardProfile\AuthorizedApplications\List and sets the desired access to all access.
   1. This is so it can set the register value in list and enable winmgr.exe (Microsoft Windows Manager), a file that is a component of the sample. This way winmgr.exe is being set as an authorized application to run on the computer.
2. A key is opened in Microsoft\Windows\CurrentVersion\Run with the desired access to read and write. It then sets the registry value so that winmgr.exe runs every time the computer starts.
3. A key is created in Explorer\User Shell Folders and Explorer\Shell Folders with the desired access to maximum allowed. This means the sample has maximum access to all the default folders (like “My Pictures) and the contents inside them as well as the configuration settings of applications on the computer.
4. A key is opened in HKCU (more than once) with the desired access to maximum allowed. This means the sample has access to configuration information for Windows and software specific to the currently logged in user.
5. A key is created in WinTrust\TrustProviders\SoftwarePublishing with desired access as read.

### Winmgr.exe creates/modifies/accesses the following registry keys:

1. A key is created in Internet Settings with the desired access to read and write.
2. A key is created in HKLM\System\CurrentControlSet\Services\Tcpip\Parameters with Desired access as read
3. A key is opened in Services\WinSock2\Parameters with the desired access to maximum allowed.

### Update.exe creates/modifies/accesses the following files:

1. A file is created in system32\ws2\_32.dll with desired access to execute and traverse. Ws2\_32.dll is a file that contains the windows sockets API used by most internet and network applications to handle network connections.
2. A file is created in system32\shell32.dll with desired access set to generic read and execute. Shell32.dll is a dynamic link library that controls certain API functions of Windows Shell (serves as the graphical user interface for Windows operating systems)
3. A file is created in WindowsShell\Manifest with desired access set to execute/traverse and synchronize
4. A file is created in system32\comctl32.dll with desired access set to generic read/execute. Comctl32 is known as common controls and provides many GUI functions to the Windows system.
5. A file is created in system32\URLmon.dll with desired access set to generic read/execute. URLmon.dll is used when performing OLE (Object Linking and Embedding) operations, which allows objects created in one application to be embedded in documents and other objects created by a different application.
6. The sample creates the file winmgr.exe in WINDOWS with the desired access set to generic write, read attributes, and delete. It uses operation WriteFile and SetBasicInformationFile.
7. A file is created in C:\Documents and Settings with desired access to read data/list directory.
8. The sample opens a query in Administrator\My Documents\desktop.ini and All Users\Documents\desktop.ini with desired access set to generic read. A Desktop.ini file is a file that determines the way a folder is displayed by Windows.
9. The sample creates a file, opens a query, and then reads the file in in system32\rsaenh with desired access set to generic read and then creates another file with desired access set to execute/traverse. Rsaenh.dll is a module that implements the Microsoft enhanced cryptographic service provider (CSP) that uses 128-bit encryption.
10. A query is opened in system32\cmd.exe with desired access set to read data/list directory, execute\traverse, read attributes, and synchronize.

### Cmd.exe creates/modifies/accesses the following files:

1. A file is created in C:\ with desired access set to read data/list directory and synchronize.
2. Several files are created in system32 with desired access set to read data/list directory, read attributes for different dlls like user32.dll, ole32.dll, windows shell manifest, comctl32.dll
3. Several files are created in system32 with desired access set to execute/traverse for different dlls like kernel32.dll, cmd.exe, user32.dll, ole32.dll
4. Reads documents and settings for all users and administrator

### Winmgr.exe creates/modifies/accesses the following files:

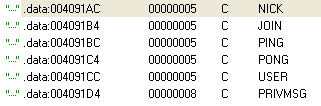
1. Goes into administrator documents and settings where application data is stored with desired access to read data/list directory and also execute/traverse and synchronize
2. Goes into administrator desktop, Favorites, Local Settings\Application Data and reads data and executes/traverse
3. Reads documents and settings for all users and administrator
4. Access files, reads data, and executes and traverses through XPMUser settings, Program Files, and WINDOWS files
5. A file is created in system32\dllcache\share.exe with desired access set to generic write, read attributes, and delete.
6. A file is created in system32\dllcache\UploadM.exe with desired access set to generic write, read attributes, and delete.
7. A file is created in WINDOWS\pchealth\UploadLB\Binaries\UploadM.exe with desired access set to generic write, read attributes, and delete.
8. A file is created in system32\share.exe with desired access set to generic write, read attributes, and delete.

## What is the network behavior (including hosts, domains and IP addresses accessed)?

### Introduction

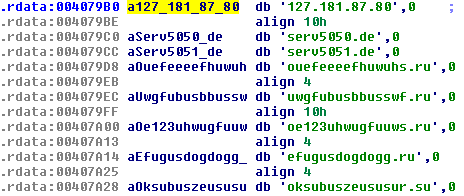
When run without an internet connection, it performs NBNS name queries for SERV5050.DE, SERV5051.DE, and WORKGROUP. This was discovered by running wireshark and capturing packets that were being sent while the malware was running. The NBNS query for WORKGROUP occurs even before the malware is run, so I will ignore it. NBNS name queries serve the same purpose as dns, it converts a hostname or a machine name into an ip address.

When inspecting the strings contained in the unpacked malware, we noticed that the data segment beginning at .data:004091AC contained the strings "NICK", "JOIN", "PING", "PONG", "USER", "PRIVMSG".

****

This suggested to me that the malware was attempting to connect to an irc server at either "SERV5050.DE" or "SERV5051.DE" on port 5050. We edited the hosts file to direct "SERV5050.DE" and "SERV5051.DE" to 127.0.0.1 and ran a local irc server using miniircd. We intended to use wireshark to capture traffic and analyze whatever the malware was sending in case it wasn't irc, but on Windows XP Wireshark isn't able to capture packets sent to the loopback address. Fortunately, the malware does connect to an irc server, so we didn't need to use wireshark to see what it was doing. When the malware connects to the irc server, it joins channel #t and waits for commands.

After looking at the assembly for how the malware was attempting to join the server, we noticed that it actually was trying to connect to 3 hosts, it first tries to connect to an irc server at 127.181.87.80, and if it isn't successful then it tries to connect to SERV5050.DE and finally tries to connect to SERV5051.DE. These connection attempts happen in an infinite loop, so if the malware doesn't have an internet connection when it starts it will keep cycling through the 3 servers attempting to connect until it is able to establish a connection. The function sub\_405300 is responsible for connecting to the irc server.



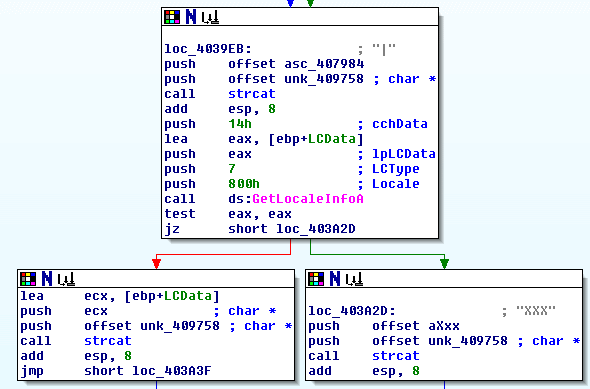
These connection attempts happen in an infinite loop, so if the malware doesn't have an internet connection when it starts it will keep cycling through the list of servers attempting to connect until it is able to establish a connection. The function sub\_405300 is responsible for connecting to the irc server.

### Nickname

The nickname that the malware joins the irc server with is generated by sub\_4037F0 and is composed of 5 parts, which are separated by the '|' character.

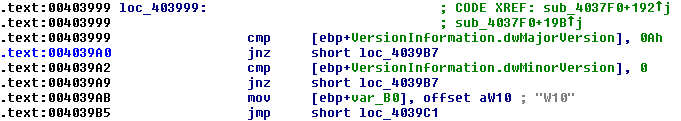
1. Country (example: "USA")

The country code is determined by calling the Windows API function GetLocaleInfoA.

****

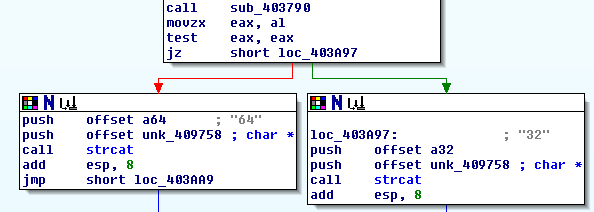
1. OS version ("95", "NT", "98", "ME", "2K", "XP", "2K3", "VS", "W7", "W8", "W10", "UNK")

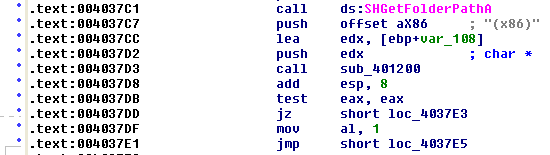
The version of Windows is determined by calling the Windows API function GetVersionExA, which, among other things, allows you to get the major and minor versions of the version of Windows that your application is running on. The fact that the malware includes Windows 10 as a possibility suggests that this malware is fairly recent, as Windows 10 was released in 2015.



1. 32- or 64-bit OS

The function that determines whether the malware is running on 32-bit Windows or 64-bit Windows is Sub\_403790

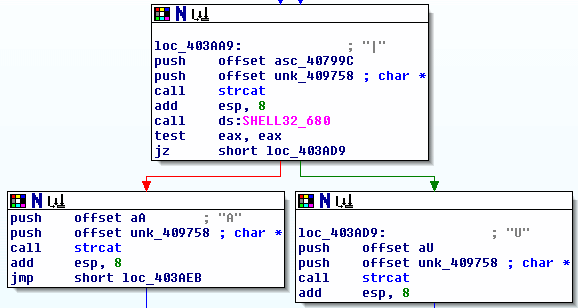
****



Inside of sub\_403790, SHGetFolderPathA is called, and then a second procedure is called with the results of SHGetFolderPathA and the string "(x86)". We assume this is checking for the existence of the folder "C:\Program Files (x86)", and if this folder exists then it is running in 64-bit Windows, and otherwise it is running in 32-bit Windows.

1. Is the user an admin? (“A” for admin, “U” for otherwise)

The function SHELL32\_680 is called to determine whether or not the user account the malware is running under is an admin.

****

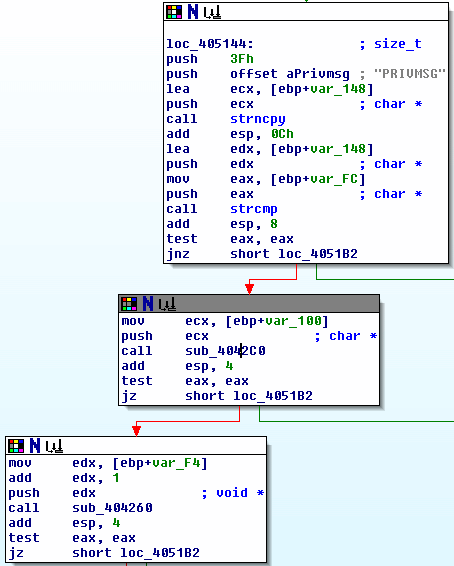
This function is exported from the SHELL32 dll by ordinal, not name, so "680" is not the name of the function, but it is the index in which it appears in the dll export table. According to https://www.geoffchappell.com/studies/windows/shell/shell32/API/index.htm SHELL32\_680 is actually named "IsUserAnAdmin".

1. Random string of characters (generated by sub\_402F10)

The last part of the nickname is a random string of characters. The purpose of these random characters is so there is no conflict between similar machines (when country/os/arch/admin all match).

### Authentication

To get the malware to execute a command you must send it the command either through one of the channels it has joined in the irc server, or through a direct message; however, the malware will not execute commands sent to it from any user in the irc channel. When you send a message to a channel or user on an irc server, you use the **PRIVMSG** command. The recipients of the message receive the text **:$nick!$real@$ip PRIVMSG$dest :$MESSAGE\r\n**.

****

Before executing any command, the malware inspects the ip field, and only proceeds to execute the command if the ip address field is equal to x. The explanation behind this is that some irc servers allow you to set what is called a 'cloak', which is displayed instead of the ip address. Since this functionality is restricted by the server administrator, we believe it is being used as a form of authentication. To get around this, we patched miniircd to display all ip addresses as "x", which was enough to get the malware to respond to my commands. The function in the malware that does this check is sub\_4042C0.

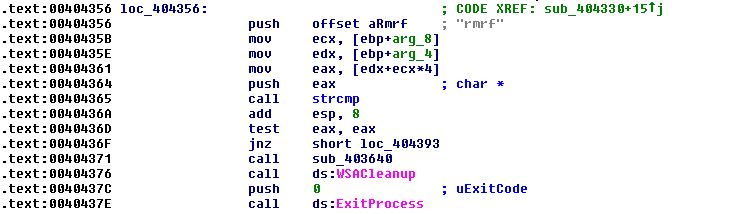
### IRC Commands

Commands can be sent in a channel where multiple instances of the malware are listening, in which case all instances that receive the message with execute it, or by a private message to a specific instance of the malware. Additionally, you can set a command as the topic of a channel, and when an instance of the malware connects to that channel it will execute the command.

The available commands are as follows:

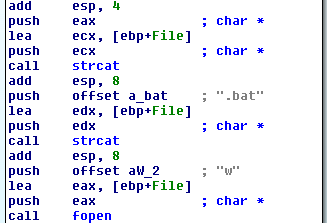
1. .rmrf

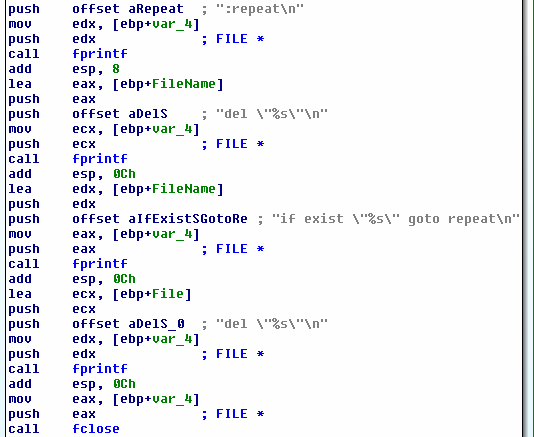
Removes the malware from the target computer and exit.



As we can see in the assembly, the rmrf command is implemented by sub\_403640, and after it is

executed the process is exited.





In sub\_403640, a batch file is created, and the text

:repeat

del "$FileName"

if exist "$FileName" goto repeat

del "$File"

is written to the file, where $FileName is the name of the malware executable, and $File is the randomly

generated name of the batch file (generated by calling sub\_402F10) that is being written to. Finally, the

batch script is executed by calling ShellExecuteA. The reason the malware uses a batch file to delete

itself is because Windows will not allow you to delete the executable for a program that is currently

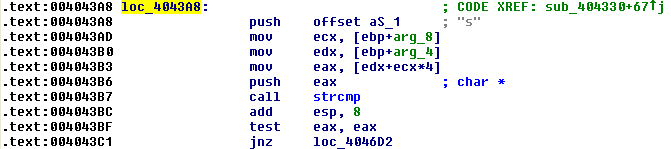
running, so the batch file will just wait in a loop until it is successfully able to delete the executable after

it is closed, and then the batch file removes itself.

Because all this command does is delete the malware executable, all of the registry keys and files modified by the malware will not be restored to their original state.

1. .s $MODE

Joins a specific irc channel based on system information, requires an argument to specify how to determine which channel to join.



1. o - operating system

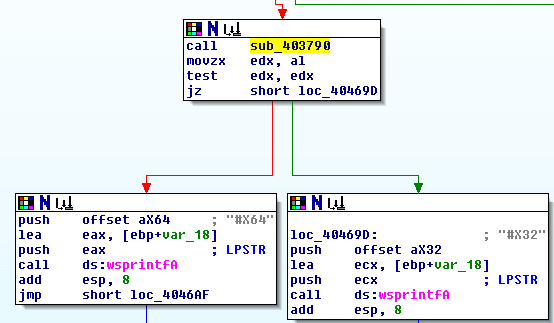
Joins the irc channel for the version of windows that the malware is running under. The possible values are "95", "NT", "98", "ME", "2K", "XP", "2K3", "VS", "W7", "W8", "W10", and "UNK". The version of windows is determined by calling the Windows API function GetVersionExA.

1. u - user

Joins the irc channel for the user account type the malware is being run under. "#ADMIN" for administrators and "#USER" for anything else. Whether the user is an admin or not is determined by calling SHELL32\_680 (aka IsUserAnAdmin).

1. a - architecture

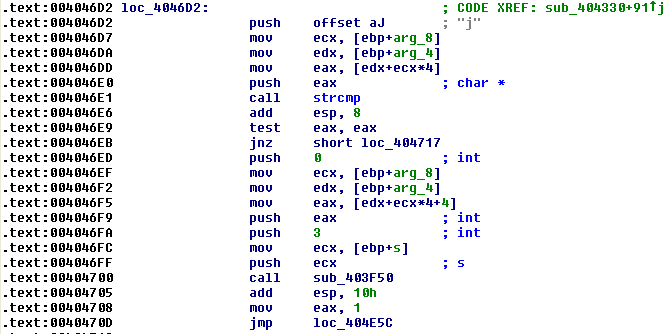
Joins the irc channel for the architecture that Windows was built for, "#X32" for 32-bit Windows and "#X64" for 64-bit Windows.



The method for detecting whether it is running on 32-bit or 64-bit windows is the same as was used to generate the architecture portion of the irc nickname.

1. .j $NAME

Joins the irc channel specified by the $NAME parameter. Since this is an irc channel, $NAME must begin with the '#' character.

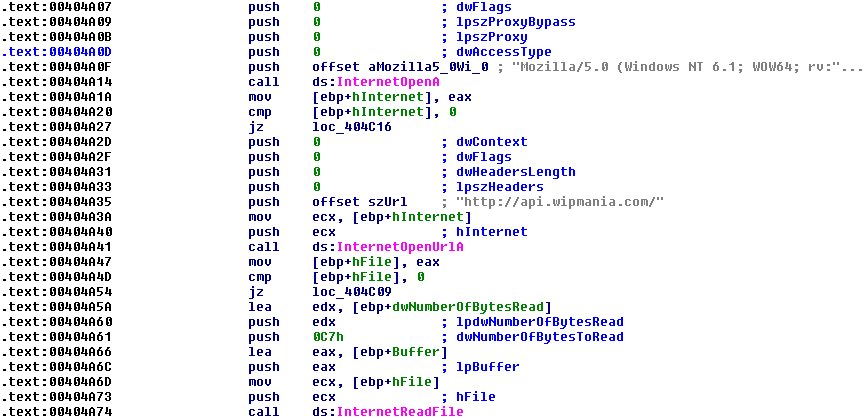


This command simply passes the $NAME parameter of the command to sub\_403F50, which is the function that sends messages to the irc server, the 2nd argument, 3, determines what action to take, in this case to join a channel.

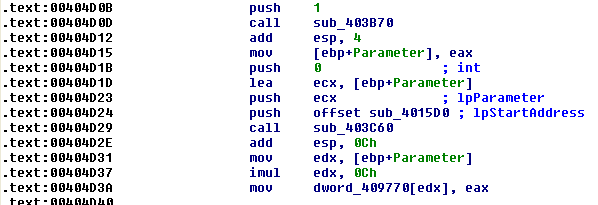
1. .d $ARG1 $URL

Downloads a file from a URL to a temporary directory and executes it.

The options for $ARG1 are "x", "u", "a", "t", and "c". If $ARG1 isn't "x" or "u", then the performs an HTTP GET request using the Windows WinINet API for the page http://API.wipmania.com, which is used to determine the country that the malware is connecting from. The HTTP GET request to the geolocation service simply returns an html page with the public ip and 2-character country code for the computer that the malware is accessing the service from.



The $URL parameter appears to be encrypted using some kind of xor encryption and has a string of numbers followed by a single alphabetic character prepended to the URL. If you try to run this command with a plaintext URL then it will not work, as the URL will be mangled when the malware attempts to decrypt it. The function responsible for the decryption is located at 4033A0h.



The part of the code that actually downloads and executes the file is located in sub\_4014D0. This malware creates a new thread to download the file so that it doesn’t block the main irc thread.

1. .f $IP $PORT $ITERATIONS

Performs a dos attack on a remote computer. In this attack, the malware creates 0x39 TCP sockets and then connects them all to the remote host at the ip and port given. Then the program sleeps for 100 milliseconds, and then closes all of the sockets without calling the disconnect function, leaving them connected to the remote server. This is done in a loop, with the number of iterations specified as the final argument of the command string so that the attacker can control how many connections are created.

Like with the .d command, the string containing the ip address appears to be encrypted (it's decrypted with the same function that decrypts that URL for the .d command, sub\_4033A0).

## What are the time and local system dependent features?

* + 1. Not relevant to our specific malware

## What is method and means by which this sample communicates to the external environment?

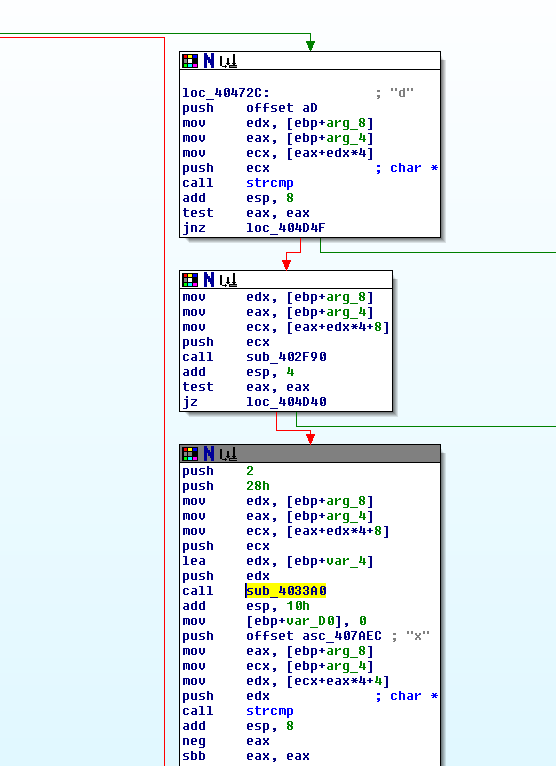
This malware will copy itself to removable and network drives and will locate files there that will activate once users open those drives. Once activated the malware will activate and start its payload.

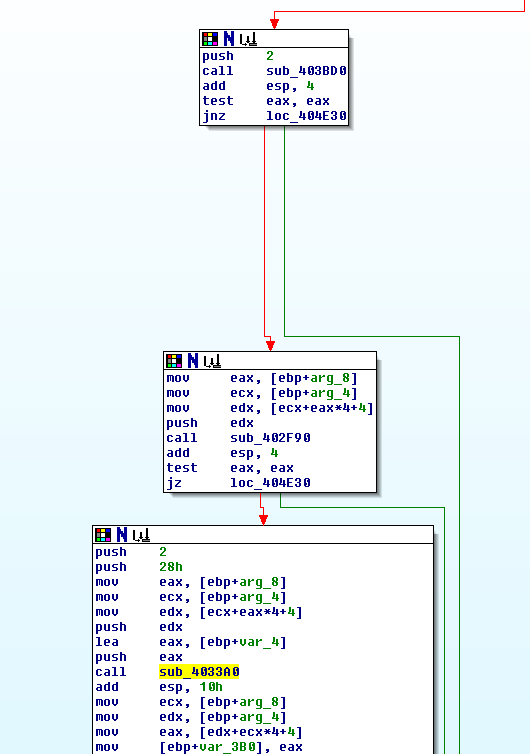
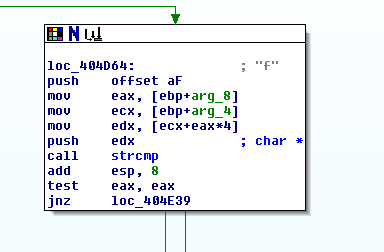
## 

## What is the original infection vector and propagation methodology?

The original infection vector would be downloading this file off the internet, opening an infected network drive or infected removable drive. The file will can automatically run itself if the network/removable drives have the auto start files.

## What use does this sample make of encryption for storage, communication?

The IRC commands .d and .f expected encrypted URL and IP address, respectively, and the malware will decrypt them to, so it can function correctly at 



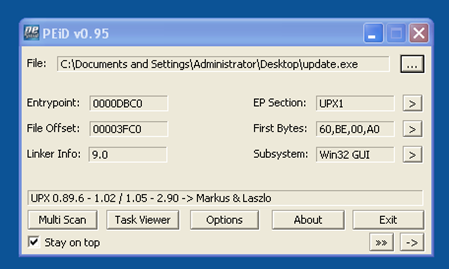
sub\_4033A0h.

## What self-modifying or encrypted code does this sample employ?

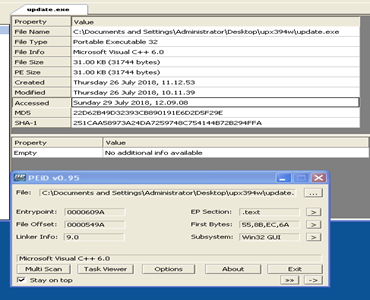
This malware does not implement any forms of encryption or self-modifying other than using UPX to pack itself.

## What ancillary information is available concerning the development of this sample compiler type, country of origin, author names/handles, etc.)

When first receiving the sample, we saw that is was packed with UPX. We knew that we had to unpack this UPX using the upx394w tool in order to continue our observations of the malware. When inspecting the sample further, we saw that version of UPX used was UPX 0.89.6-1.02 / 1.05 – 2.90 and this particular UPX was created by Markus and Laszlo. Additional information about this UPX can be seen below along with the tool that we used to observe this, PEiD.



Once installing and downloading tools to unpack this malware we were able to proceed at examining the malware. Upon further inspection of this sample, we see that there is a wide array of ancillary information available. First, we can see that this code was compiled using Microsoft Visual C++ 6.0. This compiler type is fairly old, so we can guesstimate that this malware was created a few years ago. We observed this by using the PEiD tool along with the CFF Explorer tool. Screenshots of these observations are be shown below.



Another key observation was figuring out when this sample was created. Upon examining the PE file header and using UNIX calculator tools for Time Date Stamp analysis we were able to see that this program was created on Tuesday, May 10, 2016. This corresponds well will our previous assumptions based on the fact the compiler used was quite old and that the malware has the possibility to connect to Windows 10 which came out in 2015. Screenshots of this observation can be shown below.

