

# ISEC5000 - Computer Security Project

## Windows User guide

Written by Timothy Covich

I hope the solution that I have come up with will suit the needs of the client.

The goal of this application is to gather information about the system, and the calls it is making in the background unbeknown to the user. It will output captured information into several formats, inducing a human readable format.

There will be two ways to run the programs.

The first will be to run the python scripts directly, the second will be a pre-compiled packaged windows executable that will be a simple click and run.

Included files for Windows are as follows:

1. **tracemon.py**  
Tracemon is the heart of the program. It runs everything, sorts the outputs, and does all the statistical analysis.
2. **Procmon.exe**  
Process Monitor - A GUI program written and released by Microsoft (and SysInternals) to show everything that any application is doing in the background. Used to gather information in lieu of something like strace.
3. **config.pmc**  
The configuration file for Procmon.
4. **Listdlls.exe**  
A command-line program also written by Microsoft (and SysInternals), that will output DLLs being used but current running applications.
5. **REDIST folder**  
This folder will contain all the additional files and applications you would need to run on a fresh machine. Includes the Microsoft VC Redistributable Packages (x86 and x64), the Python 3.7.1 x64) setup, Procmon and ListDlls.
6. **Windows User Guide.pdf**  
This document, in all its glory.
7. **debug-tracemon.exe**  
This is a precompiled executable for Windows OS allowing for simple click-and-execute functionality for the above tracemon.exe. This should avoid the need of installing any python framework and allow standalone operation.  
There are known compatibility issues however.
8. **debug-killme.exe**  
A debug executable to use when the program hangs. Will close the ListDlls process and allow the program to continue.

## Installation Guide

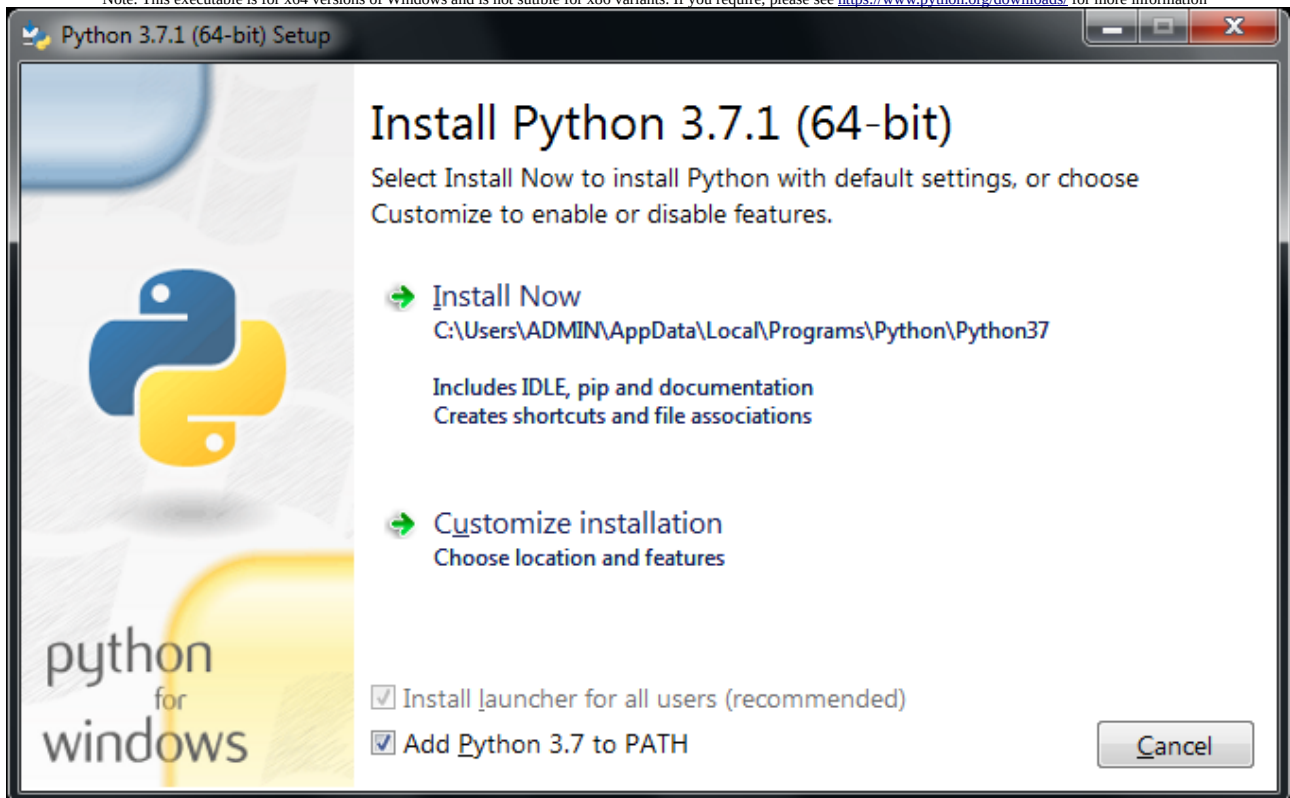
This project runs using the Python frameworks, and specifically Python 3 implementation. Most versions of Python 3 should work, however, this has mainly been tested with Python 3.6 and 3.7.

### 1. Python 3.7

Included is the 'python-3.7.1-amd64.exe'. Install that if your current system does not have a Python3 install currently

*(Python 3.7 is used here, although any python 3 variant should be fine).*

Note: This executable is for x64 versions of Windows and is not suitable for x86 variants. If you require, please see <https://www.python.org/downloads/> for more information



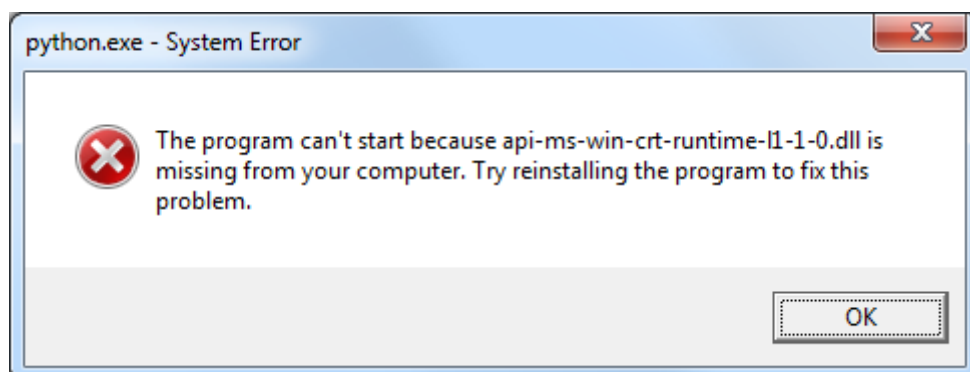
\*Check the 'Add Python 3.7 to PATH' for easier python use

You can simply use the 'Install Now' feature, unless you wish to customise your path, or use other options. You will *need* to install the 'pip' functionality (which is checked by default)

### 2. Microsoft Visual Redistributable Package

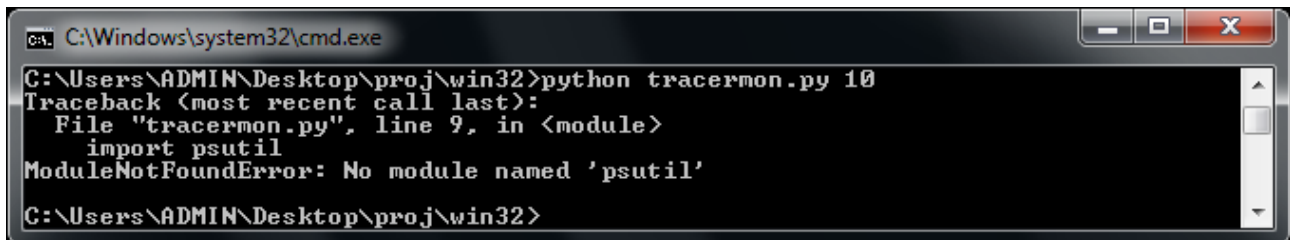
Included is the 'vc\_redist.x86.exe', which should be all you need to run Python3.

*(I will also included 'vc\_redist.x64.exe', but it should not be required)*



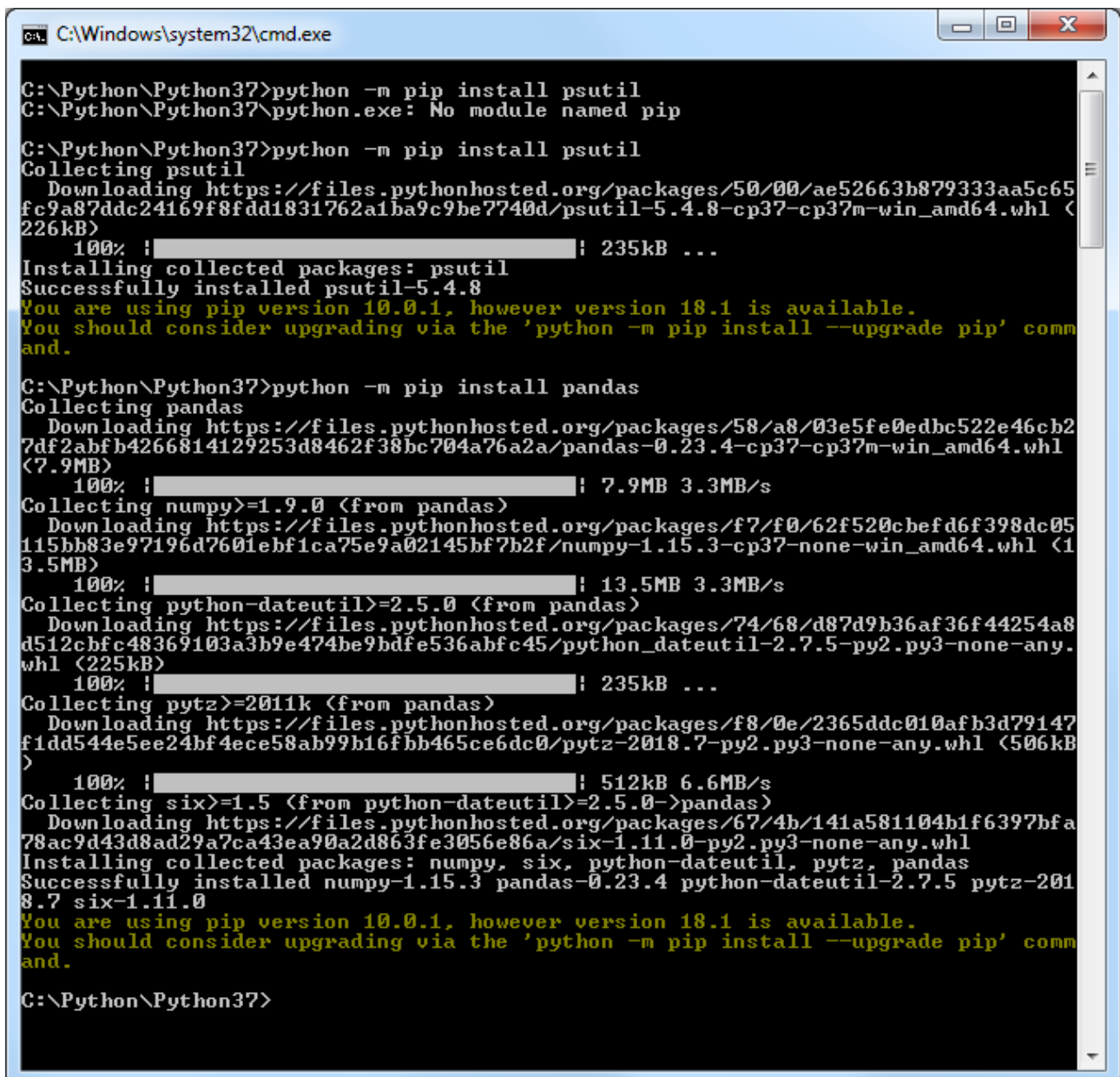
\*This is the type of error you get when you try to run python without having the vc redistributable installed

3. Installing the needed extra 2 addons for python — psutil and pandas from Windows Command Line Interface
  - > python -m pip install psutil
  - > python -m pip install pandas



```
C:\Windows\system32\cmd.exe
C:\Users\ADMIN\Desktop\proj\win32>python tracermon.py 10
Traceback (most recent call last):
  File "tracermon.py", line 9, in <module>
    import psutil
ModuleNotFoundError: No module named 'psutil'
C:\Users\ADMIN\Desktop\proj\win32>
```

\*This is the screen you get if you don't have these addons installed and try to run tracermon.



```
C:\Windows\system32\cmd.exe
C:\Python\Python37>python -m pip install psutil
C:\Python\Python37\python.exe: No module named pip

C:\Python\Python37>python -m pip install psutil
Collecting psutil
  Downloading https://files.pythonhosted.org/packages/50/00/ae52663b879333aa5c65fc9a87ddc24169f8fdd1831762a1ba9c9be7740d/psutil-5.4.8-cp37-cp37m-win_amd64.whl (226kB)
    100% |#####| 235kB ...
Installing collected packages: psutil
Successfully installed psutil-5.4.8
You are using pip version 10.0.1, however version 18.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Python\Python37>python -m pip install pandas
Collecting pandas
  Downloading https://files.pythonhosted.org/packages/58/a8/03e5fe0edbc522e46cb27df2abfb4266814129253d8462f38bc704a76a2a/pandas-0.23.4-cp37-cp37m-win_amd64.whl (7.9MB)
    100% |#####| 7.9MB 3.3MB/s
Collecting numpy>=1.9.0 (from pandas)
  Downloading https://files.pythonhosted.org/packages/f7/f0/62f520cbefdb6f398dc05115bb83e97196d7601ebf1ca75e9a02145bf7b2f/numpy-1.15.3-cp37-none-win_amd64.whl (13.5MB)
    100% |#####| 13.5MB 3.3MB/s
Collecting python-dateutil>=2.5.0 (from pandas)
  Downloading https://files.pythonhosted.org/packages/74/68/d87d9b36af36f44254a8d512cbfc48369103a3b9e474be7bdf536abfc45/python_dateutil-2.7.5-py2.py3-none-any.whl (225kB)
    100% |#####| 235kB ...
Collecting pytz>=2011k (from pandas)
  Downloading https://files.pythonhosted.org/packages/f8/0e/2365ddc010afb3d79147f1dd544e5ee24bf4ece58ab99b16fbb465ce6dc0/pytz-2018.7-py2.py3-none-any.whl (506kB)
    100% |#####| 512kB 6.6MB/s
Collecting six>=1.5 (from python-dateutil>=2.5.0->pandas)
  Downloading https://files.pythonhosted.org/packages/67/4b/141a581104b1f6397bfa78ac9d43d8ad29a7ca43ea90a2d863fe3056e86a/six-1.11.0-py2.py3-none-any.whl
Installing collected packages: numpy, six, python-dateutil, pytz, pandas
Successfully installed numpy-1.15.3 pandas-0.23.4 python-dateutil-2.7.5 pytz-2018.7 six-1.11.0
You are using pip version 10.0.1, however version 18.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Python\Python37>
```

\*installing the addons.

## Running Details

**NOTE:** To use the Windows portion of this project, you will need to be running a command prompt with elevated privileges, otherwise you will get popups asking you to confirm programs running multiple times.

You will need to either have set your python 3 executable path already (allowing you to call just python, or python3 (depending on how it's been setup)), or to point toward the python 3 executable yourself manually.

EG:

```
$> python3 tracermon.py
```

or

```
$> "C:\Program Files x86\Python 3\python.exe" tracermon.py
```

Furthermore, you can give the program a custom run-time

EG:

```
$> python3 tracermon.py 30
```

In which the program will collect information for thirty seconds, before processing.

Once the application has run its course, you can find all the information from the capture in organised folders within the “working” directory. They are sorted in a ISO 8601-style standard (YEAR-MONTH-DAY-HOUR-MINUTE-SECOND) into folders, so you should always be able to find your latest output.

Each folder will contain several files, including a ‘raw.pml’ which contains raw-unreadable information, an ‘output.csv’ which contains all the captured data in a comma separated value style output, and a ‘statistics.txt’ which contains an overview of all calls and accompanying DLLs.

## Authors Notes:

The statistics output is not sorted by PID at this time. They are listed in the order that they appear in the RAW output and the CSV output. I am not sure if this is an issue at this time, but I did not think it necessary or worth the extra in-program time/cycles.

The way it is now, it can sometimes take longer than the allotted wait time (ie 10 seconds) as getting the DLL information can sometimes take more time than it needs, leading to a delay and extra wait. Tried to get around this with a custom timer.

Probably should have given this program a better name. In hindsight the "*portmanteau*" of 'tracer' and 'process monitor' isn't very sexy.

Because of the way windows is compared to Linux, it was initially difficult to get this project off the ground. Linux has a very well put-together system in the form of strace (and its variants).

Windows does not have any of this native functionality. As I was given the honour of working on the Windows solution on my own, I went out to try to find a prepackaged solution, in the vein of strace. These solutions technically did exist in the form of NTtrace [<http://www.howzatt.demon.co.uk/NtTrace/>], straceNT [<http://intellectualheaven.com/default.asp?BH=StraceNT>], DrMemory [<http://www.drmemory.org>], plus a couple of others. However, none of these solutions was the magic cure-all for what I needed. They didn't work, crashes on startup, crashed the traced program, could only trace one process and hang the whole machine while doing so, or whatever else.

I happened on my current solution after dismissing it earlier. Microsoft's 'Process Monitor' is a beautiful GUI based program that captures calls made by the system and displays them for you in a nice little package. This was not what I was looking for however, needing a streamlined CLI-style implementation. Eventually, I learned that it could be run in a minimal fashion and its contents output in just the right way with some command line trickery.

With the gift of hindsight, I could have taken a much different approach to this project. While there is no perfect Windows solution, a DIY solution would likely have been possible given various Microsoft debugging tools and many hours of reading, debugging and possibly reverse-engineering of the Ntdll.dll among other things. This solution of building something from the ground up would have probably taken more time than was given, and I am not sure a working prototype could have been ready in the time-frame.

After tweaking and testing I have realised a misstep I have taken, I am only checking for DLL usage at the start of the application running. For an extended run, with things being open and accessed in the background, more DLLs and other applications may be loaded, and this is not taken into account.

In a future revision of this application, I would run this at the start (as it is now) and at the end, and during also, giving you a good safety net of all things running.

There is a way to list the DLLs that each program is using with a simple command line call: 'tasklist /m /fi "imagename eq chrome.exe"' or just 'tasklist /m' to list every single program running and their DLLs.

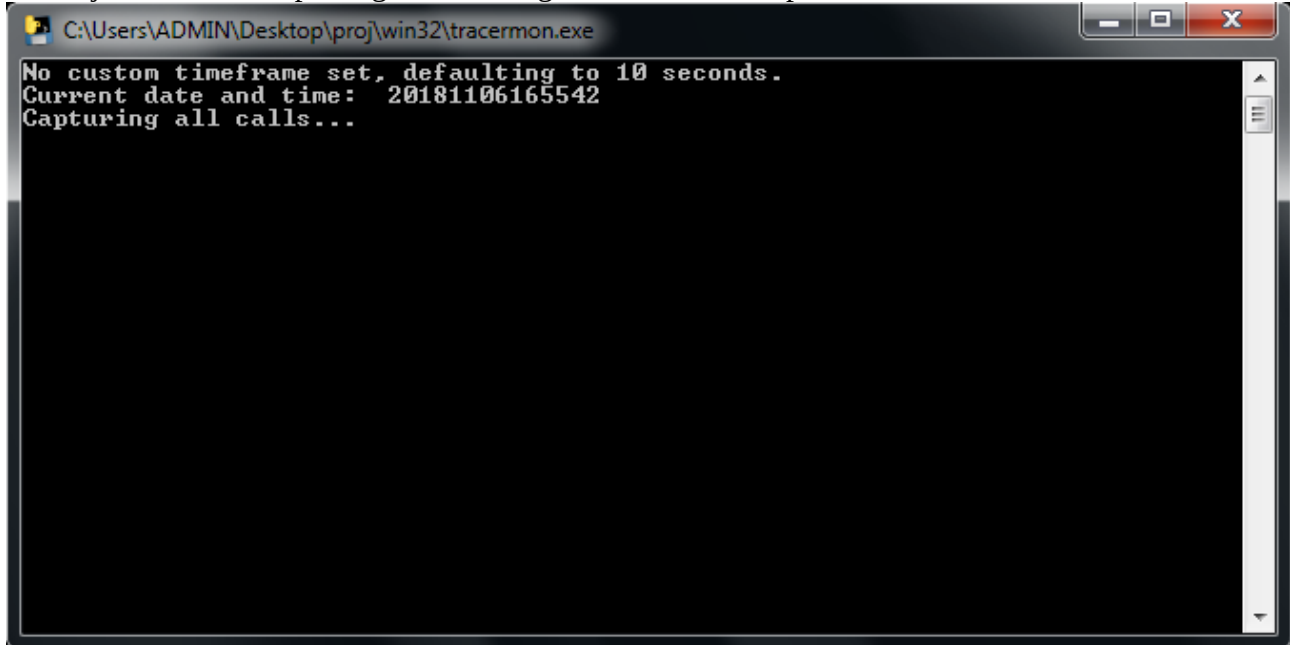
This may have been a better way to go in hindsight given the enormous troubles I eventually had with ListDlls. I also had issues getting this method to work from within python. I ran into issues trying to run a CMD via python and getting the output (worked via PowerShell, but couldnt rely on it being on every machine). There may have been a better way to achieve this by writing a file then reading that file again.

Tested on Windows 10 x64, Windows 7 x64 and Windows XP x32

## Bugs and Quirks:

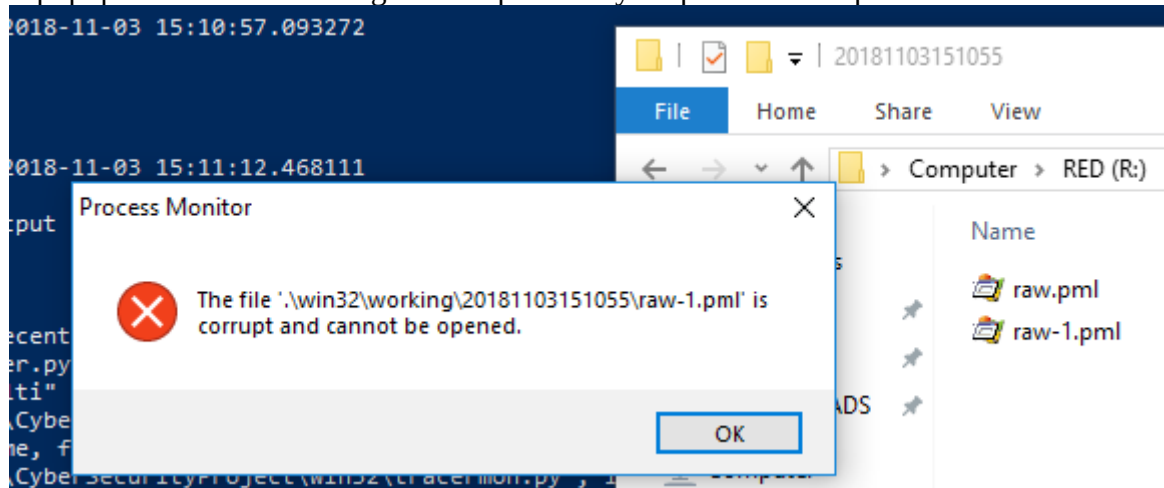
Originally, I had a bug where the DLL output method would just freeze for no reason. I ended up rewriting how it was called in an effort to stop it from hanging. Tested it for a long time while adding new features and squashing bugs... However, after compiling the python script into an executable, on Windows 10, I am seeing this hang more often than not.

It will just sit here, 'capturing' and waiting for the ListDLLs processes to close...



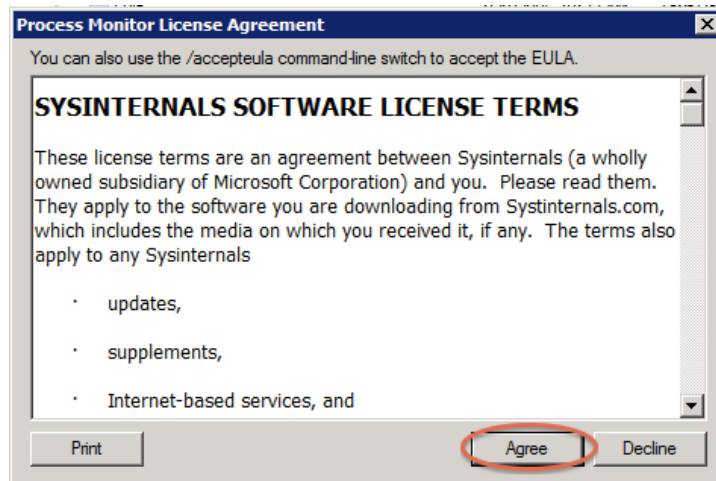
While it should not be an issue any longer when running the python script directly, I have included a 'debug-killme.exe' which will simply stop the processes running.

I got a popup once or twice telling me that part of my output was corrupt:



This doesn't seem to affect the output, or the program running.

Occasionally on a fresh machine install, you will get a popup when the program runs which looks similar to this:



I am not sure why this shows up as I am using the /accepteula as suggested when running the program myself. A strange bug perhaps? Accept/agree to continue.