Detailed Analysis Of Sykipot (Smartcard Proxy Variant)

GIAC (GREM) Gold Certification

Author: Chong Rong Hwa, ronghwa.chong@gmail.com
Advisor: Antonios Atlasis

Accepted: 1st April 2012

Abstract

On January 2012, AlienVault reported a Sykipot variant with smartcard access capability that has drawn high attention in the security industry. The internals of this malware sample, such as flow of the malware, backdoor capabilities, tricks and techniques, and encryption algorithm are described in this paper. Additionally, its backdoor capabilities are compared with the analysis work of another Sykipot variant published by Symantec. This comparison displays the vast improvements that Sykipot has made. And most importantly, this paper facilitates the security analysts or researchers to response and remediate Sykipot infections, analyze the impact of Sykipot infection, decrypt Sykipot encrypted messages, or even design a fake bot to communicate with the attackers for future research works.

1. Introduction

According to Symantec, Sykipot has been used in targeted attacks for the past few years since 2006 (Thakur, 2011). It was mentioned that this malware does only target Government departments, but it also affects other market sectors such as Telecommunications, Computer Hardware, Chemical and Energy.

As reported by AlienVault, this malware is proliferated through spear-phishing email with malicious attachment or link. This malicious payload then deposits the Sykipot malware into the system (Blasco, 2012).

In Thakur's report, Sykipot is analyzed to be a backdoor malware that supports the execution of both command prompt and customized commands remotely. Additionally, it allows uploading or downloading of files, which could possibly allow the attackers to steal information or plant new malwares. And interestingly, it is also reported that this malware could be instructed to dial back to the Command and Control (CnC) server at a delayed time. This feature could possibly impede network forensic using time-pattern. For example, a network analyst would probably miss the connections made by Sykipot, if he chooses to analyze only network connections that are established at a regular interval.

On January 2012, AlienVault reported an interesting Sykipot variant that accesses smartcards of the infected machine (Blasco, 2012). This feature is probably added to facilitate the attacker to access deeper into the network for protected resources.

In this paper, the internals of this smartcard proxy variant (kindly shared by AlienVault) are detailed, to facilitate security analysts or researchers to: response and remediate Sykipot infections; analyze the impact of Sykipot infection; decrypt Sykipot encrypted messages; or even design a fake bot to communicate with the attackers for future research works.

2. Overview of Sykipot (Smartcard Variant) Malware

As depicted in Figure 1, Sykipot has two malware components - Sykipot EXE and DLL. Sykipot EXE is an executable file with Sykipot DLL embedded unencrypted in its resource section (see section 3.2). When the user opens a malicious link or attachment inside the spear-phishing email, Sykipot EXE is then deposited and executed.

Upon executing Sykipot EXE for the first time, it copies itself to its working directory (one level above %temp% directory) as "**dmm.exe**". Sykipot DLL is then saved into this working directory as "**MSF5F9.dat**" in preparation for DLL injection. Following that, Sykipot EXE monitors for the presence of Outlook, Firefox and Internet Explorer, and inject Sykipot DLL into them (see section 3.1).

The Sykipot DLL is observed to perform key logging and clipboard copying in one thread; and opens a backdoor to the CnC server in another. The functionalities this malware offers ranges from remote execution of backdoor commands, to access secured resources that requires authentication against smartcard (see section 4).

As a mean to survive reboot in a stealthy manner, Sykipot EXE relocates itself to the start up folder as "**taskmost.exe**", only upon closure of the Windows session; and removes traces in the start up folder when run. This inevitably impedes live system forensic when start-up entry points are inspected (see section 3.4).

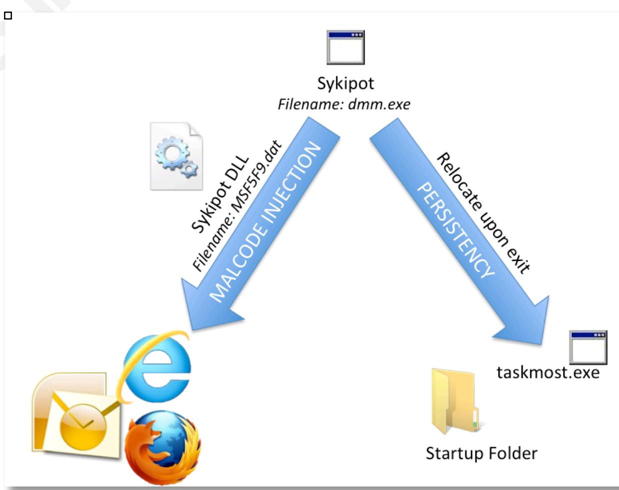


Figure 1. Overview of Sykipot

Chong Rong Hwa, ronghwa.chong@gmail.com

3. Analysis of Sykipot EXE

The filename, MD5 hash and size of this particular sample are **dmm.exe** (or **taskmost.exe**), B0F9DC538F08E49C4B0DA93972BC48A3 and 69632 bytes respectively. The primary purpose of Sykipot EXE is to drop and inject Sykipot DLL into Outlook, Firefox and Internet Explorer (see section 3.2); and its secondary purpose is to maintain persistent in the system (see section 3.4).

3.1. Flow Of Sykipot EXE

Figure 2 describes the flow of the Sykipot EXE (**dmm.exe**) derived through static code analysis, and verified using behavioral analysis and debugging.

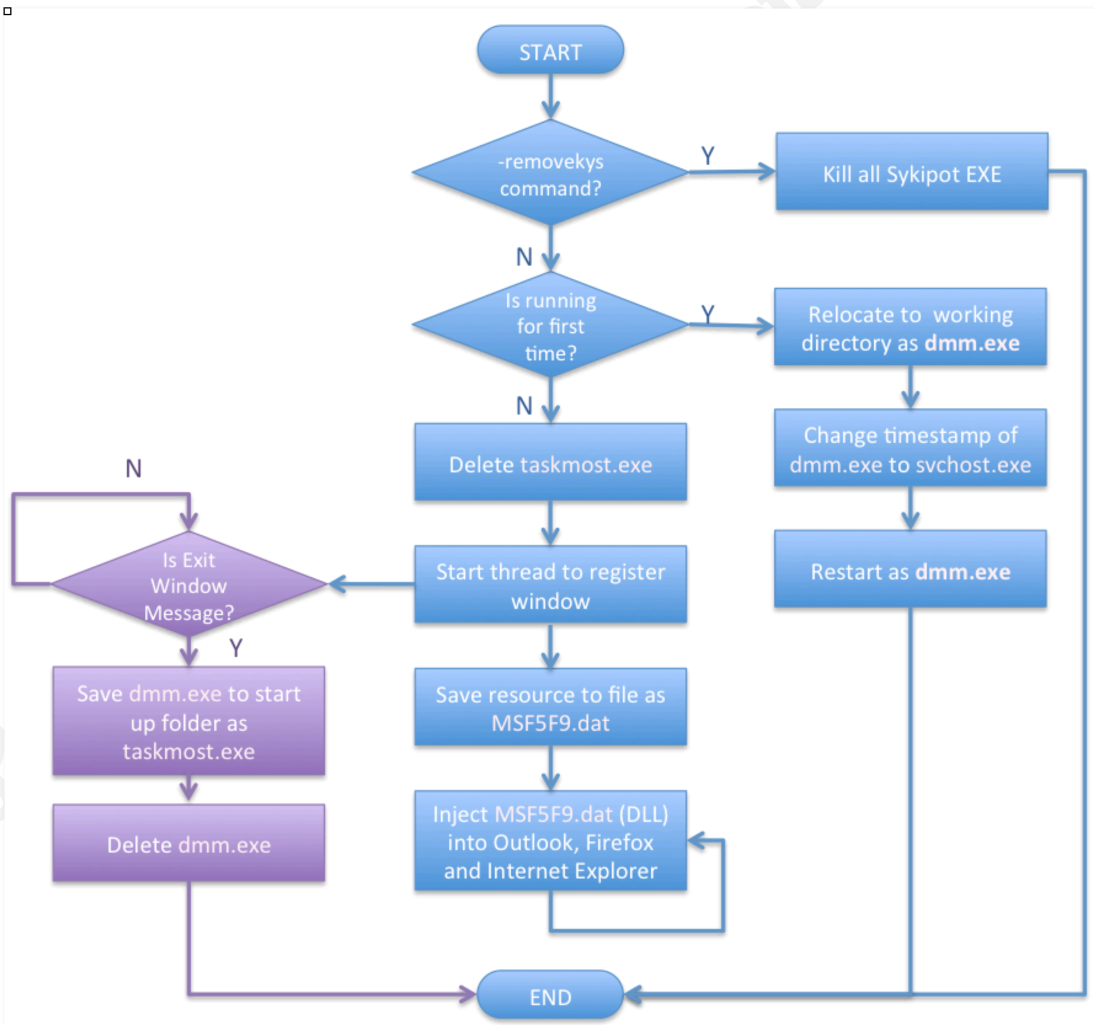


Figure 2. Flow of Sykipot EXE

Chong Rong Hwa, ronghwa.chong@gmail.com

As described in the flowchart above, this malware also has the ability to uninstall itself through command line with argument "**-removekys**". Otherwise, it would either restart itself in its designated working directory, or run two threads to perform DLL injection and maintain persistency.

3.2. DLL Injection

To perform DLL injection, all processes are enumerated to identify targeted processes - **outlook.exe**, **iexplore.exe** and **firefox.exe** (see Figure 3).

```
.text:00401BC9                 lea     eax, [esp+265Ch+moduleName]
.text:00401BCD                 push    eax             ; Str
.text:00401BCE                 call    ds:_strlwr
.text:00401BD4                 mov     edi, ds:strstr
.text:00401BDA                 lea     ecx, [esp+2660h+outlook]
.text:00401BE1                 lea     edx, [esp+2660h+moduleName]
.text:00401BE5                 push    ecx             ; SubStr
.text:00401BE6                 push    edx             ; Str
.text:00401BE7                 call    edi ; strstr
.text:00401BE9                 add     esp, 0Ch
.text:00401BEC                 test    eax, eax
.text:00401BEE                 jnz     short IsRightProcessToInject
.text:00401BF0                 lea     eax, [esp+265Ch+firefox] ; firefox?
.text:00401BF7                 lea     ecx, [esp+265Ch+moduleName]
.text:00401BFB                 push    eax             ; SubStr
.text:00401BFC                 push    ecx             ; Str
.text:00401BFD                 call    edi ; strstr
.text:00401BFF                 add     esp, 8
.text:00401C02                 test    eax, eax
.text:00401C04                 jz      short notFirefox
.text:00401C06
.text:00401C06 IsRightProcessToInject:                 ; CODE XREF: Inject
.text:00401C06                 mov     edx, [esi]
.text:00401C08                 push    ebp             ; hObject
.text:00401C09                 mov     [esp+2660h+pidToInject], edx
.text:00401C0D                 call    ds:CloseHandle
.text:00401C13                 mov     [esp+265Ch+b_OL_FF_IE_found], 1
.text:00401C18
.text:00401C18 notFirefox:                             ; CODE XREF: Inject
.text:00401C18                 lea     eax, [esp+265Ch+iexplore]
.text:00401C1F                 lea     ecx, [esp+265Ch+moduleName]
.text:00401C23                 push    eax             ; SubStr
.text:00401C24                 push    ecx             ; Str
.text:00401C25                 call    edi ; strstr
```

Figure 3. Targeted Processes For DLL Injection

Chong Rong Hwa, ronghwa.chong@gmail.com

Sykipot DLL is injected into targeted processes using the CreateRemoteThread with LoadLibrary Technique (Kuster, 2003). This technique uses **VirtualAllocEx** to allocate a memory page in the targeted process; **WriteProcessMemory** to write the path of the malicious DLL into allocated memory space of the targeted process; and **CreateRemoteThread** to start a new thread with **LoadLibraryA** as thread entry point to load specified DLL (see Figure 4).

```
.text:0040163C                 call    ds:VirtualAllocEx
.text:00401642                 mov     edi, eax
.text:00401644                 mov     [ebp-28h], edi
.text:00401647                 test    edi, edi
.text:00401649                 jnz     short loc_401650
.text:0040164B                 mov     [ebp-24h], eax
.text:0040164E                 jmp     short loc_4016A9
.text:00401650 ; ---------------------------------------------------------------------------
.text:00401650
.text:00401650 loc_401650:                             ; CODE XREF: InjectDLLIntoProcess+79↑j
.text:00401650                 push    0               ; lpNumberOfBytesWritten
.text:00401652                 push    esi             ; nSize
.text:00401653                 mov     ecx, [ebp+0Ch]
.text:00401656                 push    ecx             ; lpBuffer
.text:00401657                 push    edi             ; lpBaseAddress
.text:00401658                 push    ebx             ; hProcess
.text:00401659                 call    ds:WriteProcessMemory
.text:0040165F                 test    eax, eax
.text:00401661                 jnz     short loc_401668
.text:00401663                 mov     [ebp-24h], eax
.text:00401666                 jmp     short loc_4016A9
.text:00401668 ; ---------------------------------------------------------------------------
.text:00401668
.text:00401668 loc_401668:                             ; CODE XREF: InjectDLLIntoProcess+91↑j
.text:00401668                 push    offset ProcName ; "LoadLibraryA"
.text:0040166D                 push    offset ModuleName ; "Kernel32"
.text:00401672                 call    ds:GetModuleHandleA
.text:00401678                 push    eax             ; hModule
.text:00401679                 call    ds:GetProcAddress
.text:0040167F                 mov     [ebp-2Ch], eax
.text:00401682                 test    eax, eax
.text:00401684                 jnz     short loc_40168B
.text:00401686                 mov     [ebp-24h], eax
.text:00401689                 jmp     short loc_4016A9
.text:0040168B ; ---------------------------------------------------------------------------
.text:0040168B
.text:0040168B loc_40168B:                             ; CODE XREF: InjectDLLIntoProcess+B4↑j
.text:0040168B                 push    0               ; lpThreadId
.text:0040168D                 push    0               ; dwCreationFlags
.text:0040168F                 push    edi             ; lpParameter
.text:00401690                 push    eax             ; lpStartAddress
.text:00401691                 push    0               ; dwStackSize
.text:00401693                 push    0               ; lpThreadAttributes
.text:00401695                 push    ebx             ; hProcess
.text:00401696                 call    ds:CreateRemoteThread
```

Figure 4. DLL Injection Using CreateRemoteThread with LoadLibraryA

Chong Rong Hwa, ronghwa.chong@gmail.com

As Sykipot DLL is embedded unencrypted in the resource section of Sykipot EXE, it could be easily identified using PE parser such as PEview (see Figure 5). This DLL dropped into the Sykipot working directory as **MSF5F9.dat** (mentioned in Figure 2, Flow of Sykipot EXE).

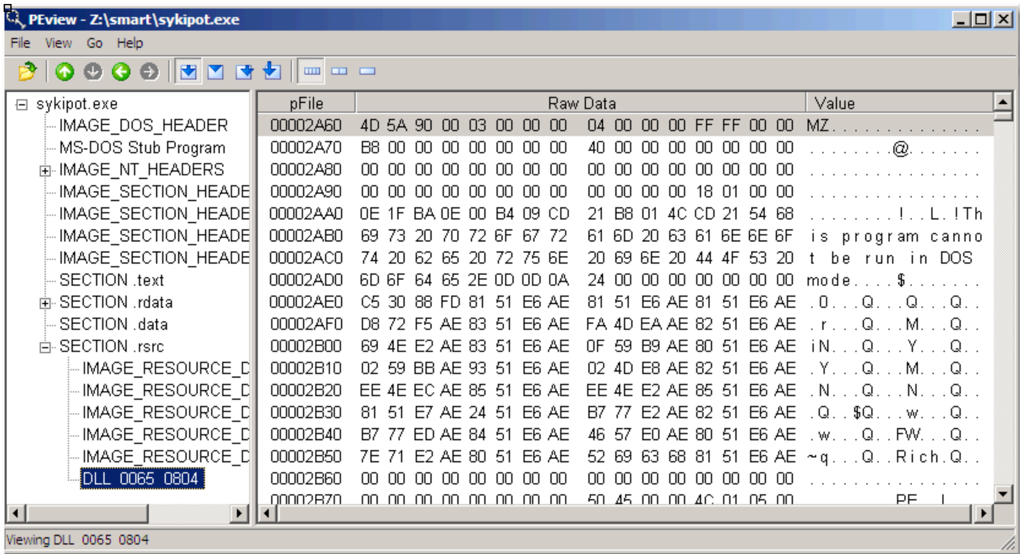


Figure 5. Statically Examine Sykipot EXE using PEview

To impede memory or disk forensic, this DLL disguises itself as a Microsoft related executable file. It appears to be a legitimate "**IPv4 Helper DLL**" created by "**Microsoft Corporation**" (see Figure 6). And certainly, this could possibly pass the eyes of an inexperienced malware analyst when listing DLL using Process Explorer (live forensic tool) or Volatilty dlllist plugin (memory forensic tool).

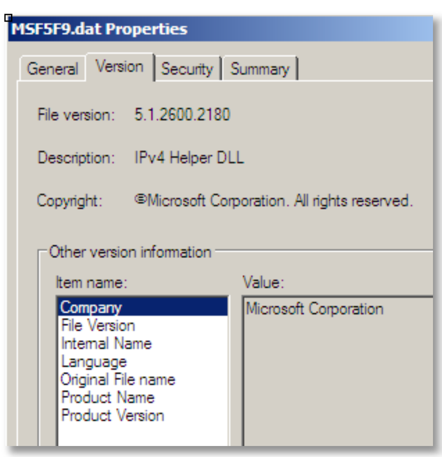


Figure 6. File Properties of Sykipot DLL

Chong Rong Hwa, ronghwa.chong@gmail.com

According to Volatility command reference, DLL injected using this technique would not be flagged as malicious by the Volatility malfind plugin. (Volatilty Command Reference, 2012). Consequently, Sykipot achieves stealth by not hiding itself.

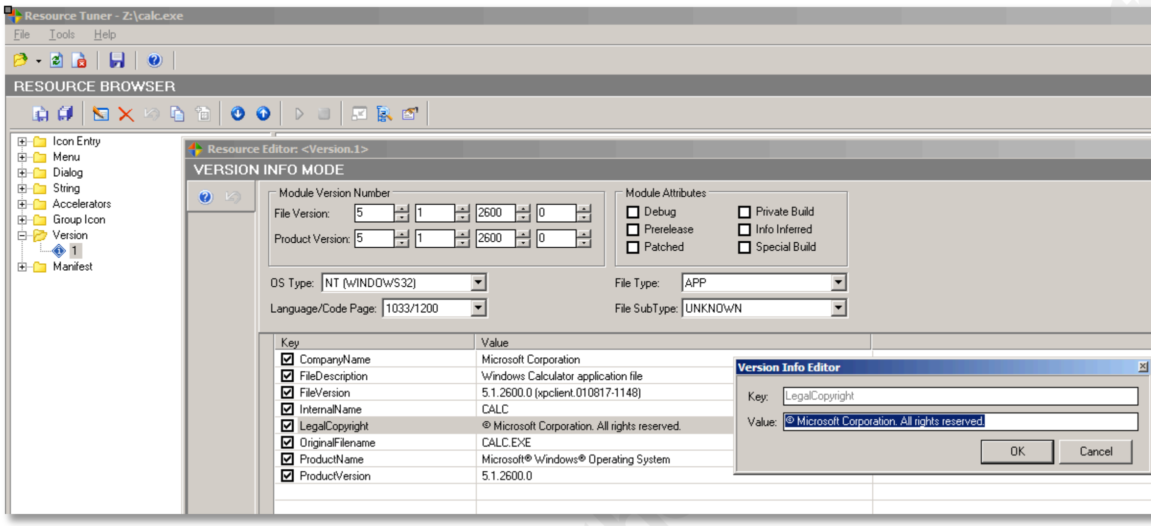


Figure 7. Editing of Version Information

As seen in the figure above, the version information of an executable file can be modified using a resource editor such as Resource Tuner from Heaven Tools (Visual Resource Editor, 2012). Hence, it is not surprising to see malware authors to use this (simple yet convincing) technique to evade detection.

Chong Rong Hwa, ronghwa.chong@gmail.com

3.3. Time Stomping

Like most anti-forensic malwares, it would stomp the timestamp of its executable files to be the same as the system files (see Figure 8). In this instance, Sykipot stomps the timestamp of Sykipot EXE executable file to be the same as svchost.exe (a windows system file). It would probably be filtered and unseen when a disk forensic analyst filters the list of files using timestamp of Window’s system executable files.

```
00401FD4 call    ds:GetSystemDirectoryA
00401FDA lea     edx, [esp+330h+startupFolder]
00401FDE push    offset String2  ; "\\svchost.exe"
00401FE3 push    edx             ; lpString1
00401FE4 call    ds:lstrcatA
00401FEA mov     edi, ds:CreateFileA
00401FF0 push    0               ; hTemplateFile
00401FF2 push    0               ; dwFlagsAndAttributes
00401FF4 push    3               ; dwCreationDisposition
00401FF6 push    0               ; lpSecurityAttributes
00401FF8 push    0               ; dwShareMode
00401FFA push    0               ; dwDesiredAccess
00401FFC push    eax             ; c:\windows\system32\svchost.exe
00401FFD call    edi ; CreateFileA
00401FFF push    0               ; hTemplateFile
00402001 push    0               ; dwFlagsAndAttributes
00402003 push    3               ; dwCreationDisposition
00402005 mov     esi, eax
00402007 push    0               ; lpSecurityAttributes
00402009 push    0               ; dwShareMode
0040200B lea     eax, [esp+344h+localSetting_dmm.exe]
00402012 push    0C0000000h      ; dwDesiredAccess
00402017 push    eax             ; %localsetting%\dmm.exe
00402018 call    edi ; CreateFileA
0040201A lea     ecx, [esp+330h+LastWriteTime_TokenHandle]
0040201E mov     edi, eax
00402020 lea     edx, [esp+330h+LastAccessTime]
00402024 push    ecx             ; lpLastWriteTime
00402025 lea     eax, [esp+334h+CreationTime]
00402029 push    edx             ; lpLastAccessTime
0040202A push    eax             ; lpCreationTime
0040202B push    esi             ; hFile
0040202C call    ds:GetFileTime  ; get file time of svchost
00402032 lea     ecx, [esp+330h+LastWriteTime_TokenHandle]
00402036 lea     edx, [esp+330h+LastAccessTime]
0040203A push    ecx             ; lpLastWriteTime
0040203B lea     eax, [esp+334h+CreationTime]
0040203F push    edx             ; lpLastAccessTime
00402040 push    eax             ; lpCreationTime
00402041 push    edi             ; hFile
00402042 call    ds:SetFileTime  ; set it to malware
```

Figure 8. Time Stomping of Sykipot EXE

3.4. Persistency Mechanism

One other important function of Sykipot EXE is to maintain persistency in a stealthy manner. Sykipot deletes “**taskmost.exe**” from start up folder to remove traces of persistency when run. At the same time, a new thread is started to listen for the following windows messages to detect exit of windows session - WM_QUIT (0X12), WM_DESTROY (0X02), WM_QUERYENDSESSION (0X11) and WM_ENDSESSION (0X16) (see Figure 9).

```
GetModuleFileNameA(0, &ExistingFileName, 0x104u);
SHGetSpecialFolderPathA(0, &startupFolder, CSIDL_STARTUP, 0);
strcat(&startupFolder, "\\");
strcat(&startupFolder, (const char *)"taskmost.exe");
switch ( Msg )
{
  case 2u:                                    // WM_DESTROY
    PostQuitMessage(0);
    CopyFileA(&ExistingFileName, &startupFolder, 0);
    if ( TokenHandle )
    {
      CloseHandle(TokenHandle);
      RevertToSelf();
    }
    exit(0);
    return result;
  case 0x12u:                                 // WM_QUIT
    CopyFileA(&ExistingFileName, &startupFolder, 0);
    if ( TokenHandle )
    {
      CloseHandle(TokenHandle);
      RevertToSelf();
    }
    exit(0);
    return result;
  case 0x11u:                                 // WM_QUERYENDSESSION
    CopyFileA(&ExistingFileName, &startupFolder, 0);
    if ( TokenHandle )
    {
      CloseHandle(TokenHandle);
      RevertToSelf();
    }
    exit(0);
    return result;
  case 0x16u:                                 // WM_ENDSESSION
    CopyFileA(&ExistingFileName, &startupFolder, 0);
    if ( TokenHandle )
    {
      CloseHandle(TokenHandle);
      RevertToSelf();
    }
    exit(0);
    return result;
  default:
    return DefWindowProcA(hWnd, Msg, wParam, lParam);
```

Figure 9. Relocate Sykipot EXE to Survive Reboot

Only when windows exit, Sykipot relocates itself to the start up folder again as “**taskmost.exe**” to survive reboot. Since the executable file only exists in start up folder when required, live analysis would probably miss this executable when start-up entries are inspected (see Figure 9).

Chong Rong Hwa, ronghwa.chong@gmail.com

4. Analysis of Sykipot DLL

The filename, MD5 hash and size of this particular sample are **MSF5F9.dat**, C2821DDE5D309962337434AA6062EAA9 and 58368 bytes respectively. The purpose of the DLL executable file is to log all keystrokes and maintain backdoor for the attacker to remote control the victimized system (see section 4.1). The technical details of the malicious artifacts, backdoor, proxy selection and encryption are covered in section 4.2, 4.3, 4.4 and 4.5 respectively.

4.1. Flow of DLL

Figure 10 and Figure 12 depicts the flow of a key logger thread and a backdoor thread respectively, derived through static code analysis and verified through behavioral analysis and debugging. See section 4.2 for details of malicious file artifacts.

It is evident that this malware is not only interested in logging all keystrokes, it also captures all clipboard contents (see Figure 11). Obviously, this would be for the purpose of a comprehensive information stealing.

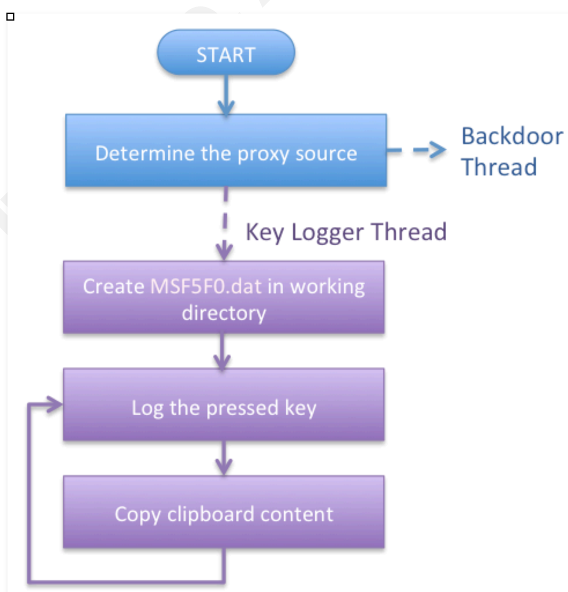
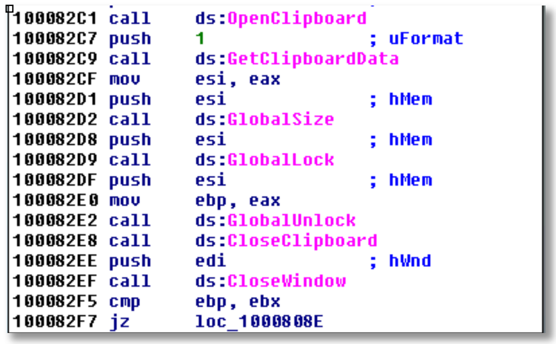


Figure 10. Flow Key Logger Thread

Chong Rong Hwa, ronghwa.chong@gmail.com



```
100082C1 call    ds:OpenClipboard
100082C7 push    1                ; uFormat
100082C9 call    ds:GetClipboardData
100082CF mov     esi, eax
100082D1 push    esi              ; hMem
100082D2 call    ds:GlobalSize
100082D8 push    esi              ; hMem
100082D9 call    ds:GlobalLock
100082DF push    esi              ; hMem
100082E0 mov     ebp, eax
100082E2 call    ds:GlobalUnlock
100082E8 call    ds:CloseClipboard
100082EE push    edi              ; hWnd
100082EF call    ds:CloseWindow
100082F5 cmp     ebp, ebx
100082F7 jz      loc_1000808E
```

Figure 11. Copy Clipboard Data

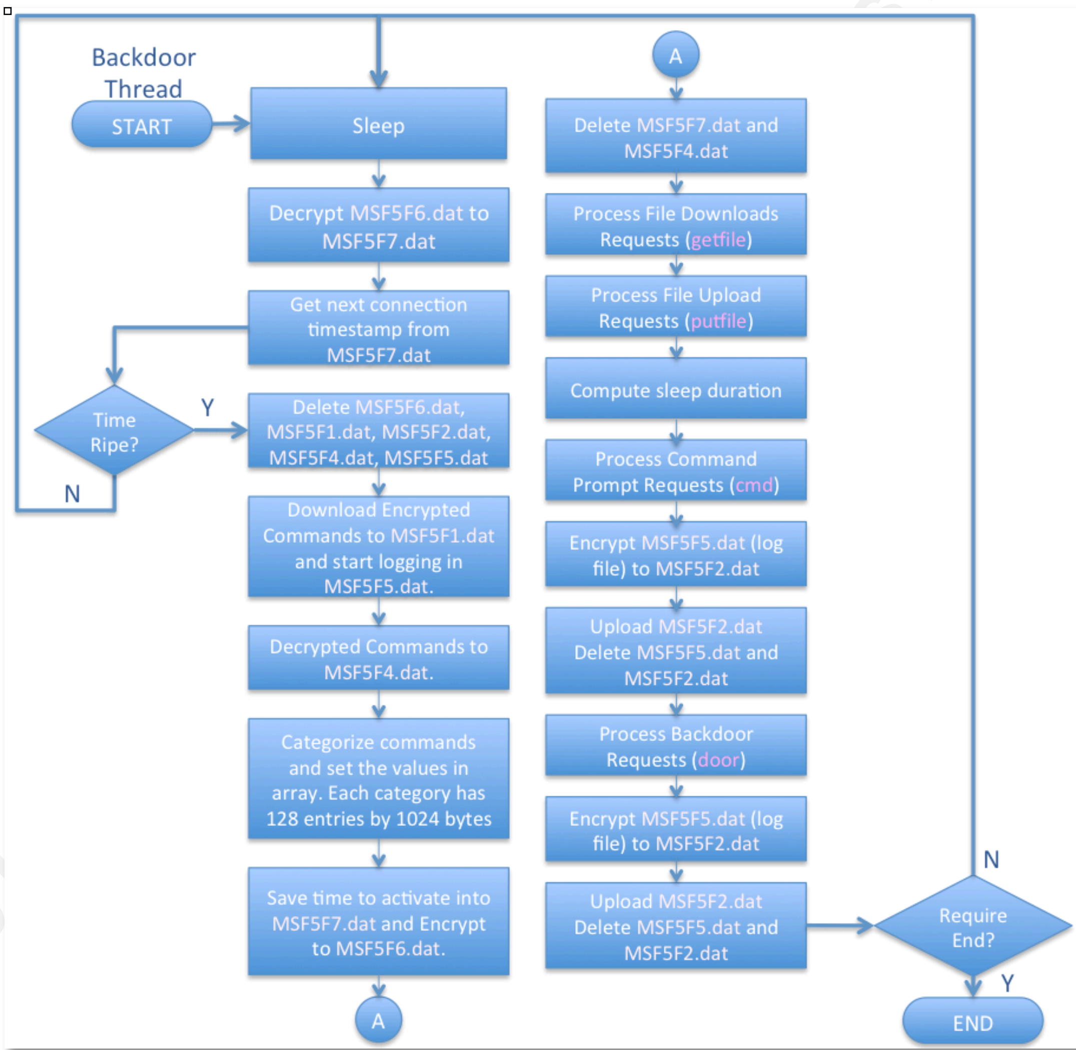


Figure 12. Flow of Backdoor Thread

From the flow above, it is observed that the encrypted commands are downloaded into MSF5F1.dat. The commands are then classified into five different groups, and are found to be the same as the commands described in the Symantec's report – **cmd**, **door**, **getfile**, **putfile** and **time** (Thakur, 2011). As seen in Figure 13, it is analyzed that the contents of each group are stored in a 2D array (a maximum of 128 string entries). The functionality of each group is described in the list below.

- **cmd** contains a list of command-prompt commands.
- **door** contains a list of backdoor commands.
- **getfile** refers to a list of files to be downloaded.
- **putfile** refers to a list a files to be uploaded.
- **time** refers to the next connection time.

```
.data:10012598 ; char bufArray4_putfile[128][1024]
.data:10012598 bufArray4_putfile db 20000h dup(?)
.data:10012598
.data:10032598 ; char bufArray3_GetFile[128][1024]
.data:10032598 bufArray3_GetFile db 20000h dup(?)
.data:10032598
.data:10052598 ; char bufArray5_Time[128][1024]
.data:10052598 bufArray5_Time  db 20000h dup(?)
.data:10052598
.data:10072598 ; char bufArray2_door[128][1024]
.data:10072598 bufArray2_door  db 20000h dup(?)
.data:10072598
.data:10092598 ; char bufArray1_command[128][1024]
.data:10092598 bufArray1_command db 20000h dup(?)
```

Figure 13. Data Type of Command Categories

Chong Rong Hwa, ronghwa.chong@gmail.com

4.2. Malicious File Artifacts

All related executable and configuration files depicted in Figure 14 are stored in the Sykipot's working directory. Figure 15 depicts the code used Sykipot to determine its designated working directory (one level above %temp% directory).

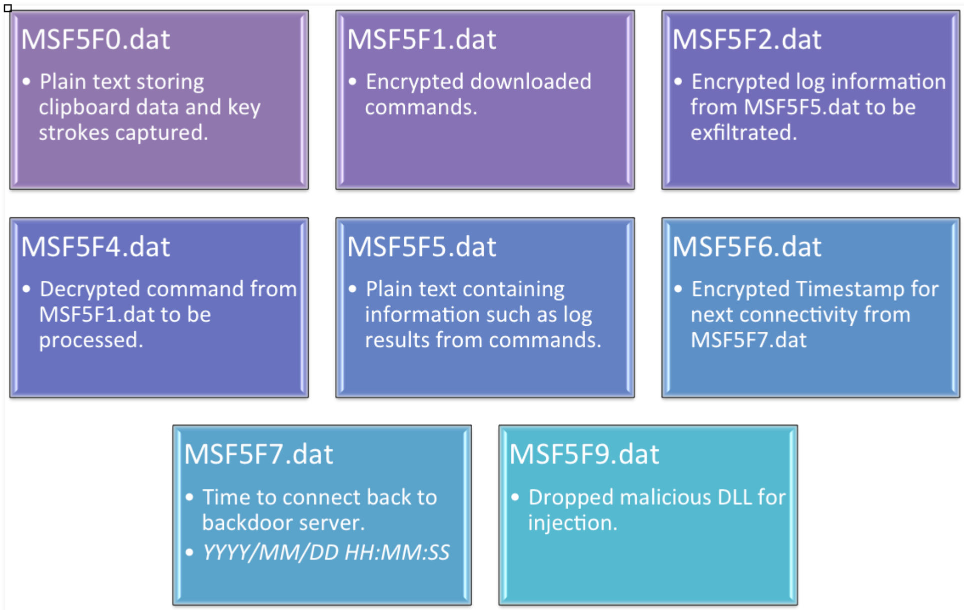


Figure 14. Sykipot File Artifacts

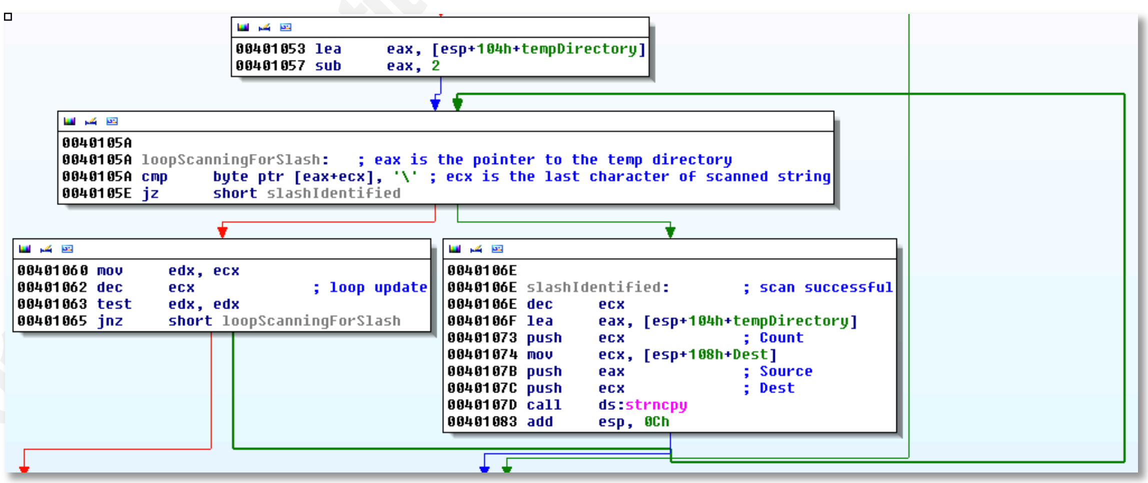


Figure 15. Sykipot Working Directory

Despite the filenames and purpose of all file artifacts are identified, we should not use the file name or path to ascertain if a system is not compromised by Sykipot. This is

Chong Rong Hwa, ronghwa.chong@gmail.com

because the file names used by Sykipot are different in different variants. See table below.

File name	Function
Gtpretty.tmp	Orders from the CnC.
Gdtpretty.tmp	Decrypted version of orders from the CnC.
Pdtpretty.tmp	Log file.
Ptpretty.tmp	Encrypted version of log file.

Table 1. File Artifacts Identified by Symantec

4.3. Backdoor Commands

The backdoor commands can be divided into two main groups, generic and smartcard-specific backdoor commands, which are described in section 4.3.1 and 4.3.2 respectively.

4.3.1. Generic Backdoor Commands

Table 2 compares the list of functionalities identified in this sample against the functionalities reported by Symantec (Thakur, 2011).

Index	Command	Alienvault Identified Variant	Symantec Identified Variant
1	shell	Removed from this variant	Do nothing
2	run	Executes using WinExec	Executes using WinExec
3	reboot	Restarts the computer	Restarts the computer
4	kill	Ends a process	Ends a process
5	process	List processes	Not implemented
6	runtime	List time	Not identified
7	system	Execute a file	Not identified
8	ipconfig	List network configuration	Not identified
9	move	Move file	Not identified
10	del	Secure delete file	Not identified
11	rundll	Load a DLL	Not identified
12	enddll	Unload a DLL	Not identified
13	dir	List directory contents	Not identified
14	port	List TCP and UDP connections	Not identified
15	uninstall	Uninstall Sykipot	Not identified
16	key	Get key logger results	Not identified

Table 2. Backdoor Command Comparison

It is interesting to see the improvements that the malware author has made. The improvement ranges from reconnaissance functionalities to loading/unloading of DLL and secure deletion of file. Figure 16 reveals the pseudo code to secure delete a file by overwriting each byte in the file with “0x00” prior deletion.

```
hFileDelete = CreateFileA(sFileDelete, 0, 0, 0, 3u, 0, 0);
fileDelete_size = GetFileSize(hFileDelete, 0);
CloseHandle(hFileDelete);
v31 = fopen(sFileDelete, "w");
hMSF5F5 = v31;
if ( !v31 )
  return 0;
for ( i = 0; !(v31->_flag & 0x10) && i < (signed int)fileDelete_size; ++i )
{
  fputc(0, v31);                    // zerorise the file
  v31 = hMSF5F5;
}
fclose(v31);
hMSF5F5 = fopen(fileNameFromFileRecon, "a");
if ( !hMSF5F5 )
  return 0;
if ( DeleteFileA(sFileDelete) )
  deleteStatus = "del success!\n";
else
  deleteStatus = "del false!\n";
fprintf(hMSF5F5, deleteStatus);
fclose(hMSF5F5);
```

Figure 16. Secure File Deletion

4.3.2. Smartcard Specific Backdoor Commands

Table 3 tabularizes the smartcard specific backdoor functionalities identified in this sample.

Index	Command	Purpose
1	cl	List certificates associated with private keys
2	cm	Loads ActivClient DLL List of card readers and cards available
3	krundll	Load custom DLL with three exported functions: LoginFunc, PutFunc and GetFunc.
4	kenddll	Unload the custom DLL
5	kshow	Show card login status
6	klogin	Invoke LoginFunc
7	kput	Invoke PutFunc
8	kget	Invoke GetFunc
9	kfile	Set the upload file name
10	kpin	Set the pin value
11	kcert	Set the cert value
12	kheader	Set the header value
13	kreferer	Set the referer value

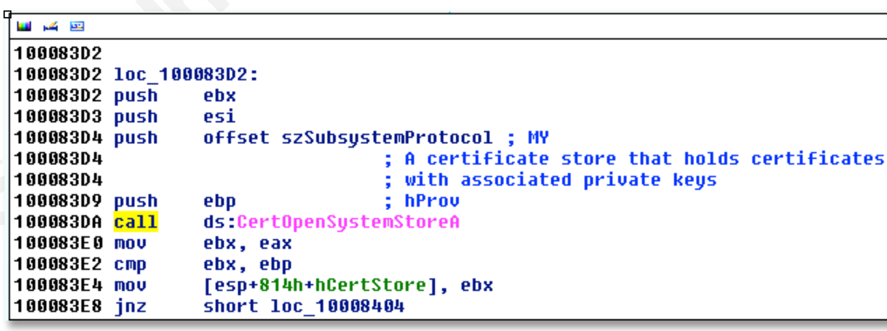
Table 3. Smartcard Specific Backdoor Commands

As the custom DLL (loaded through krundll command) is not available for analysis, it becomes an analysis blind spot. However, its intention can be induced through its exported function name and parameters. The function prototype of the custom smartcard related DLL is analyzed as follows:

- LoginFunc (URL, referer, header, uploadFileName, certificate, PIN, dataout)
- PutFunc (hInternet, putString, referer, header, URL, b_putfile_or_putdata, uploadFileName, certificate, PIN, dataout)
- GetFunc (hInternet, URL, referer, header, uploadFileName, certificate, PIN, dataout)

From the list of smartcard specific backdoor commands, it is not seen to hack the smartcard to extract private certificate. Despite so, it has effectively used the victimized machine as a smartcard proxy, to access the protected resources that require smartcard as 2nd-factor authentication using "**klogin**", "**kput**" and "**kget**" commands.

As mentioned in Table 3, "**cl**" lists all the card issuer and subject of certificates associated with private keys (see dead listings in Figure 17 and Figure 18). However, this does not imply extraction of private key. Additionally, a properly configured smartcard should not allow extract of private key.



```
100083D2
100083D2 loc_100083D2:
100083D2 push    ebx
100083D3 push    esi
100083D4 push    offset szSubsystemProtocol ; MY
100083D4                         ; A certificate store that holds certificates
100083D4                         ; with associated private keys
100083D9 push    ebp             ; hProv
100083DA call    ds:CertOpenSystemStoreA
100083E0 mov     ebx, eax
100083E2 cmp     ebx, ebp
100083E4 mov     [esp+814h+hCertStore], ebx
100083E8 jnz     short loc_10008404
```

Figure 17. Open System Store

Chong Rong Hwa, ronghwa.chong@gmail.com

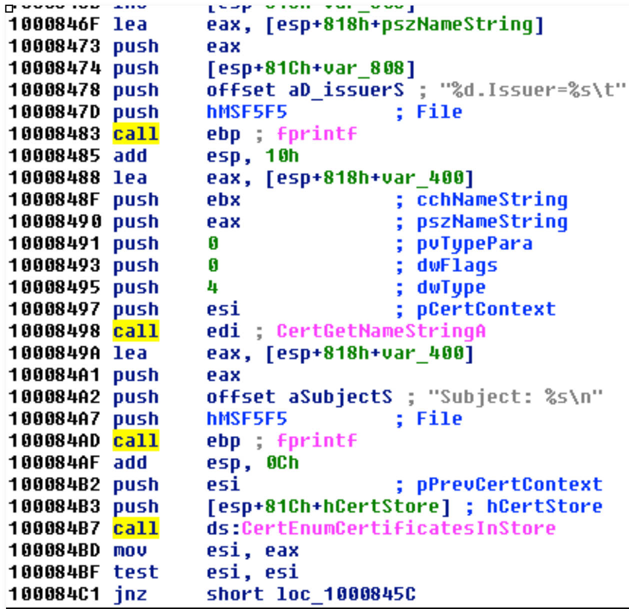


Figure 18. Retrieve Certificate Information

Another interesting command to mention is "**cm**". When this command is invoked, it attempts to load "**acpkcs201.dll**", an ActivClient DLL, to get the list of card readers and card status (see Figure 19).

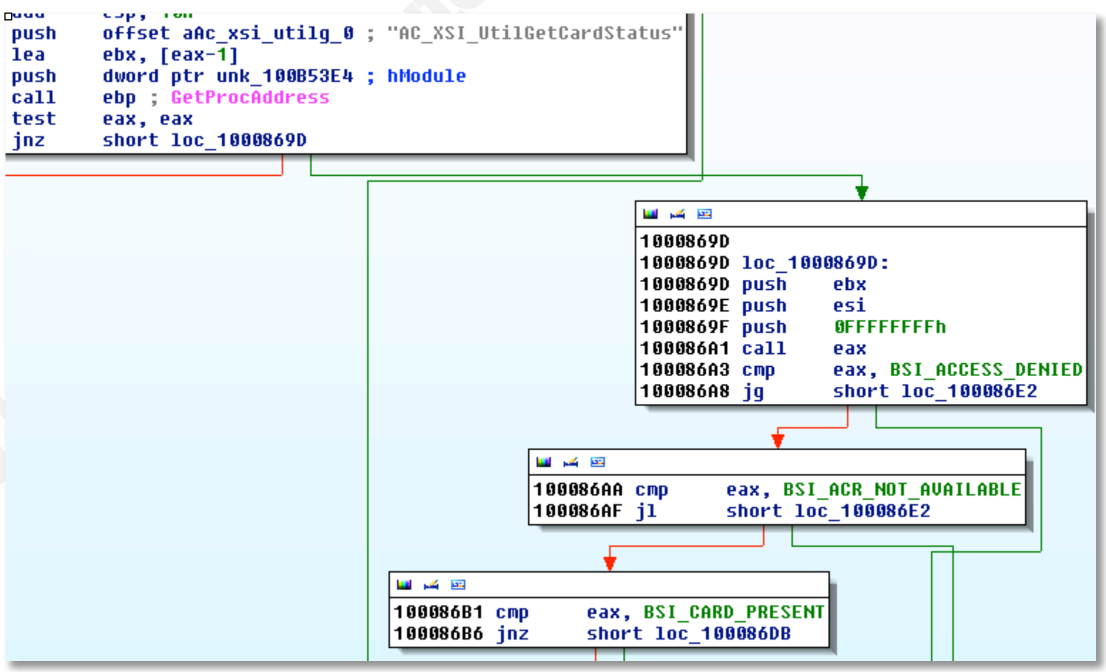


Figure 19. Retrieve Card Status

Chong Rong Hwa, ronghwa.chong@gmail.com

As seen in Figure 20, Sykipot loads acpkcs201.dll (ActivClient DLL) from any of the three possible paths - System directory, “C:\Program Files\ActivIdentity\ActivClient” or “C:\Program Files(x86)\ActivIdentity\ActivClient”. This reveals that the attacker is probably aware that the targeted user is using ActivClient DLL.

```
.text:10008553    call    ds:GetSystemDirectoryA
.text:10008559    push    offset aAcpkcs201_dll ; "\\acpkcs201.dll"
.text:1000855E    push    esi             ; Dest
.text:1000855F    call    strcat
.text:10008564    mov     eax, dword ptr unk_100B53E4
.text:10008569    pop     ecx
.text:1000856A    test    eax, eax
.text:1000856C    pop     ecx
.text:1000856D    jnz     loc_100085F8
.text:10008573    push    offset a1       ; "1\n"
.text:10008578    push    hMSF5F5         ; File
.text:1000857E    call    edi ; fprintf
.text:10008580    mov     ebp, ds:LoadLibraryA
.text:10008586    pop     ecx
.text:10008587    pop     ecx
.text:10008588    push    esi             ; lpLibFileName
.text:10008589    call    ebp ; LoadLibraryA
.text:1000858B    test    eax, eax
.text:1000858D    mov     dword ptr unk_100B53E4, eax
.text:10008592    jnz     short loc_100085F8
.text:10008594    push    offset a2       ; "2\n"
.text:10008599    push    hMSF5F5         ; File
.text:1000859F    call    edi ; fprintf
.text:100085A1    push    ebx             ; Size
.text:100085A2    push    0               ; Val
.text:100085A4    push    esi             ; Dst
.text:100085A5    call    memset
.text:100085AA    push    offset aCProgramFilesA ; "C:\\Program Files\\ActivIdentity\\ActivCli"..
.text:100085AF    push    esi             ; Dest
.text:100085B0    call    strcpy
.text:100085B5    add     esp, 1Ch
.text:100085B8    push    esi             ; lpLibFileName
.text:100085B9    call    ebp ; LoadLibraryA
.text:100085BB    test    eax, eax
.text:100085BD    mov     dword ptr unk_100B53E4, eax
.text:100085C2    jnz     short loc_100085F8
.text:100085C4    push    offset key?     ; "3\n"
.text:100085C9    push    hMSF5F5         ; File
.text:100085CF    call    edi ; fprintf
.text:100085D1    push    ebx             ; Size
.text:100085D2    push    0               ; Val
.text:100085D4    push    esi             ; Dst
.text:100085D5    call    memset
.text:100085DA    push    offset aCProgramFilesX ; "C:\\Program Files(x86)\\ActivIdentity\\Act"..
```

Figure 20. Paths to Load ActivClient DLL

4.4. Proxy Selection

As depicted in Figure 21, it is interesting to see that this malware selects the proxy value depending on the application that it injects into.

Chong Rong Hwa, ronghwa.chong@gmail.com

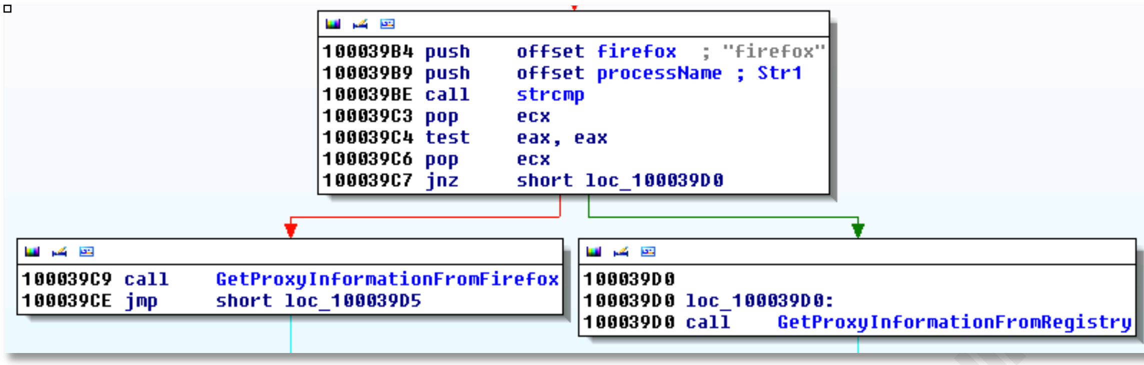


Figure 21. Proxy Selection

Suppose if it is a DLL loaded inside firefox, it will use the proxy setting found inside **"%APPDATA% \Mozilla\Firefox\Profiles\<profile folder>\prefs.js**" (see Figure 22). In other cases, proxy information is extracted from the registry "**HKEY_USERS\%SID%\Software\Microsoft\Windows\CurrentVersion\Internet Settings\Proxyserver**".

```
.text:100073BD                 push    CSIDL_APPDATA   ; csidl
.text:100073BF                 push    eax             ; pszPath
.text:100073C0                 push    0               ; hwnd
.text:100073C2                 call    ds:SHGetSpecialFolderPathA
.text:100073C8                 lea     eax, [ebp+Appdata_Mozilla_Firefox_Profiles]
.text:100073CE                 push    offset aMozillaFirefox ; "\\Mozilla\\Firefox\\Profiles"
.text:100073D3                 push    eax             ; Dest
.text:100073D4                 call    strcat
.text:100073D9                 lea     eax, [ebp+Appdata_Mozilla_Firefox_Profiles]
.text:100073DF                 push    offset aPrefs_js ; "prefs.js"
```

Figure 22. Retrieve Firefox Settings

Furthermore, it also noticed that Sykipot connects over port **80** or **443** (see Figure 23). These ports are probably chosen to increase the chance of connecting to the CnC server, as ports 80 or 433 are commonly used for HTTP and HTTPS web traffics respectively (Service Name and Transport Protocol Port Number Registry, 2012).

```
if ( !InternetCrackUrlA(sURL, 0, 0, &UrlComponents) )
  return 3;
strcpy(szObjectName, &v24[1]);
strcat(szObjectName, v25);
v4 = stricmp(Str1, "https");
hConnect = InternetConnectA(hInternet, szServerName, v4 != 0 ? 80 : 443, c, c, 3u, 0, 0);
```

Figure 23. Connection over HTTP or HTTPS

4.5. Encryption Mechanism Overview

The figure below depicts the usage of the wrapped encryption and decryption functions. For example, the pseudo code on the left reveals that the EncryptFile function is invoked to encrypt the data in "**MSF5F7.dat**" (plain text) and save the result to "**MSF5F6.dat**" (cipher) using a preprocessed key (string value "**19990817**"). This preprocessed key is further encoded before use in its encryption core (see Figure 25).

Use of Encryption

```
lea     eax, [ebp+fileDestination_MSF5F6.dat]
lea     ecx, [ebp+var_2C]
push    eax             ; fileDestination
lea     eax, [ebp+fileSource_MSF5F7.dat]
push    offset a19990817_key ; "19990817"
push    eax             ; fileSource
call    EncyptFile
```

Use of Decryption

```
lea     eax, [ebp+Path_MSF5F6.dat]
push    offset aMsf5f6_dat ; "MSF5F6.dat"
push    eax             ; Dest
call    strcat
lea     eax, [ebp+Path_MSF5F7.dat]
push    offset aMsf5f7_dat ; "MSF5F7.dat"
push    eax             ; Dest
call    strcat
add     esp, 18h
lea     ecx, [ebp+var_14]
call    sub_10001000
lea     eax, [ebp+Path_MSF5F7.dat]
xor     esi, esi
push    eax             ; MSF5F7.dat - destination
lea     eax, [ebp+Path_MSF5F6.dat]
push    offset a19990817_key ; "19990817"
push    eax             ; MSF5F6.DAT - source
lea     ecx, [ebp+var_14]
mov     [ebp+var_4], esi
call    DecryptFile
```

Figure 24. Usage of Encryption and Decryption Functions

Figure 25 depicts the flow and pseudo code of how Sykipot encrypts or decrypts a data block (64 bits) using a key (64 bits). As seen in its pseudo code, the 64 bits input data is represented using two separate DWORD variables. E.g. dataInDWHigh and dataInDWLow are DWORD variables, which store higher and lower order DWORD values of the input data respectively.

Additionally, the pseudo code also reveals that the data is encoded, before and after use of the custom DES function, using two different functions. With these additional layers of encoding, it further complicates the analysis of Sykipot encryption function.

The analysis of the encoder and custom DES functions are further detailed in section 4.5.1 and 4.5.2 respectively.

Chong Rong Hwa, ronghwa.chong@gmail.com

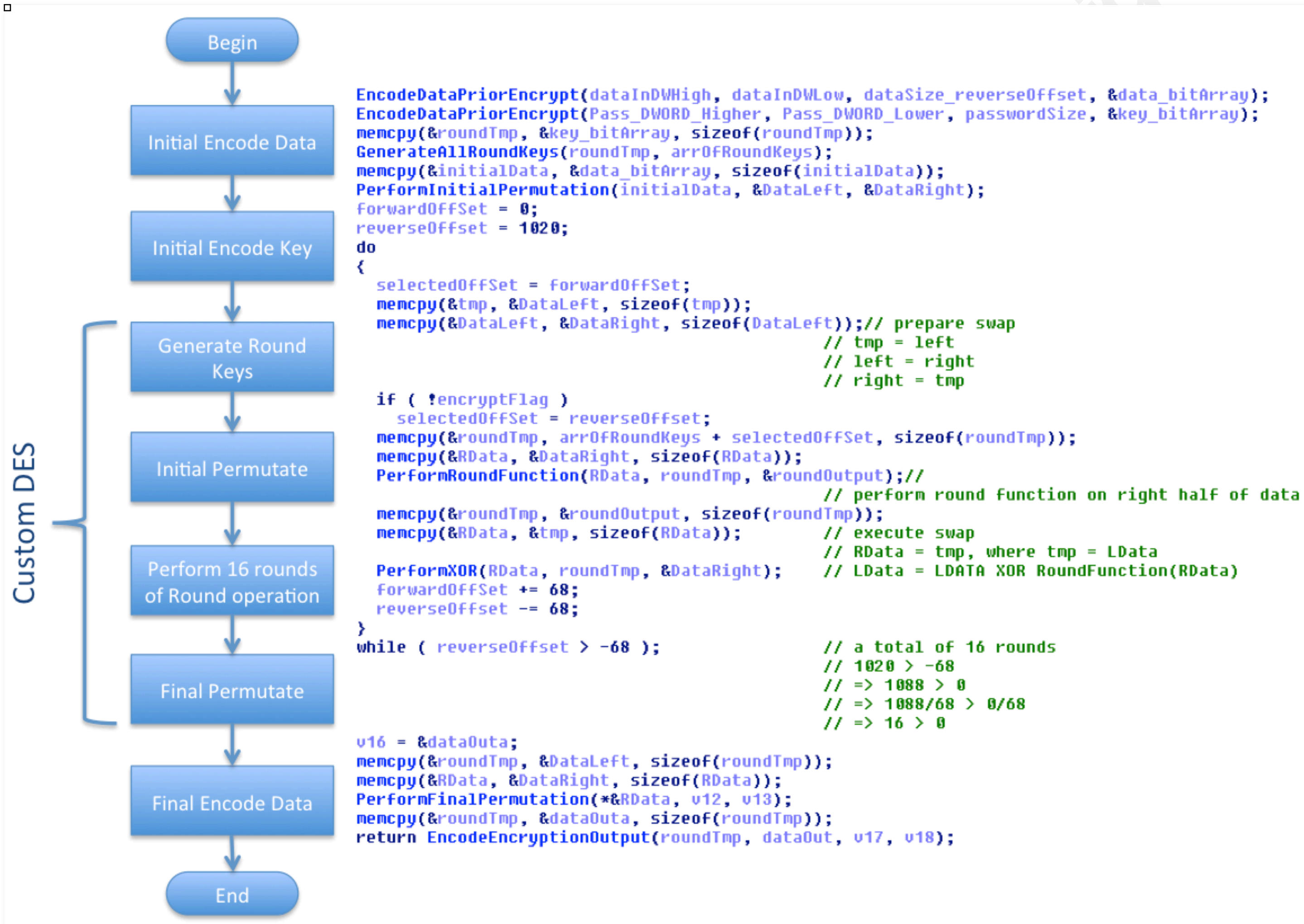


Figure 25. Flow of Encrypting/Decrypting a Block of Data

4.5.1. Encoder Functions

The Initial Encode Function (IEF), as shown in Figure 26, reveals that each byte of the data is first added with an encoding key (integer value **28**), and then converted into an array of bit values. As seen in Figure 25, this function is used to encode both input data and key prior use of custom DES.

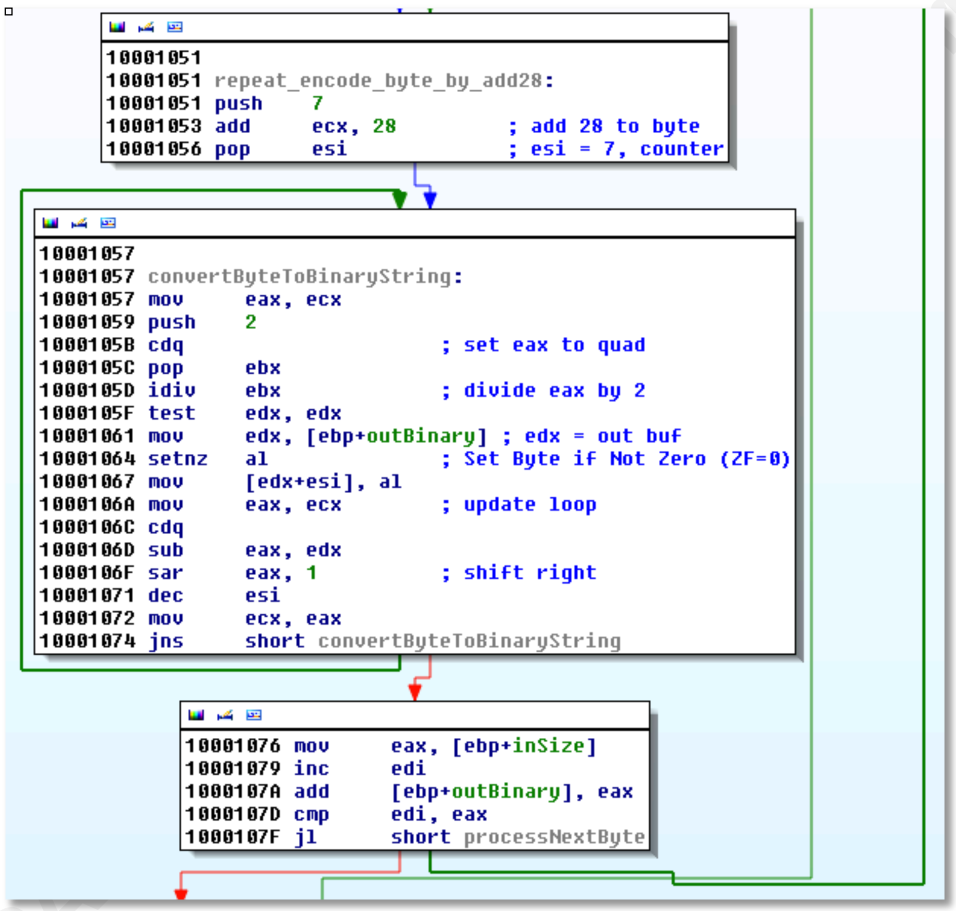


Figure 26. Initial Encode Function

A majority of the binary data used within the Sykipot Encryption/Decryption functions are stored using the data structure described in Figure 27, where bits and size are fields of type BYTE [64] and DWORD respectively. The bits field is used to store data binary manipulation, while the size field describes the number of bits stored.

```
00000000 Data_64Bits      struc ; (sizeof=0x44)
00000000 bits             db 64 dup(?)
00000040 size             dd ?
00000044 Data_64Bits      ends
```

Figure 27. Data Structure Used to Store Binary Values

The Final Encode Function (FEF) shown in Figure 28 reveals that a binary array is converted into a byte value, and then subtracts the byte value with an encoding key (integer value **28**). As described in Figure 25, this function is used to encode data after encrypting the data using the custom DES function.

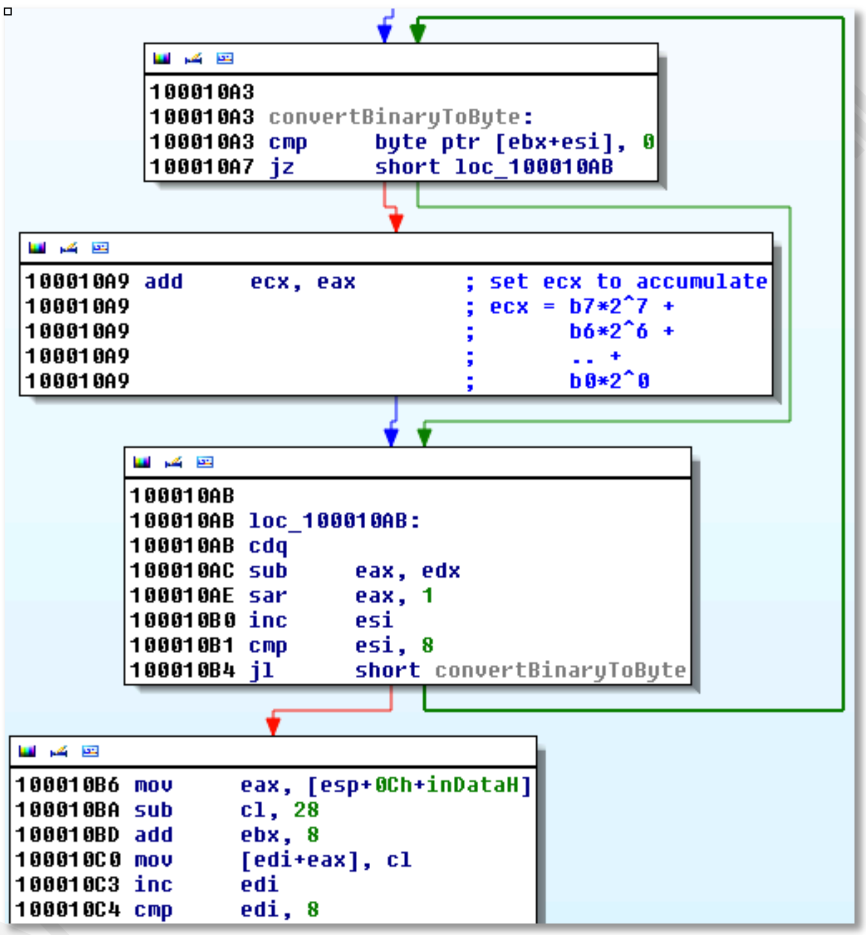


Figure 28. Final Encode Function

From the implementation of IEF and FEF, it shows that they are two simple inversely related functions, where IEF and FEF encode by addition and subtraction using the same encoding key respectively.

To generalize this analysis, Figure 29 mathematically proofs that if IEF and FEF are inversely related, Sykipot Decryption Function is then guaranteed to be able to decrypt the data encrypted using Sykipot Encryption Function. Hence, it implies that the malware author could possibly further complicate the analysis by implementing a more complex IEF, as long as IEF and FEF are inversely related.

Chong Rong Hwa, ronghwa.chong@gmail.com

Let S^E be the Sykipot encryption function,
S^D be the Sykipot decryption function,
A be the Sykipot initial encoding function,
B be the Sykipot final encoding function,
DES_k be the DES encryption function,
DES_k^{-1} be the DES decryption function,
k be an arbitrary key used by the DES encryption and decryption function, and
P be an arbitrary plain text,
where $S^E = B \circ DES_k \circ A$ and
$S^D = B \circ DES_k^{-1} \circ A$.

Suppose if function A and B are inversely related, then

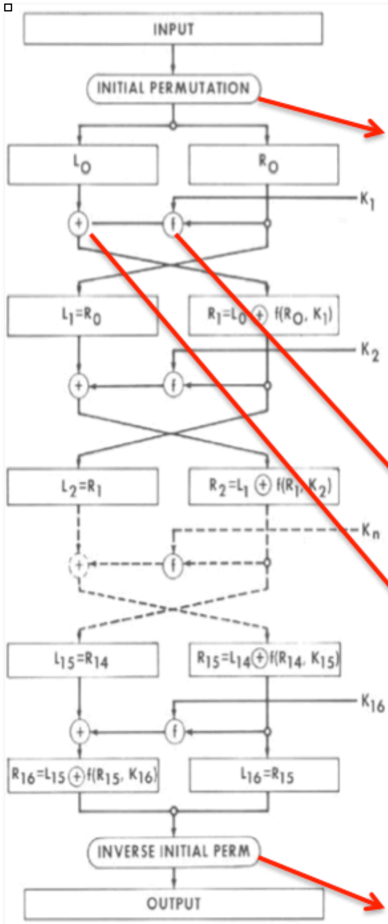
$S^D \circ S^E (\mathrm{P}) = B \circ DES_k^{-1} \circ A \circ B \circ DES_k \circ A (\mathrm{P})$
$= B \circ DES_k^{-1} \circ (A \circ B) \circ DES_k \circ A (\mathrm{P})$ (since composite function is associative)
$= B \circ (DES_k^{-1} \circ DES_k) \circ A (\mathrm{P})$ (since A is an inverse function of B)
$= B \circ A (\mathrm{P})$ (since DES_k^{-1} is an inverse function of DES_k)
$= \mathrm{P}$ (since A is an inverse function of B)

Hence, S^D is an inverse function of S^E.
Q.E.D.

Figure 29. Proof of Sykipot Decryption Function

4.5.2. Custom DES Function

From the pseudo code in Figure 30, it is obvious that the Sykipot encryption function has sub functions that match the flow of DES Feistel Structure to perform Initial Permutation, Round Manipulation, XOR and Final Permutation.




```
GenerateAllRoundKeys(roundTmp, arrOfRoundKeys);
memcpy(&initialData, &data_bitArray, sizeof(initialData));
PerformInitialPermutation(initialData, &DataLeft, &DataRight);
forwardOffSet = 0;
reverseOffset = 1020;
do
{
  selectedOffSet = forwardOffSet;
  memcpy(&tmp, &DataLeft, sizeof(tmp));
  memcpy(&DataLeft, &DataRight, sizeof(DataLeft));// prepare swap
                                               // tmp = left
                                               // left = right
                                               // right = tmp
  if ( !encryptFlag )
    selectedOffSet = reverseOffset;
  memcpy(&roundTmp, arrOfRoundKeys + selectedOffSet, sizeof(roundTmp));
  memcpy(&RData, &DataRight, sizeof(RData));
  PerformRoundFunction(RData, roundTmp, &roundOutput);//
                                               // perform round function on right half of data
  memcpy(&roundTmp, &roundOutput, sizeof(roundTmp));
  memcpy(&RData, &tmp, sizeof(RData));         // execute swap
                                               // RData = tmp, where tmp = LData
  PerformXOR(RData, roundTmp, &DataRight);     // LData = LDATA XOR RoundFunction(RData)
  forwardOffSet += 68;
  reverseOffset -= 68;
}
while ( reverseOffset > -68 );                 // a total of 16 rounds
                                               // 1020 > -68
                                               // => 1088 > 0
                                               // => 1088/68 > 0/68
                                               // => 16 > 0
v16 = &dataOuta;
memcpy(&roundTmp, &DataLeft, sizeof(roundTmp));
memcpy(&RData, &DataRight, sizeof(RData));
PerformFinalPermutation(*&RData, v12, v13);
```

Source of DES Feistel Structure Flow: (Daley & Kammer, 1999)

Figure 30. Mapping DES Feistel Structure to Skyipot

All permutations that are used by the custom DES encryption/decryption function are performed using the generic permutation function identified in Figure 31. The parameter "option" is used to select the type of permutation to perform. The options supported by this function are tabularized in Table 4.

```
// option 1: 64 bits (IP)
// option 2: 64 bits (FP)
// option 3: 48 bits (Expansion Table)
// option 4: 32 bits (P)
// option 5: 56 bits (PC1_C)
// option 6: 48 bits (PC2_C)
int __stdcall PerformPermutationByOption
        (DWORD option, Data_64Bits dataIn,
                       Data_64Bits *dataOut)
```

```
beginPermutate:
  pDataOut = dataOut;
  idx = 0;
  dataOut->size = cntMax;
  if ( cntMax > 0 )
  {
    while ( 1 )
    {
      if ( option == 1 )
      {
        constantDword = InitialPermutation[idx];
        goto updateConstantBox;
      }
      if ( option == 2 )
      {
        constantDword = FinalPermutation[idx];
        goto updateConstantBox;
      }
      if ( option == 3 )
      {
        constantDword = ExpansionTable[idx];
        goto updateConstantBox;
      }
      if ( option == 4 )
      {
        constantDword = PermutationTable[idx];
        goto updateConstantBox;
      }
      if ( option == 5 )
        break;                                  // PC1_C
      if ( option == 6 )
      {
        constantDword = PC2_C[idx];
updateConstantBox:
        pDataOut->bits[idx] = *(&option + constantDword + 3);
      }
      ++idx;
      if ( idx >= cntMax )
        return idx;
    }                                           // end while
    constantDword = PC1_C[idx];
    goto updateConstantBox;
  }
  return idx;
}
```

Figure 31. Perform Permutation By the parameter "Option"

Option	Permutation Type	Output (Number of bits)	Description
1	Initial Permutation	64	Permutates the data input prior passing through Feistel structure.
2	Final Permutation	64	Permutates the data output after passing through Feistel structure.
3	E	48	Permutates and Expands data used in round function. This E table is customized (see below for details).
4	P	48	Permutates data used in round function.
5	PC1	56	Permutates key before scheduling.
6	PC2	48	Generates round key.

Table 4. Options Supported By Generic Permutation Function


```
.data:1000B020 InitialPermutation dd 58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20
.data:1000B020                                        ; DATA XREF: PerformPermutationByOpt
.data:1000B020                   dd 12, 4, 62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40
.data:1000B020                   dd 32, 24, 16, 8, 57, 49, 41, 33, 25, 17, 9, 1, 59, 51
.data:1000B020                   dd 43, 35, 27, 19, 11, 3, 61, 53, 45, 37, 29, 21, 13, 5
.data:1000B020                   dd 63, 55, 47, 39, 31, 23, 15, 7
.data:1000B120 FinalPermutation dd 40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47, 15, 55, 23
.data:1000B120                                        ; DATA XREF: PerformPermutationByOpt
.data:1000B120                   dd 63, 31, 38, 6, 46, 14, 54, 22, 62, 30, 37, 5, 45, 13
.data:1000B120                   dd 53, 21, 61, 29, 36, 4, 44, 12, 52, 20, 60, 28, 35, 3
.data:1000B120                   dd 43, 11, 51, 19, 59, 27, 34, 2, 42, 10, 50, 18, 58, 26
.data:1000B120                   dd 33, 1, 41, 9, 49, 17, 57, 25
.data:1000B220 ExpansionTable    dd 32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8, 9, 10, 11, 12
.data:1000B220                                        ; DATA XREF: PerformPermutationByOpt
.data:1000B220                   dd 13, 12, 13, 14, 15, 16, 17, 16, 17, 18, 29, 20, 21
.data:1000B220                   dd 20, 21, 22, 23, 24, 25, 24, 25, 26, 27, 28, 29, 28
.data:1000B220                   dd 29, 30, 31, 32, 1
.data:1000B2E0 ; int SBoxValues[8][64]
.data:1000B2E0 SBoxValues        dd 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7
.data:1000B2E0                                        ; DATA XREF: PerformSboxing+3F↑r
.data:1000B2E0                   dd 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8
.data:1000B2E0                   dd 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0
.data:1000B2E0                   dd 15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13
.data:1000B2E0                   dd 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10
.data:1000B2E0                   dd 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5
.data:1000B2E0                   dd 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15
.data:1000B2E0                   dd 13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9
.data:1000B2E0                   dd 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8
.data:1000B2E0                   dd 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1
.data:1000B2E0                   dd 13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7
.data:1000B2E0                   dd 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12
.data:1000B2E0                   dd 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15
.data:1000B2E0                   dd 13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9
.data:1000B2E0                   dd 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4
.data:1000B2E0                   dd 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14
.data:1000B2E0                   dd 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9
.data:1000B2E0                   dd 14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6
.data:1000B2E0                   dd 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14
.data:1000B2E0                   dd 11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3
.data:1000B2E0                   dd 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11
.data:1000B2E0                   dd 10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8
.data:1000B2E0                   dd 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6
.data:1000B2E0                   dd 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13
.data:1000B2E0                   dd 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1
.data:1000B2E0                   dd 13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6
.data:1000B2E0                   dd 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2
```

Figure 32. Customised E Table

All the values that are used by the permutation and substitution tables are the same as the constants used in DES implementation (Daley & Kammer, 1999), except for one element in the **E Table** is changed from **19** to **29** (see Figure 32 for the number circled in red). By definition of Feistel Cipher (Backes, 2007), there is no requirement for the round function to be invertible. Hence, by changing the constants (such as E Table constants) used by the round function, does not affect the decryption of the encrypted cipher, as long as the round function implemented in both encryption and decryption algorithms are consistent.

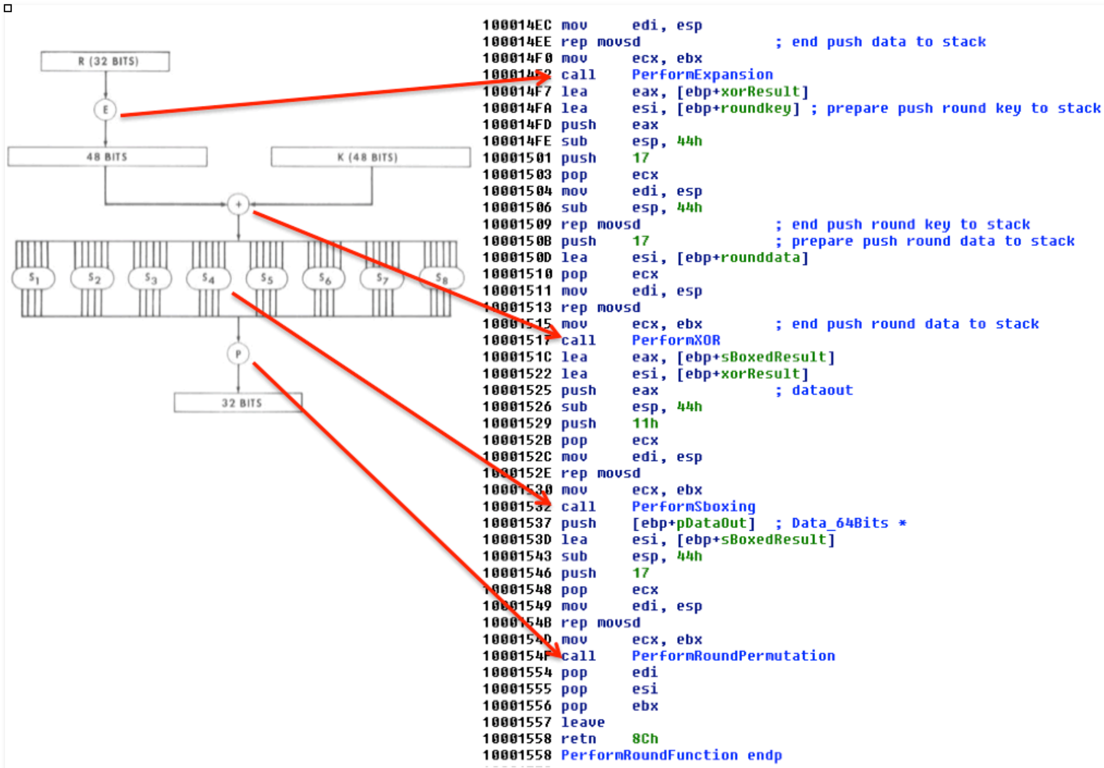
It is believed that the malware author has changed only one DES standard constant to trick the analyst into thinking that the standard DES encryption algorithm is used. It is hard to detect this minor change with the consideration that there are more than 3000 constants used in standard DES implementation.

Figure 33 reveals that the Round Key Generation function implemented by Sykipot has sub functions that match the DES Round Key Generation Flow, i.e. functions to rotate the round key seed and generate round key. Similarly, Figure 34 shows that the Round Function implemented by Sykipot also has sub functions that match the Round Function Flow, i.e. functions to expand and permutate the data, XOR the expanded data with the round key, substitute the data using the SBoxes and permutate using the round permutation table.

All these evidences suggest that the Sykipot encryption algorithm is implemented using custom DES (using modified E Table) with input data, input key and output data encoded to confuse the analyst. With this knowledge, researcher could possibly design a fake bot to interact with the attacker, to further analyze Sykipot.

Source of DES Round Key Generation Flow: (Daley & Kammer, 1999)

Figure 33. Mapping DES Round Key Generation to Skyipot



Source of Round Function Flow: (Daley & Kammer, 1999)

Figure 34. Mapping DES Round Function to Skyipot

4.5.3. Encryption Analysis Validation

After analyzing the encryption function, the next step is to validate the analysis. Below lists the steps (of one possible way) to validate the analysis of the Sykipot encryption function:

1. Generate a plaintext file with arbitrary content.
2. Encrypt the plaintext file using unpatched Sykipot (see Figure 35).
3. Encrypt the plaintext file using patched Sykipot (see Figure 36), where the patches are applied to convert Sykipot Custom DES into Standard DES.
4. Encrypt the plaintext file using standard DES (see Figure 37).
5. Compare each cipher generated by Sykipot (patched and unpatched) against the cipher generated by standard DES (see Figure 37).

Suppose if the analysis is correct, the cipher generated by the patched Sykipot should be the same as the cipher generated by the standard DES; and the cipher generated by the unpatched Sykipot should be no way close to the cipher generated by the standard DES.

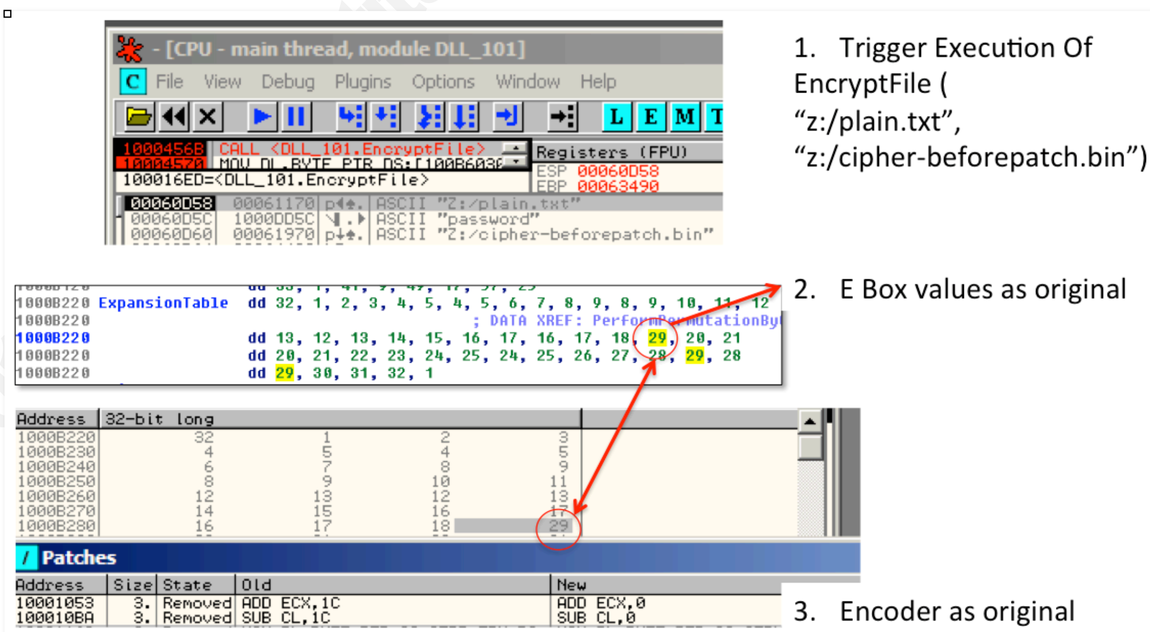


Figure 35. Generate Cipher Using Unpatched Encryption Function

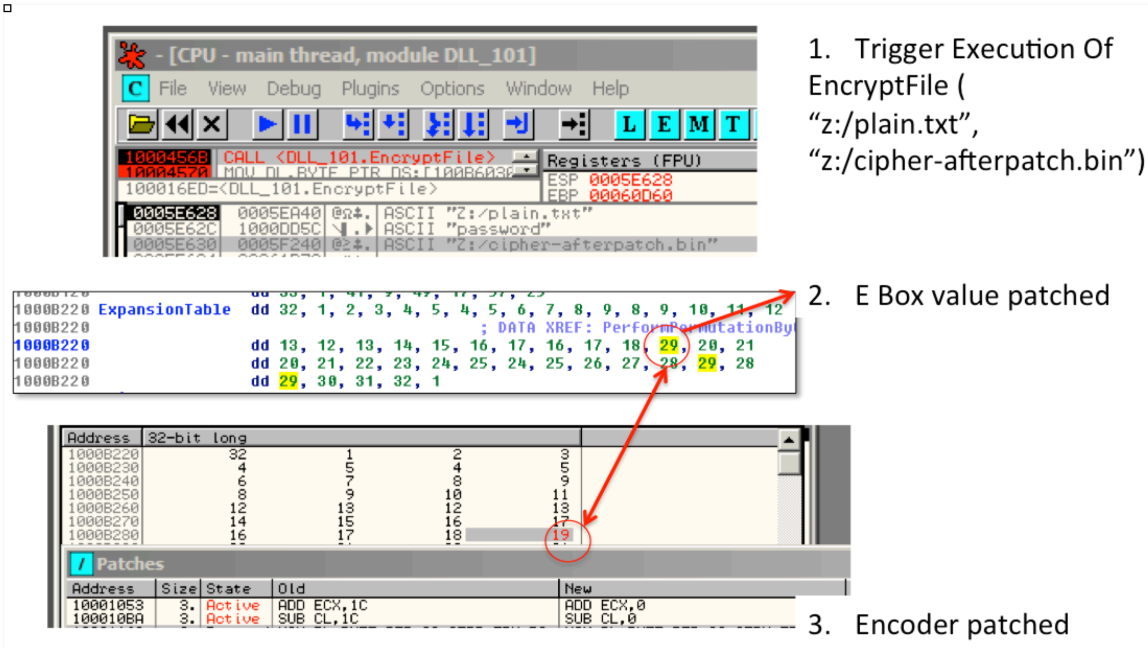


Figure 36. Generate Cipher Using Patched Encryption Function

Figure 37 and Figure 38 depict the comparison between Sykipot generated ciphers and OpenSSL generated ciphers, where OpenSSL is a tool that could be used to generate standard DES cipher (OpenSSL for Windows, 2008). As expected, the cipher generated prior patching is no way close to the cipher generated using standard DES (see Figure 37). This shows that Sykipot is not encrypting using DES. However, it is surprising to note that the last eight bytes between the ciphers generated by the patched Sykipot and OpenSSL are different (see Figure 38). This shows that the patched Sykipot generates the same cipher as DES, except for the last block (64 bits).

Chong Rong Hwa, ronghwa.chong@gmail.com

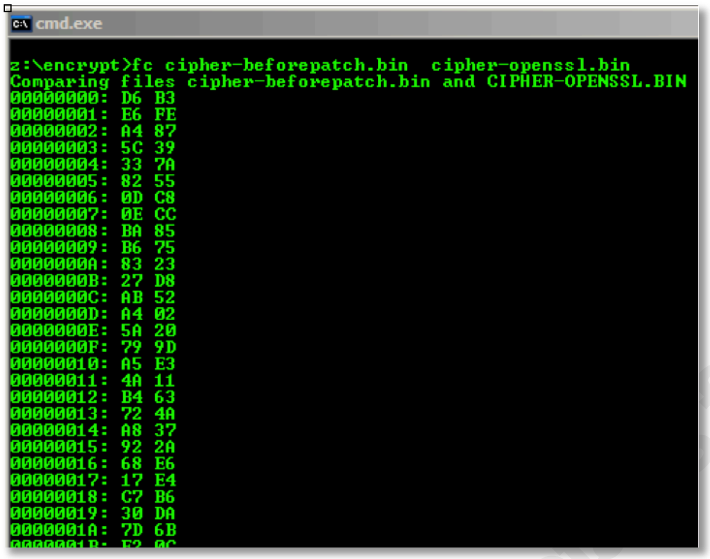


Figure 37. Comparing Sykipot Cipher with DES Cipher

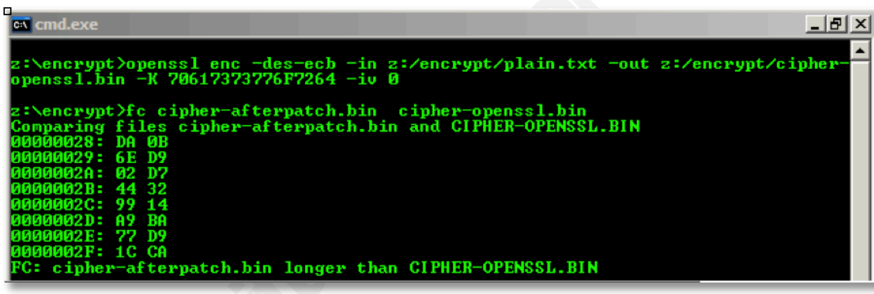


Figure 38. Comparing Sykipot (After Patch) Cipher with DES Cipher

To investigate this difference, the code is examined deeper. As shown in Figure 39, the pseudo code implies that the plain text is padded with 0x20 to a file size divisible by 8 bytes (since the block size is 64 bits).

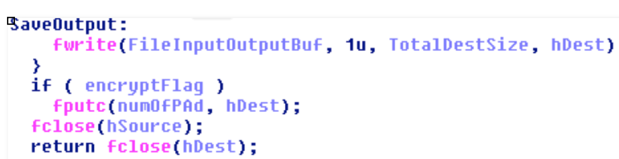


Figure 39. Code to Pad Plain Text

Additionally, it is also observed that a one-byte pad information is appended to the end of cipher to indicate the number of pad used (see Figure 40).

Chong Rong Hwa, ronghwa.chong@gmail.com

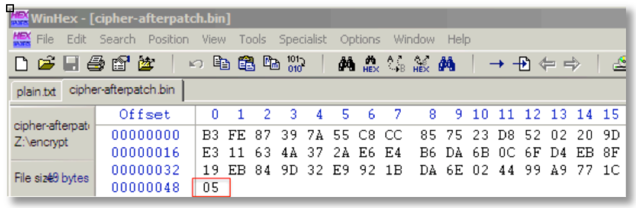


Figure 40. Pad Information

To verify the abovementioned analysis, the plain text is padded with pad (0x20) to shortest possible file length divisible by 8. In this case, 5 bytes of pads are applied to the plain text (see in Figure 41).

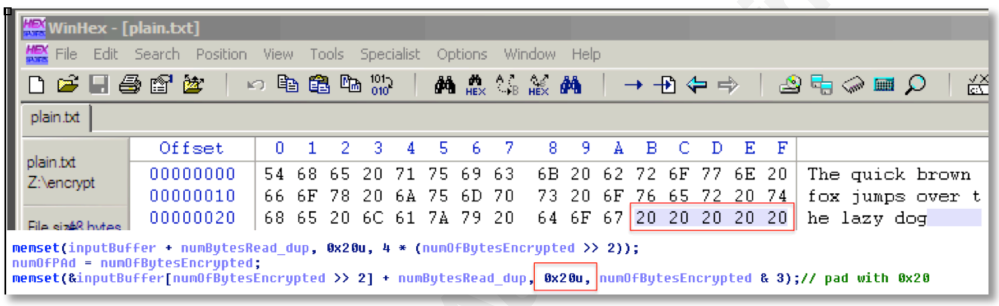


Figure 41. Padded Plaintext

After a retest, it is verified that there is no discrepancies between the ciphers generated by patched Sykipot and OpenSSL, other than the additional padding information added by Sykipot (see Figure 42).

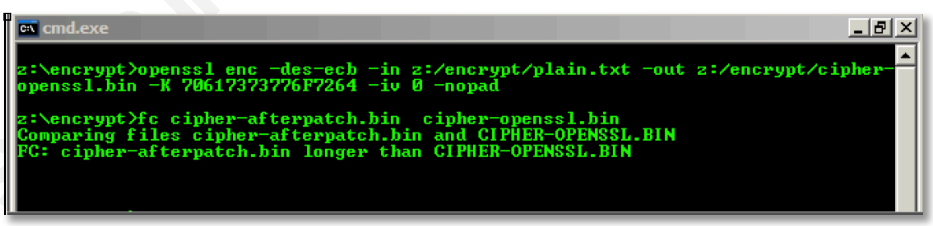


Figure 42. Comparing Sykipot (After Patch) Cipher with DES Cipher

Chong Rong Hwa, ronghwa.chong@gmail.com

5. Remediation Measures

Infection caused by this Sykipot sample can be easily remediated with the following steps:

1. Close all targeted processes (i.e. Internet Explorer, Firefox and Outlook) to unload malicious DLL.
2. Kill "dmm.exe". One possible way is to use Process Explorer (see Figure 43).
3. Remove all malicious artifacts (files with name starting with "MSF5F" and "dmm.exe") found in the Sykipot's working directory.
4. Remove "taskmost.exe" from the start up folder if it exists. See Figure 44 to open start up folder using "shell:startup" command.

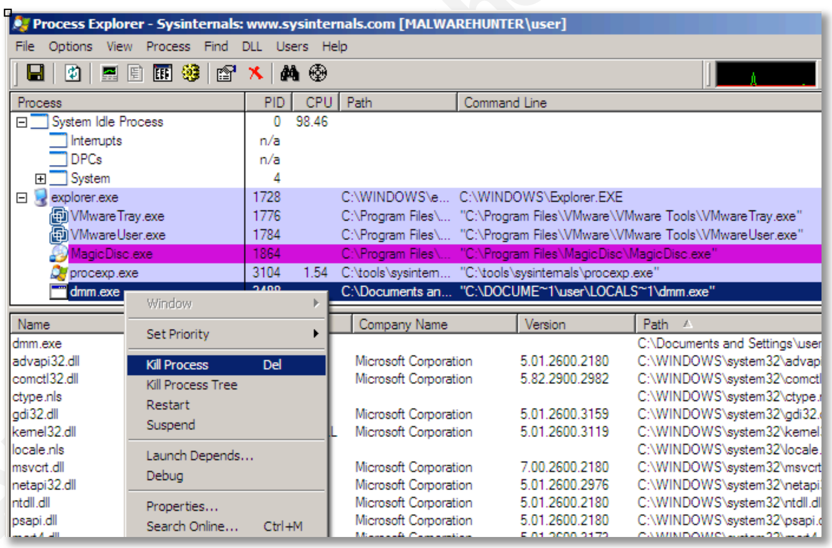


Figure 43. Killing of Sykipot in Process Explorer

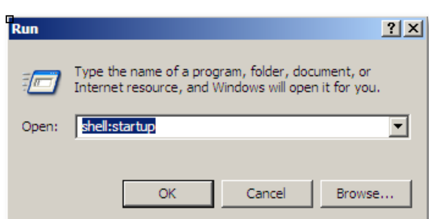


Figure 44. Open Start-up Folder

Chong Rong Hwa, ronghwa.chong@gmail.com

6. Conclusion

From the analysis in this paper, it is obvious that Sykipot is an espionage malware designed to steal victim's information, access protected resources and maintain backdoor in a persistent and stealthy manner. By understanding the techniques used by Sykipot, it helps the analysts to take note of the tricks that Sykipots has used to avoid detection.

Unlike a majority of malwares that dial back to CnC server at periodic interval, Sykipot is able to connect to the CnC at a time specified by the attacker. By having an indeterministic dial back time, it is hard to notice Sykipot's connection as a network anomaly. Additionally, its connection is unlikely to be blocked by firewall as it is connected out over port 80 or 443, via the injected processes that are expected to have HTTP or HTTPS connections (see Figure 23 and Figure 3). Hence, it is dangerous for an analyst to assume that a system is clean, even if there is no network connection performed at a regular time interval.

Additionally, an analyst should not assume executable files that have timestamp or version information that appears to be a Microsoft system file to be safe (see Figure 8 and Figure 6). Instead, the analyst should also consider the path of the executable files when performing forensic. In this case, it is suspicious for a Microsoft system file to be located in local settings, and therefore this anomaly should be flagged.

On top of that, by injecting Sykipot DLL using CreateRemoteThread with LoadLibrary technique, Sykipot would not be flagged as malicious by Volatility malfind plugin. This effectively helps Sykipot to camouflage itself as a benign DLL. Consequently, an analyst should not be overly reliant on automated scripts to identify anomalies.

Last but not least, in the event if a new Sykipot is identified, an analyst could possibly try to use the analyzed encryption algorithm to decrypt Sykipot related messages to further understand intent of the malware.

7. References

Kuster, R. (2003, August 20). Three Ways to Inject Your Code into Another Process. Retrieved from http://www.codeproject.com/Articles/4610/Three-Ways-to-Inject-Your-Code-into-Another-Proces

Blasco, J. (2012, January 12). Sykipot variant hijacks DOD and Windows smart cards. (AlienVault) Retrieved from http://labs.alienvault.com/labs/index.php/2012/when-the-apt-owns-your-smart-cards-and-certs/

Thakur, V. (2011, December 14). The Sykipot Attacks. (Symantec) Retrieved from http://www.symantec.com/connect/blogs/sykipot-attacks

Volatilty Command Reference. (2012, March 1). Retrieved from http://code.google.com/p/volatility/wiki/CommandReference

OpenSSL for Windows. (2008, December 4). Retrieved from http://gnuwin32.sourceforge.net/packages/openssl.htm

Daley, W. M., & Kammer, R. G. (1999, October 25). DATA ENCRYPTION STANDARD (DES). (U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology) Retrieved from http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

Visual Resource Editor. (2012, March 25). (Heaventools) Retrieved from http://www.heaventools.com/resource-tuner.htm

Service Name and Transport Protocol Port Number Registry. (2012, March 28). (Internet Assigned Numbers Authority) Retrieved from http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt

Backes, M. (2007). Block Ciphers. (Saarland University) Retrieved from http://web.cs.du.edu/~ramki/courses/security/2011Winter/notes/feistelProof.pdf

Chong Rong Hwa, ronghwa.chong@gmail.com

Last Updated: April 8th, 2014

Upcoming Training

SANS Security West 2014	**San Diego, CA**	**May 08, 2014 - May 17, 2014**	**Live Event**
Mentor Session - FOR 610	**Columbia, MD**	**May 21, 2014 - Jul 23, 2014**	**Mentor**
Digital Forensics & Incident Response Summit	**Austin, TX**	**Jun 03, 2014 - Jun 10, 2014**	**Live Event**
Community SANS Ottawa	**Ottawa, ON**	**Jun 16, 2014 - Jun 21, 2014**	**Community SANS**
SANSFIRE 2014	**Baltimore, MD**	**Jun 21, 2014 - Jun 30, 2014**	**Live Event**
SANS vLive - FOR610: Reverse-Engineering Malware: Malware Analysis Tools and Techniques	**FOR610 - 201407,**	**Jul 14, 2014 - Aug 20, 2014**	**vLive**
SANS Virginia Beach 2014	**Virginia Beach, VA**	**Aug 18, 2014 - Aug 29, 2014**	**Live Event**
SANS Baltimore 2014	**Baltimore, MD**	**Sep 22, 2014 - Sep 27, 2014**	**Live Event**
SANS DFIR Prague 2014	**Prague, Czech Republic**	**Sep 29, 2014 - Oct 11, 2014**	**Live Event**
SANS vLive - FOR610: Reverse-Engineering Malware: Malware Analysis Tools and Techniques	**FOR610 - 201410,**	**Oct 13, 2014 - Nov 19, 2014**	**vLive**
Community SANS Paris @ HSC - FOR610 (in French)	**Paris, France**	**Nov 24, 2014 - Nov 28, 2014**	**Community SANS**
SANS OnDemand	**Online**	**Anytime**	**Self Paced**
SANS SelfStudy	**Books & MP3s Only**	**Anytime**	**Self Paced**