ANALYSIS OF SAKUREL MALWARE

By

Drew Kerby

Alk203

"On my honor, as a Mississippi State University student, I have neither given nor received unauthorized assistance on this academic work."

CSE 6363 Software Reverse Engineering
Class Section 1
Dr. Wesley McGrew
April 17th, 2017

Introduction

The purpose of this assignment is to conduct further research and analysis on a version of the *Sakurel* malware sample used to attack the aerospace industry in 2014 [1]. This report will first provide some background on the *Sakurel* sample, then provide an analysis of the advanced anti-reverse engineering techniques used, as well as the process the malware analyst used to circumvent these attempts at obfuscation. Next, a discussion on the various files created by the malware and the persistence mechanism the malware sample uses to stay on the target's machine for extended periods of time. Lastly, the command & control (C2) server was analyzed to the best of the malware analyst's present skills, and this analysis will be discussed in the context of the intended use of the malware sample, namely data exfiltration.

Malware Identification

MD5	c869c75ed1998294af3c676bdbd56851
SHA1	dd3a61eed9c454cf96d882f290abc86108ffeea5
Common Name	Sakurel, Sakula

VirusTotal.com Report

The malware sample report was found on VirusTotal.com using its MD5 hash value. The link to the full report can be found in the references section [10], and a summary will be provided here, due to the considerable length of the report. Out of fifty-seven total antivirus engines used to scan the sample, forty-five detected the malware. It was noted that the signing certificate of the malware was revoked by the issuer, and the malware appeared to be packed. The file metadata indicated that the sample was a Microsoft Windows executable targeting the Intel x86 architecture. Lastly, the report indicated that when executed, the malware sample manipulated files, created processes, and made HTTP requests to the endpoint *oa[.]ameteksen.com*, among other behaviors characteristic of the sample that are outlined in more detail in the full report [10].

Selection of Sample

After browsing the APTnotes repository [2] for a suitable malware family to focus on, I found a security response report from Symantec [1] detailing campaigns targeting the aerospace, energy, and healthcare industries. In this report, MD5 hashes were listed for several of the samples used by Black Vine in their campaigns of recent years. Through search by MD5 hash on VirusShare.com, I located a sample of a Trojan known as *Sakurel* that was used to attack a European aerospace industry in the first half of 2014, according to the Symantec report [1]. Samples of the newer *Mivast* malware used in Black Vine's attack on Anthem [1], a healthcare insurance company, were unable to be located on VirusShare.com or similar sites. A *Mivast* sample would have been preferred due to its more recent creation.

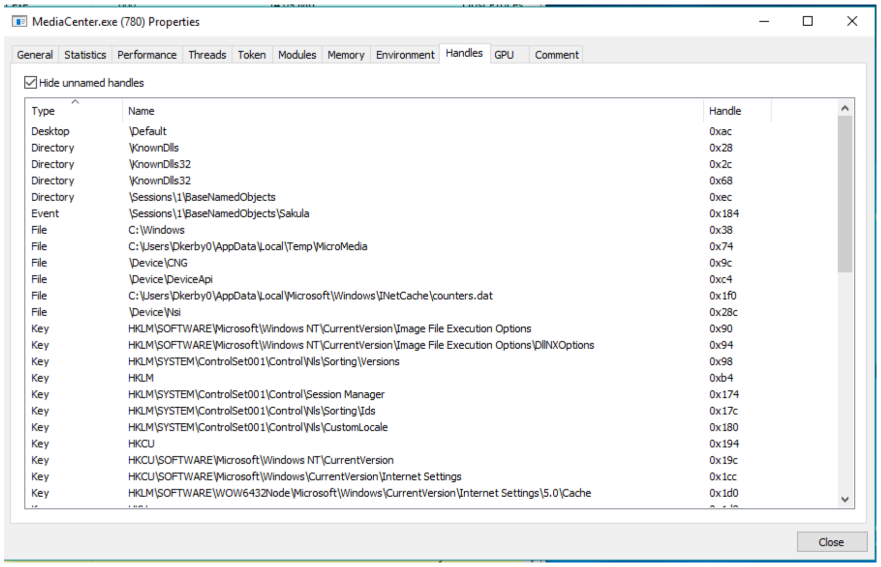
Anti-Reverse Engineering Techniques

Initial static analysis of the *Sakurel* executable file obtained from VirusShare.com [10], named *MediaCenter.exe*, did not suggest much complexity. The number of imported Windows APIs was very low. However, the many imports related to heap and virtual memory allocation, such as VirtualAlloc and HeapAlloc, suggested that this sample may allocate chunks of memory at runtime [7]. Additionally, the presence of LoadLibraryA and GetProcAddress, coupled with the large size of the initial executable file (138kb), suggested that this sample obfuscated much of its functionality to prevent analysis by a malware analyst [7]. At this point, with static analysis techniques stalled, the malware was run inside of a Windows 10 virtual machine in order to determine further information about the malware. Using Process Hacker 2.0 [12], the properties of the *MediaCenter.exe* process were examined. This provided two more key details about the malware. Examining the strings present in the memory provided definitive proof that the malware sample used packing and obfuscation techniques, as there were many error messages, DNS names, and Windows APIs that were not present during static analysis of the initial executable file. The second key detail provided was a handle to the directory *C:\Users\[username] \AppData\Local\Temp\MicroMedia*, as shown in Figure 1. These details were further examined in subsequent analysis.

Once it was confirmed that the malware sample unpacks itself in memory, several attempts were made to extract the unpacked DLL from memory with an intact import table. Without an intact import table [8], IDA Pro 5.0 did not properly name the Windows APIs used by the malware sample, making through analysis difficult if not impossible. First, Import Reconstructor v1.6 was [5] was used to attach to the running MediaCenter.exe process, however only Microsoft-authored DLLs were found by the tool. The DLL containing the malware C2 server functionality could not be located. Next, LordPE [3] and Scylla[4] were both used to attempt to create a dump of the DLL containing the code written by the author of the malware sample. Attempts to locate and dump the DLL were unsuccessful using these tools. Following this, Process Dump v2.1 [6] was used to successfully find and dump a hidden module in the memory space of the running process of the *Sakurel* sample. The command “pd64.exe -pid 4388” was used in order to create this memory dump. The dump was identified as a 32-bit DLL in PE format, although lacking an import table, so attempts to statically analyze this sample were not successful due to the lack of Windows APIs in the disassembly by IDA Pro 5.0. An attempt was then made to use a manual technique described in Practical Malware Analysis [11]. The dumped DLL in was loaded into Immunity Debugger, and the location of each Windows API in IDA Pro in the dumped DLL was cross-referenced to the same location in memory in Immunity

Debugger, however Immunity Debugger did not properly load the names of the imported Windows APIs either, so this attempt at manually reconstructing the import table was unsuccessful.

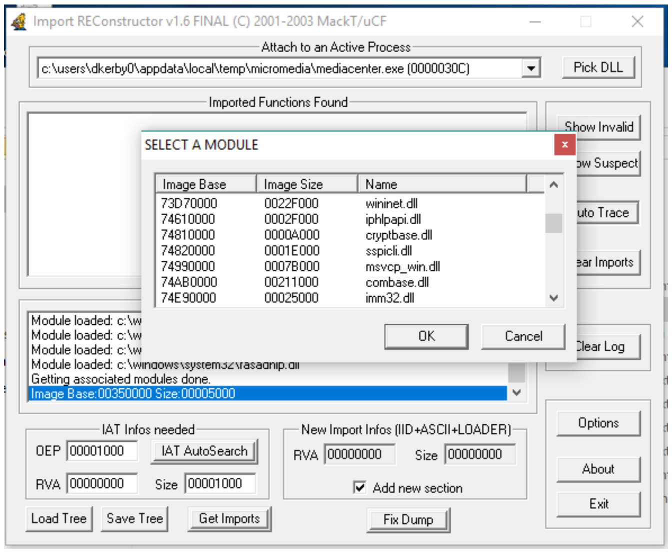
When all other options were exhausted, a successful attempt at reconstructing the import table of the dumped .dll was made by using the command “pd64.exe -g -pid 4388”, which forced the generation of the PE headers from scratch, ignoring the existing PE headers of the loaded .dll, as shown in Figure 3. Via a combination of static and dynamic analysis using IDA Pro 5.0 and Immunity Debugger, it was found that the subroutine at 004014C0h in the MediaCenter.exe executable, named createImportTableInMemory by the malware analyst, used the API functions LoadLibrary and GetProcAddress to load the additional Windows APIs found in the hidden .dll dumped from memory [7]. The .dll, once loaded, was then unlinked to avoid detection by tools such as ImportReconstructor [5], LordPE [3], and Scylla [4]. Anti-reverse engineering techniques such as this were likely the reason for the sparseness of the Symantec report [1] on *Sakurel*, as attempts to debug the dumped DLL in a standalone fashion were unsuccessful. With a working import table in the memory dump of the hidden module loaded at runtime by the sample, as well as the knowledge of the files created at the location *C:\Users\[username]\AppData\Local\Temp*, static analysis was continued.



[✓ Hide unnamed handles]

Type	Name	Handle
Desktop	\Default	0xac
Directory	\KnownDlls	0x28
Directory	\KnownDlls32	0x2c
Directory	\KnownDlls32	0x68
Directory	\Sessions\1\BaseNamedObjects	0xec
Event	\Sessions\1\BaseNamedObjects\Sakula	0x184
File	C:\Windows	0x38
File	C:\Users\Dkerby0\AppData\Local\Temp\MicroMedia	0x74
File	\Device\CNG	0x9c
File	\Device\DeviceApi	0xc4
File	C:\Users\Dkerby0\AppData\Local\Microsoft\Windows\INetCache\counters.dat	0x1f0
File	\Device\Nsi	0x28c
Key	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options	0x90
Key	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\DllNXOptions	0x94
Key	HKLM\SYSTEM\ControlSet001\Control\Nls\Sorting\Versions	0x98
Key	HKLM	0xb4
Key	HKLM\SYSTEM\ControlSet001\Control\Session Manager	0x174
Key	HKLM\SYSTEM\ControlSet001\Control\Nls\Sorting\Ids	0x17c
Key	HKLM\SYSTEM\ControlSet001\Control\Nls\CustomLocale	0x180
Key	HKCU	0x194
Key	HKCU\SOFTWARE\Microsoft\Windows NT\CurrentVersion	0x19c
Key	HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings	0x1cc
Key	HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Internet Settings\5.0\Cache	0x1d0

Figure 1. Strings in running process showing path to relocated executable.



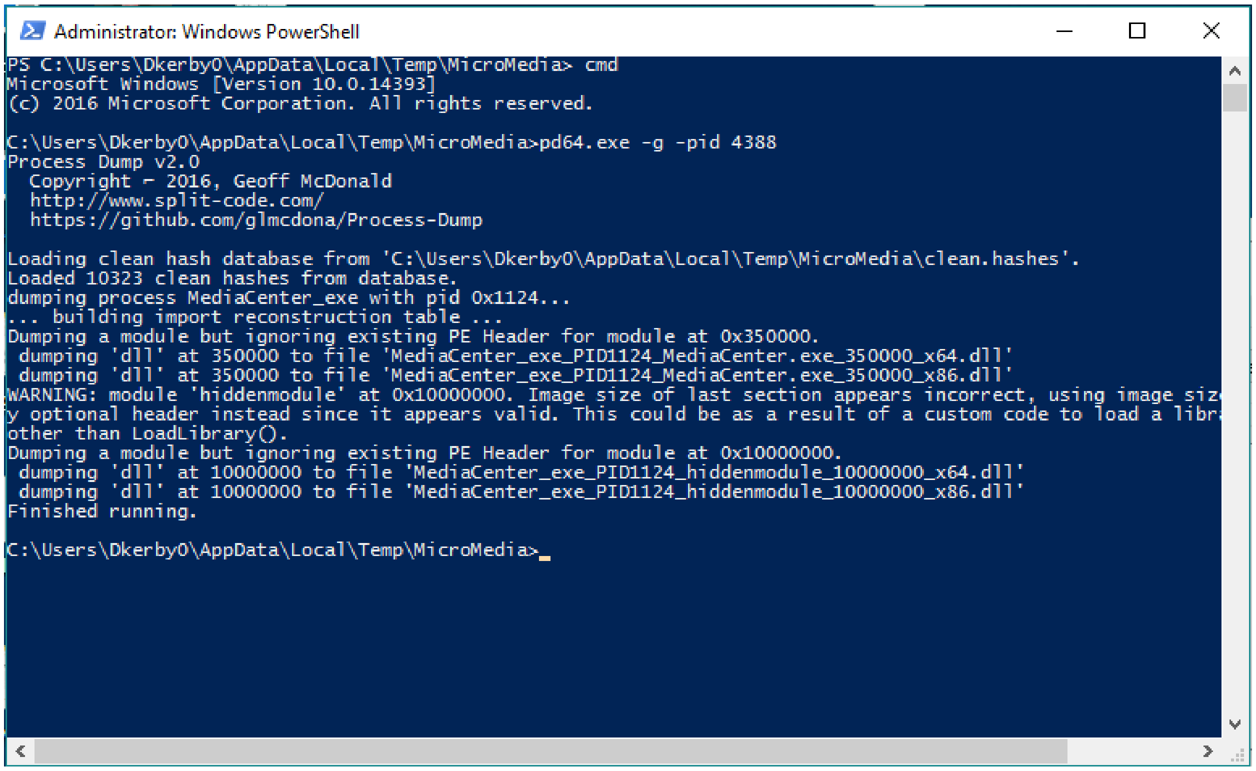
c:\users\dkerby0\appdata\local\temp\micromedia\mediacenter.exe (0000030C)

Image Base	Image Size	Name
73D70000	0022F000	wininet.dll
74610000	0002F000	iphlpapi.dll
74810000	0000A000	cryptbase.dll
74820000	0001E000	sspicli.dll
74990000	0007B000	msvcp_win.dll
74AB0000	00211000	combase.dll
74E90000	00025000	imm32.dll

```
Module loaded: c:\w
Module loaded: c:\w
Module loaded: c:\w
Module loaded: c:\windows\system32\rasadhlp.dll
Getting associated modules done.
Image Base:00350000 Size:00005000
```

OEP 00001000
RVA 00000000 Size 00001000
RVA 00000000 Size 00000000
[✓ Add new section]

Figure 2. Attempt at using Import Reconstructor to create dump of hidden DLL module.



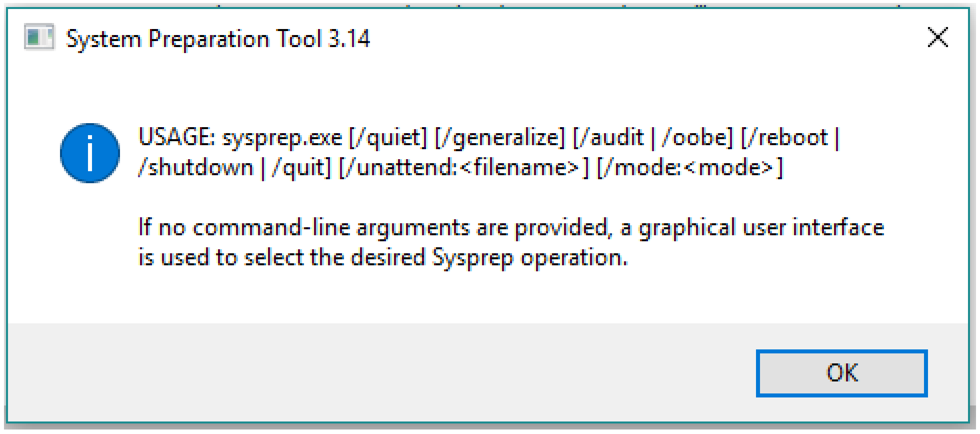
```
PS C:\Users\Dkerby0\AppData\Local\Temp\MicroMedia> cmd
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Dkerby0\AppData\Local\Temp\MicroMedia>pd64.exe -g -pid 4388
Process Dump v2.0
  Copyright - 2016, Geoff McDonald
  http://www.split-code.com/
  https://github.com/glmcdona/Process-Dump

Loading clean hash database from 'C:\Users\Dkerby0\AppData\Local\Temp\MicroMedia\clean.hashes'.
Loaded 10323 clean hashes from database.
dumping process MediaCenter_exe with pid 0x1124...
... building import reconstruction table ...
Dumping a module but ignoring existing PE Header for module at 0x350000.
 dumping 'dll' at 350000 to file 'MediaCenter_exe_PID1124_MediaCenter.exe_350000_x64.dll'
 dumping 'dll' at 350000 to file 'MediaCenter_exe_PID1124_MediaCenter.exe_350000_x86.dll'
WARNING: module 'hiddenmodule' at 0x10000000. Image size of last section appears incorrect, using image siz
y optional header instead since it appears valid. This could be as a result of a custom code to load a libr
other than LoadLibrary().
Dumping a module but ignoring existing PE Header for module at 0x10000000.
 dumping 'dll' at 10000000 to file 'MediaCenter_exe_PID1124_hiddenmodule_10000000_x64.dll'
 dumping 'dll' at 10000000 to file 'MediaCenter_exe_PID1124_hiddenmodule_10000000_x86.dll'
Finished running.

C:\Users\Dkerby0\AppData\Local\Temp\MicroMedia>_
```

Figure 3. Successful attempt at dumping hidden DLL module from using Process Dump v2.1 [6].

Installation and Persistence

Once the loader, MediaCenter.exe, has unpacked *Sakurel's* DLL into main memory and shifted execution to subroutine relocateAndInstallExecutable (10002F9Fh) inside of the DLL, the sample takes action to persist itself. First, the malware sample creates a new directory called "MicroMedia" at the location *C:\Users\[username] \AppData\Local\Temp\*. Next, the malware sample moves its executable, *MediaCenter.exe*, from the location it was originally launched from, into the newly created directory. The executable is then deleted from the old location. A second file is also placed in this folder, MicroSoftSecurityLogin.ocx. An attempt is made to install this file as an ActiveX module using the *regvr32.exe* utility built into the Windows OS [7]. Analysis of this file revealed that MicroSoftSecurityLogin.ocx contained almost no malware author-written code. This DLL contains numerous functions relating to Microsoft's OLE framework [9], so it is likely that this DLL is included in the unpacked malware executable as a means of running the fake System Preparation Tool GUI code. It should be noted that in the event of an error during the installation of the OLE framework code [9], a message box is created that displays the message "fail to install activex. Maybe you should reinstall it". A registry key is then created at *SOFTWARE\Microsoft\Windows\CurrentVersion\Run\* as a persistence mechanism for this DLL once it is successfully registered. A registry entry is also created in this same location to set up persistence for the main executable, *MediaCenter.exe*.

The malware sample also places a non-descript file, *25485406.dat*, in the *C:\Users\[username] \AppData\Local\Temp\*, which is a 64-bit DLL. This file could not be examined with the free version of IDA Pro 5.0 due to its lack of 64-bit support, but an examination of the strings in this file suggest that it contains a few interesting features, including imported Windows APIs such as CreateProcess, GetModuleFilename, and ShellExecuteEx [7], as well as information for 3 different signing certificates, suggesting that it may install the signing certificates located inside of the file, or potentially execute shellcode. The string "sysprep.exe" is also found in *25485406.dat*, so it is likely that the code to generate the dialogue box mimicking the System Preparation Tool comes from this location as well, as shown in Figure 4, with the necessary OLE framework code to execute it having been installed previously as a falsified ActiveX module. The sample first checks if the user is an administrator on the system, and if so, spawns a process to execute the *.dat* file using "rundll32 /s" as the command to execute with location of the file path as an argument [7]. With the files moved to a more discreet location and the persistence mechanism of the malware sample in place, control is then shifted to a loop that takes commands from the malware's C2 server.



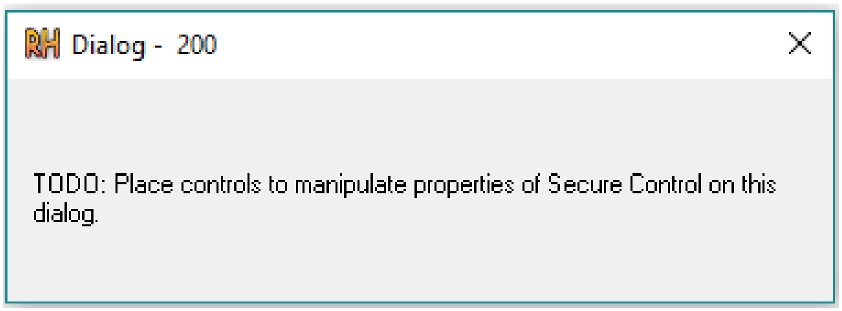
System Preparation Tool 3.14

```
USAGE: sysprep.exe [/quiet] [/generalize] [/audit | /oobe] [/reboot |
/shutdown | /quit] [/unattend:<filename>] [/mode:<mode>]

If no command-line arguments are provided, a graphical user interface
is used to select the desired Sysprep operation.
```

Figure 4. Dialogue box produced by the malware sample to mislead users.

Attribution and Artifact of Continued Development

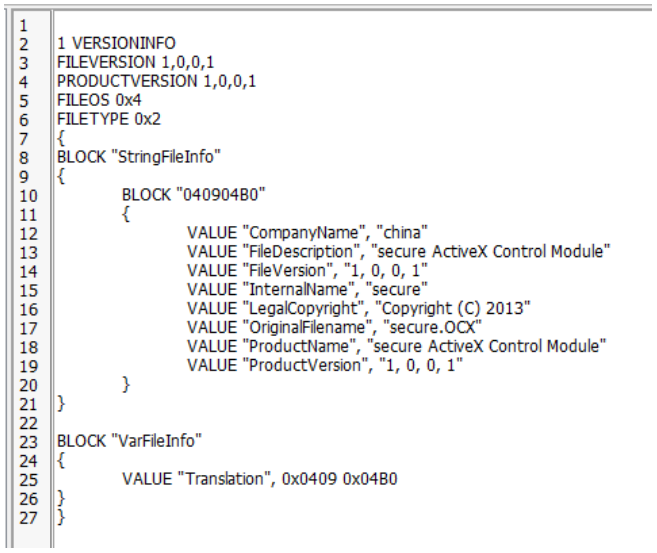
In the resource section of *MicroSoftSecurityLogin.ocx*, a dialogue box was found (Figure 5), showing that the malware author(s) were still actively developing the malware at the time of compilation, and had more features planned to mislead unsuspecting users infected with future malware samples. Additionally, Figure 6 shows a Company Name attribute found in the resource section, which is set to China.



Dialog - 200

TODO: Place controls to manipulate properties of Secure Control on this dialog.

Figure 5. Dialogue box showing planned work.

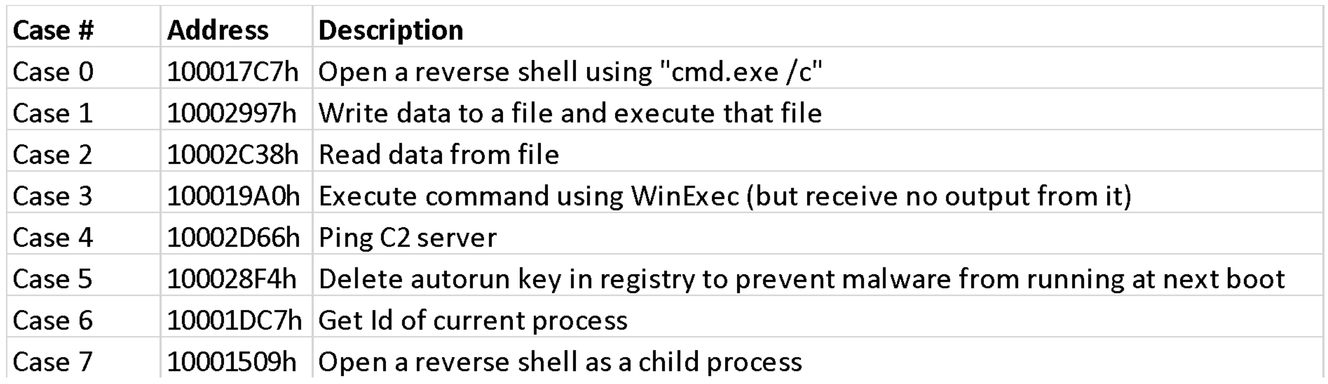


```

1 VERSIONINFO
FILEVERSION 1,0,0,1
PRODUCTVERSION 1,0,0,1
FILEOS 0x4
FILETYPE 0x2
{
BLOCK "StringFileInfo"
{
        BLOCK "040904B0"
        {
                VALUE "CompanyName", "china"
                VALUE "FileDescription", "secure ActiveX Control Module"
                VALUE "FileVersion", "1, 0, 0, 1"
                VALUE "InternalName", "secure"
                VALUE "LegalCopyright", "Copyright (C) 2013"
                VALUE "OriginalFilename", "secure.OCX"
                VALUE "ProductName", "secure ActiveX Control Module"
                VALUE "ProductVersion", "1, 0, 0, 1"
        }
}

BLOCK "VarFileInfo"
{
        VALUE "Translation", 0x0409 0x04B0
}
}
```

Figure 6. Information from a fake ActiveX Control Module.

Communication with C2 Server

The DNS name of the C2 server of this sample of the *Sakurel* malware is *oa[.]ameteksen.com.* The function at 1000258Bh in the hidden DLL contained the logic for sending and receiving commands from the C2 server via HTTP. Before attempting to connect to the internet, the function at 100026A7h in the DLL modifies the Hosts file of the infected machine, as mentioned in the Symantec report [1]. Next, the PC name is retrieved using the GetComputerName API function [7], then encrypted and sent to the C2 server as a parameter in a HTTP Post request. If the initial Post request is unsuccessful, then the malware sample will sleep for 6 hours and try sending a Post request again. If the Post request is successful, an HTTP Get request is sent using the subroutine at 10002601h to fetch the next instruction from the C2 server. If the Get request fails at this particular time, the malware sample will sleep for ~15.5 seconds and then attempt to send another Post request to the C2 to start the cycle over again. If a valid command is received in response to the Get request, one of eight commands is executed in the switch statement, as shown in Figure 7. Due to the inability of the malware analyst to modify the dumped DLL to allow it to be run in a debugger, as well as the very large amount of time attempting to produce memory dump of the hidden DLL module with an intact import table, it was not possible to reverse engineer the C2 server at this time. There were subroutines used that checked for the presence of a debugger, such as the function at location 10005457h, which may have indicated the presence of anti-debugging techniques in the malware, further explaining the difficulty in debugging the malware. However, static analysis proved fruitful, as it was shown that the malware sample had the means to open a reverse shell in a child process, execute commands in a one-time fashion, read, write to, and execute files, as well as delete its own persistence mechanism once the C2 operator had finished with the malware. The presence of these commands would certainly allow for the exfiltration of data from the infected machine, and the ability of the sample to both persist and remove its persistence shows that the malware author(s) did not wish for the attack to be found.



Case #	Address	Description
Case 0	100017C7h	Open a reverse shell using "cmd.exe /c"
Case 1	10002997h	Write data to a file and execute that file
Case 2	10002C38h	Read data from file
Case 3	100019A0h	Execute command using WinExec (but receive no output from it)
Case 4	10002D66h	Ping C2 server
Case 5	100028F4h	Delete autorun key in registry to prevent malware from running at next boot
Case 6	10001DC7h	Get Id of current process
Case 7	10001509h	Open a reverse shell as a child process

Figure 7. Table showing the list of functions available in the C2 server.

Conclusion

In conclusion, the process of reverse engineering the *Sakurel* malware used to attack the European aerospace industry in 2014 [1] proved to be a very difficult task. This was mostly due to the great lengths the malware author(s) went to in order pack and obfuscate the core features of the malware, as well as to prevent the dynamic analysis of the malware sample even after a working dump of the hidden DLL was produced with an intact import table. The functionality present in the malware certainly supports Symantec's claim that the malware was used to conduct cyberespionage [1], and this report added to that body of knowledge. Overall, I learned a great deal in analyzing this malware sample as the subject of the term project, even though the features of the malware did not allow my analysis to proceed as far as I had hoped.

REFERENCES

[1] J. DiMaggio, "The Black Vine Cyberespionage Group", *Symantec Corporation*, Ver. 1.1, Jul. 2015, https://app.box.com/s/0ahidgtzecyx94hgvxoai9kmu5r6yw49 (accessed Jan 30, 2017).

[2] K. Bandla and D. Wescott, "APTnotes.csv", *APTnotes*, Dec. 2015, https://github.com/aptnotes/data/blob/master/APTnotes.csv (accessed Jan 30, 2017).

[3] "LordPE", http://www.softpedia.com/get/Programming/File-Editors/LordPE.shtml (accessed April 16, 2017).

[4] NtQuery, "Scylla - x64/x86 Imports Reconstruction", https://github.com/NtQuery/Scylla (accessed April 16, 2017).

[5] MackT, "Import REConstructor 1.6", *Tuts4you.com*, https://tuts4you.com/download.php?view.2475 (accessed April 16, 2017).

[6] McDonald G., "Process Dump v2.1", https://github.com/glmcdona/Process-Dump (accessed April 16, 2017).

[7] "Microsoft API and Reference Catalog", *Microsoft Corporation*, 2017, https://msdn.microsoft.com/en-us/enus/library/ms123401.aspx (accessed Mar 9th, 2017).

[8] "Microsoft Portable Executable and Common Object File Format Specification", *Microsoft Corporation*, Rev. 10, Jun. 2016.

[9] " OLE Concepts and Requirements", *Microsoft Corporation*, 1999, https://support.microsoft.com/en-us/help/86008/ole-concepts-and-requirements-overview (accessed April 16, 2017).

[10] "SHA256: 6b6e92be036b1a67c383d027bafc7eb63cf515006bb3b3c6ca362a2332542801", *VirusTotal.com*, https://www.virustotal.com/en/file/6b6e92be036b1a67c383d027bafc7eb63cf515006bb3b3c6ca362a2332542801/analysis/ (accessed Jan 30, 2017).

[11] Sikorski, M., & Honig, A., "Practical malware analysis: The hands-on guide to dissecting malicious software.", San Francisco: No Starch Press, 2012.

[12] Wen Jia Liu, "Process Hacker", *SourceForge*, http://processhacker.sourceforge.net/ (accessed April 16, 2017).