Detailed Analysis Of Sykipot (Smartcard Proxy Variant)

Rong Hwa Chong
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.

Chong Rong Hwa, ronghwa.chong@gmail.com
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.

![Flow of Sykipot EXE](image)

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).

![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).

![Figure 4. DLL Injection Using CreateRemoteThread with LoadLibraryA](image_url)

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).

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 Volatility dlllist plugin (memory forensic tool).

**Figure 5. Statically Examine Sykipot EXE using PEview**

**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. (Volatility Command Reference, 2012). Consequently, Sykipot achieves stealth by not hiding itself.

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.
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.

Figure 8. Time Stomping of Sykipot EXE
Detailed Analysis Of Sykipot (Smartcard Proxy Variant)

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).

```c
GetModuleFileNameA(0, &ExistingFileName, 0x100u);
SHGetSpecialFolderPathA(0, &StartupFolder, CSLD_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 12u: // WM_QUIT
        CopyFileA(&ExistingFileName, &StartupFolder, 0);
        if ( TokenHandle )
        {
            CloseHandle(tokenHandle);
            RevertToSelf();
        }
        exit(0);
        return result;

case 11u: // WM_QUERYENDSESSION
        CopyFileA(&ExistingFileName, &StartupFolder, 0);
        if ( TokenHandle )
        {
            CloseHandle(tokenHandle);
            RevertToSelf();
        }
        exit(0);
        return result;

case 16u: // 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 depict 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.

![Flow Key Logger Thread](image-url)

Figure 10. Flow Key Logger Thread

Chong Rong Hwa, ronghwa.chong@gmail.com
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

Chong Rong Hwa, ronghwa.chong@gmail.com
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 Type of Command Categories](image)

**Figure 13. Data Type of Command Categories**
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

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.

<table>
<thead>
<tr>
<th>File name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gtpretty.tmp</td>
<td>Orders from the CnC.</td>
</tr>
<tr>
<td>Gdtpretty.tmp</td>
<td>Decrypted version of orders from the CnC.</td>
</tr>
<tr>
<td>Pdtpretty.tmp</td>
<td>Log file.</td>
</tr>
<tr>
<td>Ptpretty.tmp</td>
<td>Encrypted version of log file.</td>
</tr>
</tbody>
</table>

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).

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

Table 2. Backdoor Command Comparison

Chong Rong Hwa, ronghwa.chong@gmail.com
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.

```c
hFileDelete = CreateFileA(sFileDelete, 0, 0, 0, 3u, 0, 0);
FileDelete_size = GetFileSize(hFileDelete, 0);
CloseHandle(hFileDelete);
void *fp = fopen(sFileDelete, "w");
HDFS = fopen(FileNameFromRecon, "a");

if ( fp == NULL )
    return;

for ( i = 0; *(u32) + _flag & 0x10 & i < (signed int)FileDelete_size; ++i )
{
    fprintf(fp, 0);
    // zeroise the file
    u31 = HDFS;
}
fclose(fp);
HDFS = fopen(FileNameFromRecon, "a");
if ( fp == NULL )
    return;
else
    deleteStatus = "file not found!";
printf(HDFS, deleteStatus);
fclose(HDFS);
```

Figure 16. Secure File Deletion

### 4.3.2. Smartcard Specific Backdoor Commands

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

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

Table 3. Smartcard Specific Backdoor Commands

Chong Rong Hwa, ronghwachong@gmail.com
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.

![Figure 17. Open System Store](image-url)
Detailed Analysis Of Sykipot (Smartcard Proxy Variant)

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 18. Retrieve Certificate Information

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.

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.
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”.

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).

Chong Rong Hwa, ronghwa.chong@gmail.com
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).

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 dataInDWLlow 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
Detailed Analysis Of Sykipot (Smartcard Proxy Variant)

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

begin
  encodeDataPriorEncrypt(dataInDataHigh, dataInDataLow, dataSize_reverseOffset, &data_bitArray);
  encodeDataPriorEncrypt(Pass_DWORD_Higher, Pass_DWORD_Lower, passwordSize, &key_bitArray);
  memcpy(roundTmp, &key_bitArray, sizeof(roundTmp));
  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
    if (tencryptFlag) // left - right
    {
      memcpy(Rdata, &dataRight, sizeof(Rdata));
      PerformRoundFunction(Rdata, roundTmp, &roundOutput);
      memcpy(Rdata, &dataLeft, sizeof(Rdata)); // execute swap
      Perform XOR(Rdata, roundTmp, &dataRight);
    }
    forwardOffset = 68;
    reverseOffset = -68;
  } while (reverseOffset < -68); // a total of 16 rounds
  // 1020 > -68
  // => 1088 > 0
  // => 1088/68 > 0/68
  // => 16 > 0

  v16 = &dataOut;
  memcpy(roundTmp, &dataLeft, sizeof(roundTmp));
  memcpy(Rdata, &dataRight, sizeof(Rdata));
  PerformFinalPermutation(+Rdata, v12, v13);
  memcpy(Rdata, &dataOut, sizeof(Rdata));
  return EncodeEncryptionOutput(roundTmp, dataSet, v17, v18);
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.

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
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.
Detailed Analysis Of Sykipot (Smartcard Proxy Variant)

Figure 30. Mapping DES Feistel Structure to Sykipot
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.

### Figure 31. Perform Permutation By the parameter “Option”

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

Table 4. Options Supported By Generic Permutation Function
Detailed Analysis Of Sykipot (Smartcard Proxy Variant)

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

Figure 32. Customised E Table
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.

Chong Rong Hwa, ronghwa.chong@gmail.com
Detailed Analysis Of Sykipot (Smartcard Proxy Variant)

Figure 33. Mapping DES Round Key Generation to Skyipot

Source of DES Round Key Generation Flow: (Daley & Kammer, 1999)
Detailed Analysis Of Sykipot (Smartcard Proxy Variant)

Figure 34. Mapping DES Round Function to Skyipot

Source of Round Function Flow: (Daley & Kammer, 1999)
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
Detailed Analysis Of Sykipot (Smartcard Proxy Variant)

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
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).

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
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).

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).
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.

Chong Rong Hwa, ronghwa.chong@gmail.com
7. References


Chong Rong Hwa, ronghwa.chong@gmail.com
## Upcoming SANS Training

Click here to view a list of all SANS Courses

<table>
<thead>
<tr>
<th>Event Name</th>
<th>Location</th>
<th>Dates</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SANS Chicago Spring 2020</td>
<td>Chicago, ILUS</td>
<td>Jun 01, 2020 - Jun 06, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS ICS Europe Summit &amp; Training 2020</td>
<td>Munich, DE</td>
<td>Jun 08, 2020 - Jun 13, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Budapest June 2020</td>
<td>Budapest, HU</td>
<td>Jun 08, 2020 - Jun 13, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Las Vegas Summer 2020</td>
<td>Las Vegas, NVUS</td>
<td>Jun 08, 2020 - Jun 13, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Zurich June 2020</td>
<td>Zurich, CH</td>
<td>Jun 15, 2020 - Jun 20, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS London June 2020</td>
<td>OnlineGB</td>
<td>Jun 01, 2020 - Jun 06, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS OnDemand</td>
<td>Books &amp; MP3s OnlyUS</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
</tbody>
</table>

Last Updated: March 25th, 2020