EDR Telemetry Generation Python Script

What is it?

This script is designed to generate data for Red Canary's EDR tool. It has five functions; start a command line process, create a file, modify a file, delete a file, and transmit data via a GET request. It is entirely developed in python and is supported across Windows, Linux/Unix, and Mac OSX.

How to use it:

The script is run through the command line using multiple different options. All options can be used in one call of the script if wanted/need be. It is run with the following format:

```
$ python <path_to_script>/telegen.py <option> <argument> <option> <argument> …
```

Command Line Options:

- `--startprocess "<process>": start a process/shell command`
- `--createfile <file_path>: create a file at the specified location`
- `--modfile <file_path>: modify a file at the specified location`
- `--deletefile <file_path>: delete a file at the specified location`
- `--startconnection: start a connection to google and transmit data`

File/Path Descriptions:

- telegen.py
  - The script itself.
- telegen_test.py
  - Base case unite tests for the script.
- ./logs
  - Directory holding all logs
- ./logs/connection_log.csv
  - Log for all outgoing connections
- ./logs/file_log.csv
  - Log for all file creations, modifications, and deletions
- ./logs/process_log.csv
  - Log for all processes ran

General Design Philosophy

My initial approach to this was to have individual files for all methods then import them into one file. This would have been a good approach for a much larger and robust command line tool but minimizing directory clutter was my priority. First, I decided to tackle starting a command line process. This was tricky for me since my understanding of it was poor. My initial impression was I would be running an executable file (.exe, .sh, .py, etc.) in which I wondered how I would handle all extensions. I asked the team how to handle this and quickly learned I would only be making shell command calls. I decided to go with python's subprocess module as it has "Popen" which allows me to run shell commands. Next on my list was tackling file actions. This was simple for me as I can open files with "open()" using proper options and error handling in order to get my desired result. For creation, I check if the path does not exist then attempt to open the file in write mode. If the path does not exist or any other error is thrown, we cancel file creation and print the error. For modification we check if the file exists and attempt to open the file in append mode. If it opens successfully, we append some text to the file. If any error is thrown, we print the error and cancel file modification. For deletion, we use the "os.remove()" method. We except any errors that may occur and print them out if file deletion is unsuccessful. Lastly, connecting and transmitting data. I used the socket module to connect to google.com and make a get request. All of this is logged in the corresponding CSV files in the logs directory. I also decided to make some base case unit tests in order to show I have knowledge of the python unittest module and basic understanding of unit tests. Unfortunately I did not have time to make comprehensive unit tests.