### **Extracted — TryHackMe**

Scenario

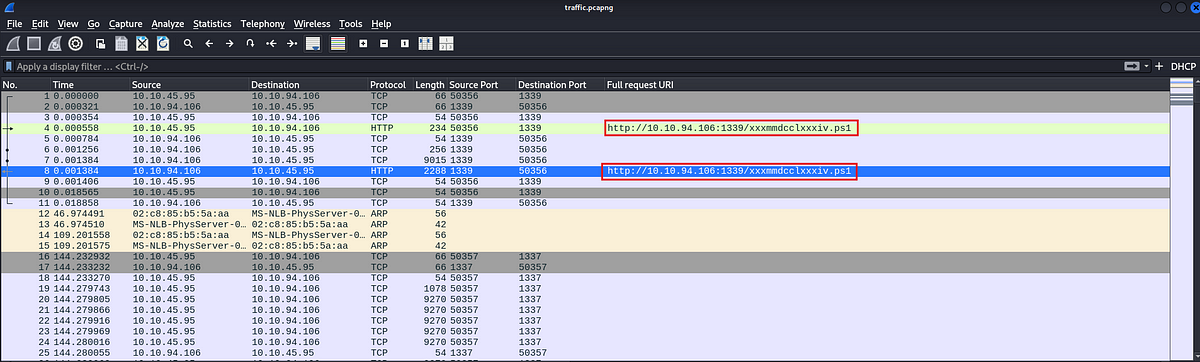
Working as a senior DFIR specialist brings a new surprise every day. Today, one of your junior colleagues raised an alarm that some suspicious traffic was generated from one of the workstations, but they couldn’t figure out what was happening.

Unfortunately, there was an issue with the SIEM ingesting the network traffic, but luckily, the network capture device was still working. They asked if you could look to find out what happened since you are known as The Magician around these parts.

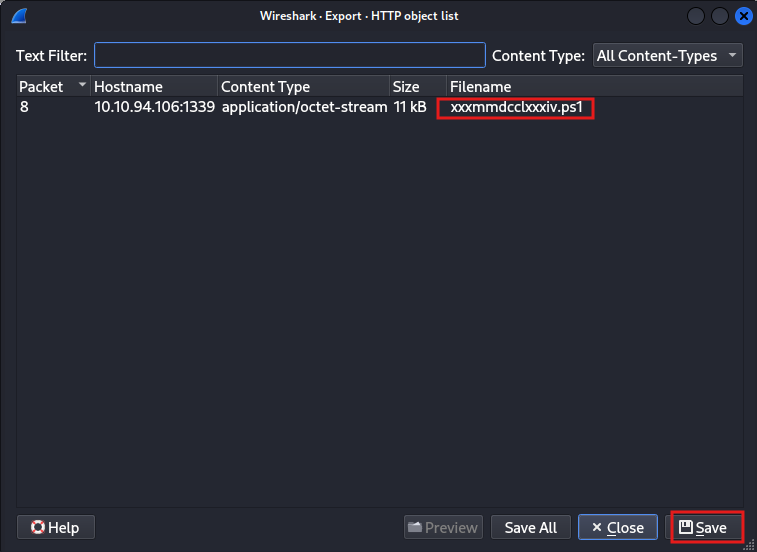
To start investigation, download the task files and extract them.

Analysis:

Immediately after opening the pcap file in wireshark, we can see that a HTTP Get request was made which requested a poweshell script, and the Powershell script was downloaded with HTTP 200 response. The name of the Powershell script was xxxmmdcclxxxiv.ps1.



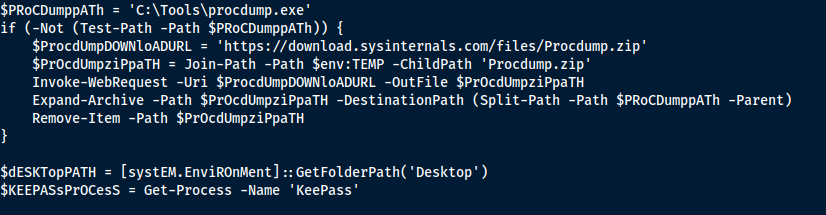
Let us extract this file, and check what exactly does this PowerShell script do. To extract the file, we can go to File -> Export Objects -> HTTP… Then select the file and click save to extract the file.



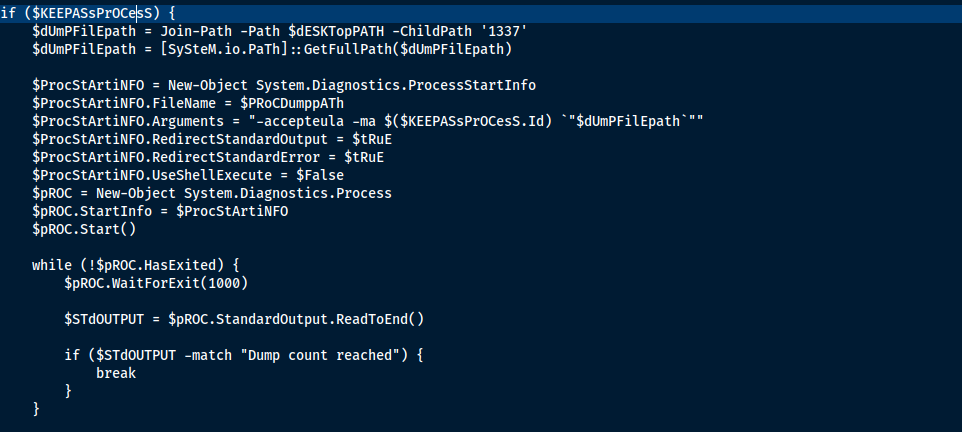
Let us analyze the PowerShell script. The first part of the script given below is some kind of a variable assignment.



The next part check if the process dump utility is present, if not present it will download the process dump from the Sysinternals website. Then it will check if KeePass is present.



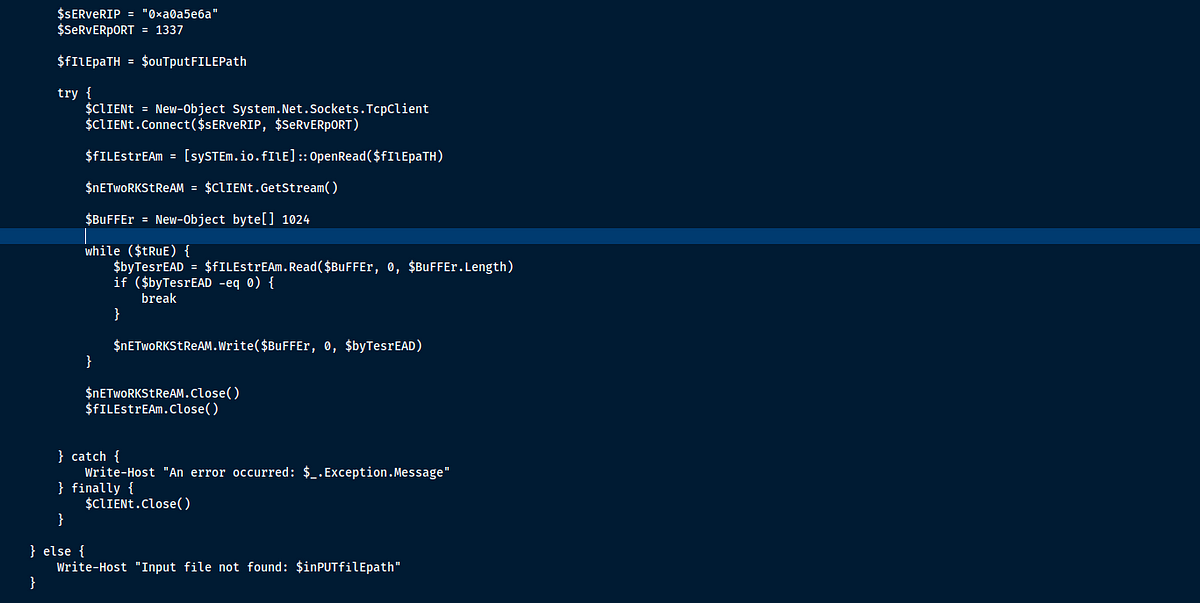
If KeePass exists, then it will execute the following code. This snippet will create a file called 1337 and then starts the Process Dump utility and writes the dump to the file created. This loop exits once the process dump is completed.



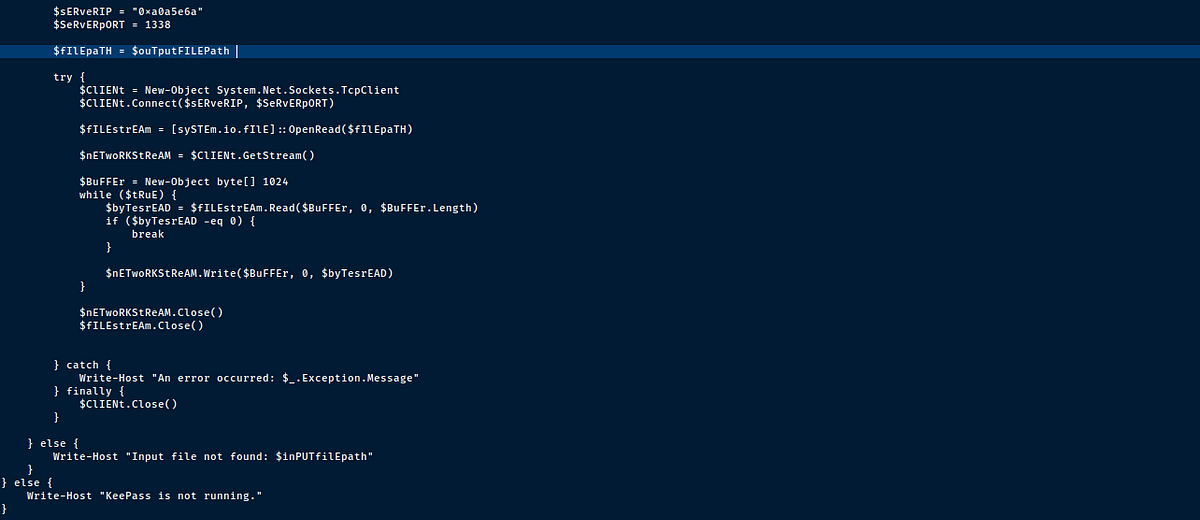
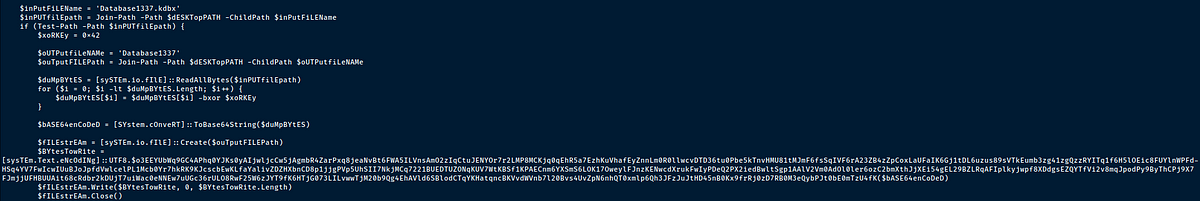
The next code snippet will take the contents of the above file which is 1337 and then implement a XOR encryption with a key — 0x41, then converts it into base64 and write the result into a file called 539.dmp.



Then it connects to an IP address over port 1337 and sends the contents of the newly created 539.dmp file in blocks of 1024 bytes.



Then it reads another file named Database1337.kdbx which is a keepass database file, then converts into xor with key — 0x42, then converts this into base64 encoded strings and then sends them over to the IP address over port 1338.



Now that we have analyzed the Powershell script, let us extract the data that was sent over from the victim to the attacker.

For this we will use Tshark which is a powerful command line utility to analyze network traffic.

First, let us extract the data sent over port 1337 with the following command

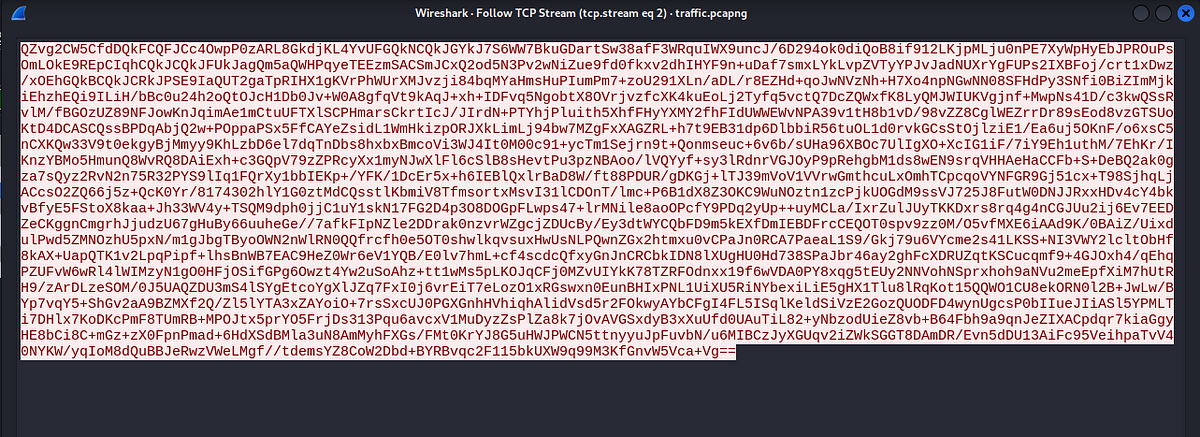
$ tshark -r traffic.pcapng -Y "tcp.dstport == 1337 and frame.len > 100" -T fields -e data | xxd -ps -r > 539.dmp

The above command extracts all the data sent over the tcp port 1337, then converts it from hex to base64 and stores it in a file name ‘539.dmp’

Then we can extract the keePass database file sent over the port 1338. For this we can just extract the data from wireshark, as it is very small. In wire shark apply the following filter.

tcp.dstport == 1338 && data.data

Then click on any packet and select follow TCP stream.



Copy this and save it in a file named database.dmp.

Now, we can start decoding the data. First we will start by decoding the 539.dmp file. The attacker did the following sequence of operations on the data

1. Convert the data into XOR encryption with key 0x41 which translates to ‘A’
2. base64 encoding
3. divide the data into streams of 1024 bytes.

Now, we have to perform these operations in reverse order to decode the data. For this I am using a python script, which is as follows.

import base64

with open('539.dmp', 'r') as file:

encoded\_data = file.read()

binary\_data = base64.b64dcode(encoded\_data)

xor\_key = b'A'

decrypted\_data = bytearray(len((binary\_data))

for i in range (len(binary\_data)):

decrypted\_data[i] = binary\_data[i] ^ xor\_key[i % len(xor\_key)]

with open('1337.dmp', 'wb') as file:

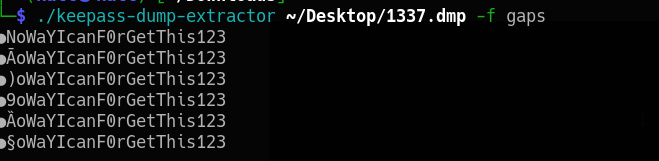
file.write(decrypted\_data)

print('decryption complete.')

The 1337.dmp file contains a memory dump of the KeePass, which due to a vulnerability leaves traces of password in the memory.

To extract this we can use a tool which is present [here](https://github.com/JorianWoltjer/keepass-dump-extractor). Use this tool as below to extract the password.

./keepass-dump extractor 1337.dmp -f gaps



We can see above that all of the password can be seen except for the first letter. Let us store all possible combinations to a file

./keepass-dump extractor 1337.dmp -f all > possible\_passes

Now, let us decode the database.dmp file. The same python script can be use to decode this file, just need to change the file names and the xor key which is 0x42 in this case. This translates to ‘B’.

import base64

with open('database.dmp', 'r') as file:

encoded\_data = file.read()

binary\_data = base64.b64dcode(encoded\_data)

xor\_key = b'B'

decrypted\_data = bytearray(len((binary\_data))

for i in range (len(binary\_data)):

decrypted\_data[i] = binary\_data[i] ^ xor\_key[i % len(xor\_key)]

with open('database.kdbx', 'wb') as file:

file.write(decrypted\_data)

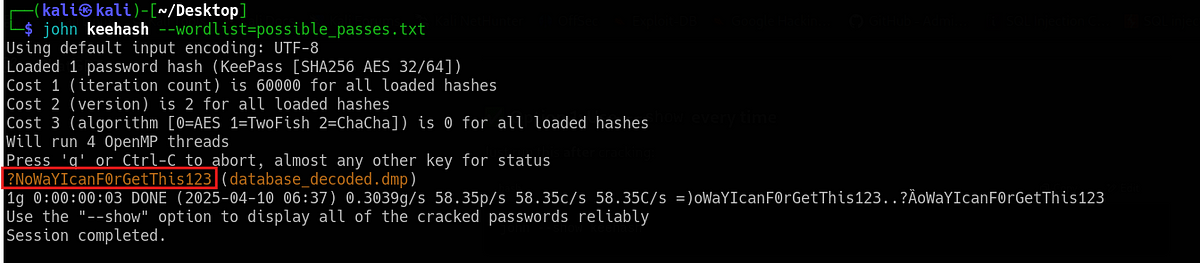
print('decryption complete.')

Now, from the database.kdbx file we can generate hashes for the KeePass database using the keepass2john tool.

keepass2john database.kdbx > keehash

Now, we can use the generate hash file, and the previously generated possible\_passes file we can generate the keePass password using john the ripper.

john keehash --wordlist=possible\_passes

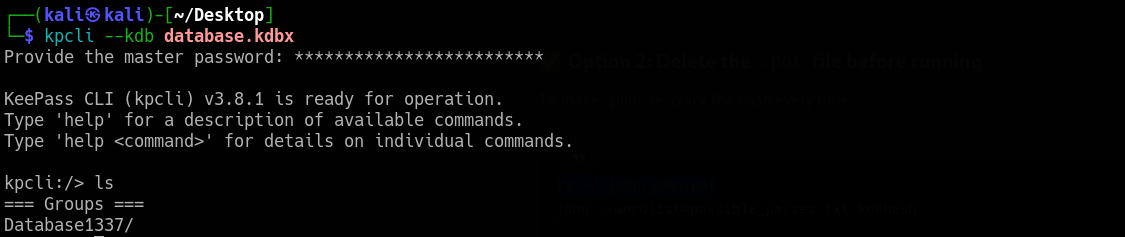


Password: ?NoWaYIcanF0rGetThis123

Now, we can connect to the database using the kpcli.

kpcli --kdb database.kdbx

Provide the password when prompted. Now we are in. We need to find the flag. Let us list the contents with ls.



Now let us go into Database1337.



We can see a file You win! let us see the contents of this file.



Answer: THM{B3tt3r\_Upd4t3\_Y0ur\_K33p455}.

This is the end of the room.