COMP 3670- FINAL PROJECT

Charles Corro, Keerthana Madhavan, Van Minh Ngai,
Nitin Ramesh, Bilal Sohail

Option two: Network Tools

GitHub repository: https://github.com/NitinR99/3670FinalProject/
(Repo will be made public after the due date)

Note to the grader

Our idea for the final project was informed to the professor, who did not raise any issues.

Regarding COMP 3670

NR

Nitin Ramesh
Sun 2020-12-06 4:21 PM
To: Sherif Saad Ahmed
Cc: Charles Corro; Keerthana Madhavan; Van Minh Ngai; Bilal Sohail

Good evening Dr Saad,

I am writing this email to let you know about the final project my team has decided to do.

We are going to make a python program with a menu to do these following actions:

1. port scanner
2. subnetting calculator
3. ping
4. traceroute
5. nslookup

It will be made with a GUI. We were thinking if this is an acceptable final project. Please let us know.

[redacted]
Please reply at your earliest convenience. Thank you.

Regards,
Nitin Ramesh

Sherif Saad Ahmed
Sun 2020-12-06 6:43 PM
To: Nitin Ramesh
Cc: Charles Corro; Keerthana Madhavan; Van Minh Ngai; Bilal Sohail

Hello Nitin

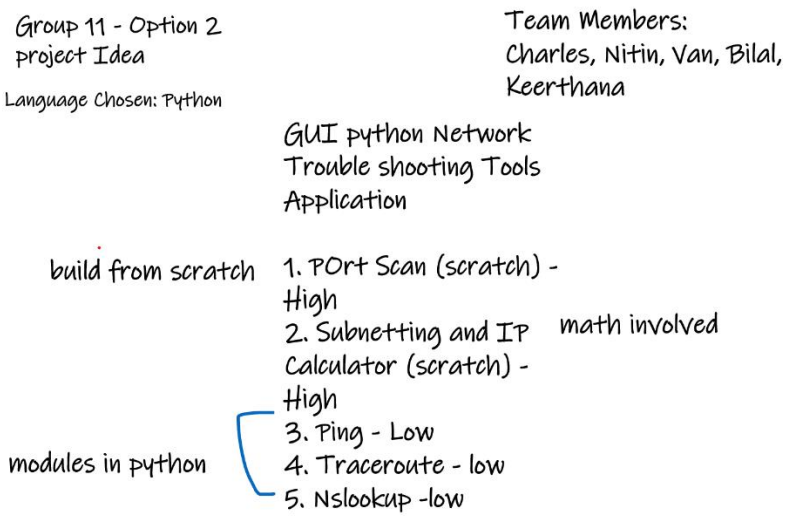
Thank you for letting me know that.

--
Sherif Saad, PhD
Assistant Professor
School of Computer Science
University of Windsor
Tel: 519.253.3000 Ext: 3793
https://www.linkedin.com/in/ebinsaad/

Table of Contents

Code overview

The responsibility of coding the tools where initially split between the team members. Once we were done coding them individually, we had a meeting on Teams, where we added the different components together in main.py. This was the breakdown:



The port scanner and Subnet&IP calculator were built from scratch without using any inbuilt functions. The ping, traceroute and nslookup tools use prewritten functions.

Project Overview

For our final project, we designed and developed a compilation of various network tools which can be accessed using a single graphical user interface. The tool includes the following functionalities:

1. A port scanner for a given IP address
2. Subnet and IP calculator
3. Sending a ping to a given IP address
4. Traceroute a given IP address
5. NSLookup a given IP address

The implementation of our project was done in Python 3. Tkinter was used for the GUI of our application.

Objectives

The network application we have developed can be used by users with ease, everyday. Instead of using the command prompt to run commands for ping, traceroute, and nslookup, running it on our application would be more efficient, as the output would automatically be saved to a text file.

Finding open ports for a particular IP address usually involves a download and install of a space-heavy application. Instead, using our application would save valuable space on your system, as well as save the output to a text file, which many port scanners which are available, do not do.

So, our objectives for the project can be summed up like so:

- User friendly GUI
- Light on resources
- Implement the mentioned functionalities
- Saves output to a text file

Design

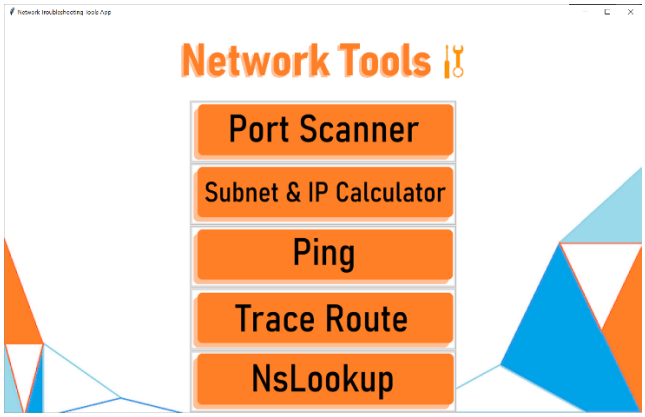
Front-end:

We designed the buttons for the application using Microsoft paint. Here is an example (Button for port scanner):

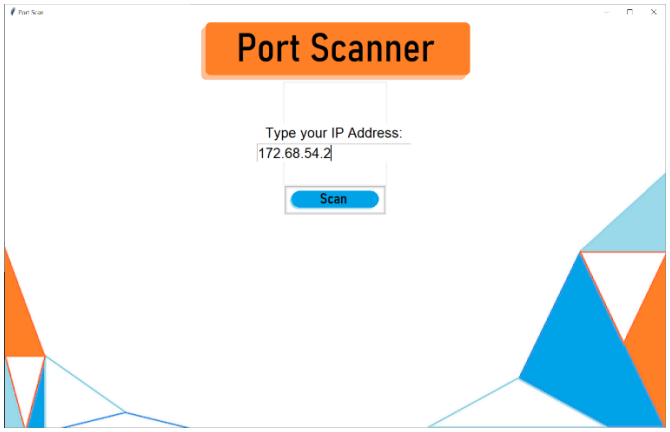
These type of images were used as an overlay for the buttons in our GUI.

The front end of our application was developed using Tkinter and Python 3. Here are the screenshots for the for the front end UI.

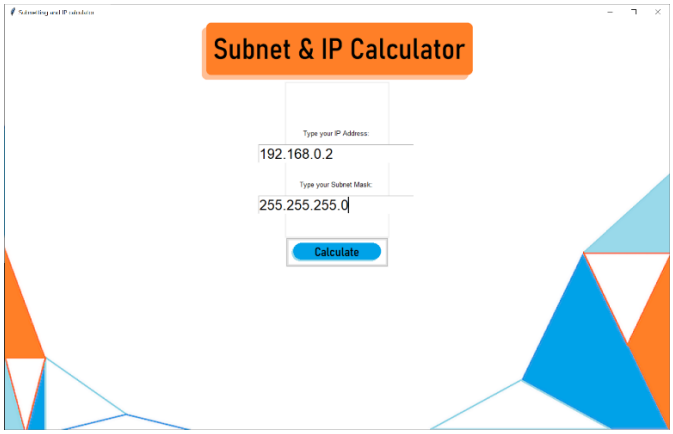
- Main screen



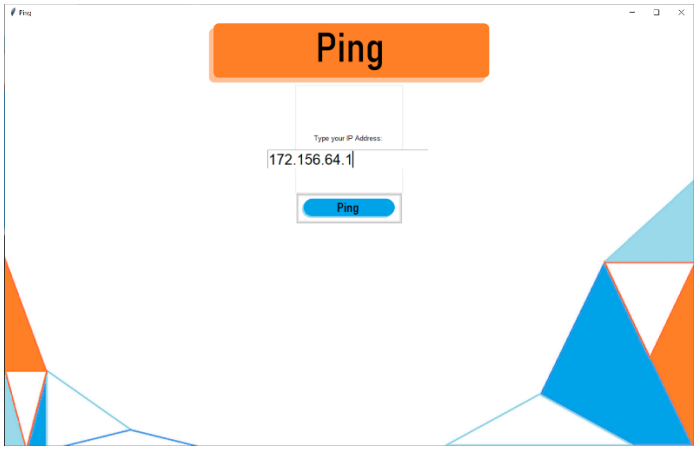
- Port scanner UI



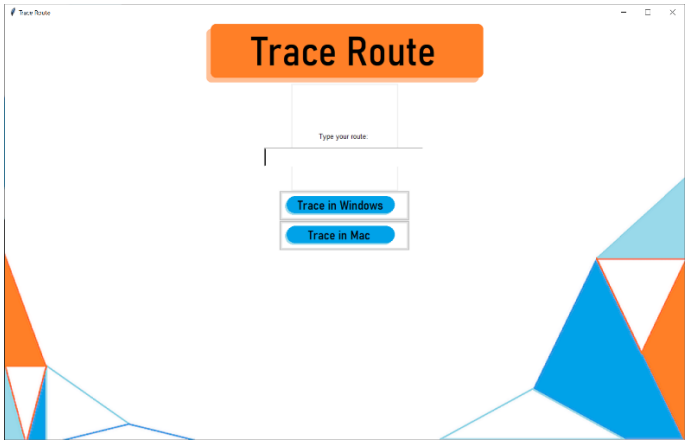
- Subnet & IP calculator UI



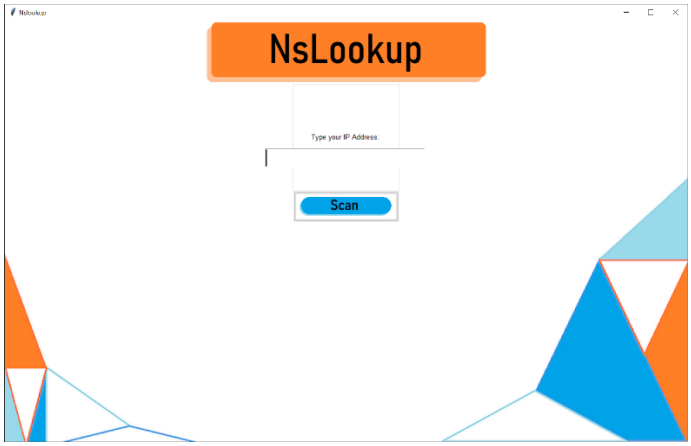
- Ping UI



- Traceroute UI



- Nslookup UI



Back-end:

1. Port Scanner (Multi-threaded operation)

```
import socket
import threading

def portScanner(host_ip='172.217.1.164'):

    connections = []        # To run connections at the same time
    result = {}         # all
    OpenPorts = []

    # translate hostname to IPv4
    ip = socket.gethostbyname(host_ip)

    # prints status block of target and when the scan starts
    print("-" * 50)
    print("Scanning: " + ip)
    print("Scanning began at: " + str(datetime.now()).split('.')[0])
    print("**approximate runtime is 1 minute 30 seconds**")
    print("-" * 50)

    # Spawning threads to scan ports
    for a in range(65535):
        t = threading.Thread(target=TCP_connect, args=(ip, a, result))
        connections.append(t)

    # Starting threads
    for b in range(65535):
        connections[b].start()

    # Locking the main thread until all threads complete
    for c in range(65535):
        connections[c].join()

    # Printing open ports
    for d in range(65535):
        if result[d] == 'open':
            print("Port", d, 'is', result[d])
            OpenPorts.append(d)

    # Printing open ports in the list

    print("\nThe Open Ports are:", OpenPorts)
    # Print out Completion time
    print("\nScanning has finished at ", str(datetime.now()).split('.')[0])

def TCP_connect(ip, port, result):
    TCPsock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    TCPsock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    TCPsock.settimeout(1)
    try:
        TCPsock.connect((ip, port))
        result[port] = 'open'
    except:
        result[port] = ''
```

To scan ports of a given IP address quickly, each port would be scanned using a separate thread. Depending on your network and CPU speed, it can take anywhere from 5 seconds to 30 seconds to finish scanning all 65535 ports. This application will only inform the user of open ports. Filtered ports are classified as closed ports. Output handling (writing to an external file) is done in another part of the code

2. Subnet and IP Calculator

```
def subnet_calculation(ip_address, subnet_mask):
    print("\n")

    # calculate subnet based on IP and subnet mask

    # convert the mask to binary string

    mask_octets_pad = []
    mask_octets_dec = subnet_mask.split(".")

    for i in range(0, len(mask_octets_dec)):
        binary_octet = bin(int(mask_octets_dec[i])).split("b")[1]

        if len(binary_octet) == 8:
            mask_octets_pad.append(binary_octet)
        elif len(binary_octet) < 8:
            binary_octect_pad = binary_octet.zfill(8)
            mask_octets_pad.append(binary_octect_pad)

        dec_mask = "".join(mask_octets_pad)

        # calculate and count num of bits in host or a subnet

        num_zeros = dec_mask.count("0")
        num_ones = (32 - num_zeros)
        num_of_hosts = abs(2 ** num_zeros - 2)

        wildcard_octets = []
        for i in mask_octets_dec:
            wi_octet = 255 - int(i)
            wildcard_octets.append(str(wi_octet))

        wildcard_mask = ".".join(wildcard_octets)

        # print(wildcard_mask)

    # convert the IP to binary

    ip_octet_pad = []
    ip_octet_dec = ip_address.split(".")

    for i in range(0, len(ip_octet_dec)):
        binary_octet = bin(int(ip_octet_dec[i])).split("b")[1]

        if len(binary_octet) < 8:
            binary_octect_pad = binary_octet.zfill(8)
            ip_octet_pad.append(binary_octect_pad)
        else:
            ip_octet_pad.append(binary_octet)

    binary_ip = "".join(ip_octet_pad)

    binary_network_address = binary_ip[:(num_ones)] + "0" * num_zeros
    # print(binary_network_address)

    binary_broadcast_address = binary_ip[:(num_ones)] + "1" * num_zeros

    # print(binary_broadcast_address)

    net_ip_octets = []
    for i in range(0, len(binary_network_address), 8):
        n_ip_octet = binary_network_address[i:i+8]
        net_ip_octets.append(n_ip_octet)

    net_ip_address = []
    for i in net_ip_octets:
        net_ip_address.append(str(int(i, 2)))

    network_address = ".".join(net_ip_address)

    max_ip_octets = []
    for i in range(0, len(binary_broadcast_address), 8):
        m_ip_octets = binary_broadcast_address[i:i+8]
        max_ip_octets.append(m_ip_octets)

    max_ip_address = []
    for i in max_ip_octets:
        max_ip_address.append(str(int(i, 2)))

    broadcast_address = ".".join(max_ip_address)

    # printing all the results
    print("\n")
    print("Network address is: %s" % network_address)
    print("Broadcast address is: %s" % broadcast_address)
    print("Number of hosts in subnet: %s" % num_of_hosts)
    print("Wildcard mask is: %s " % wildcard_mask)
    print("Mask bit is: %s " % num_ones)
```

The subnet calculator will take an IP address and subnet mask to calculate the resulting network address, broadcast address, number of hosts on the subnet, wildcard mask and the mask bit.

Step 1: Enter IP address and Subnet Mask

Function subnet_calculation() will perform all the calculation

Step 2: convert the subnet mask to binary string.
For example, 255.255.255.0 to 11111111 11111111 11111111 00000000.

Step 3: calculate the number of hosts on the subnet, 32 – the number of zeros in the mask.

Step 4: The wildcard mask is the inverse netmask used for access controls obtain the wildcard mask

Step 5: Convert the IP address to binary string.

Step 6: obtain the network address and broadcast address using the binary version of IP address and subnet mask

Step 7: Print
the network_address, broadcast_address, num_of_hosts, wildcard_mask, num_ones.

3. Ping

```
def ping(ip):
    print('-'*60)
    try:
        # if it is not != 0 then something is wrong and raise Exception
        if os.system('ping {}'.format(ip)) != 0:
            raise Exception('IP does not exist')
        Outputfileobject = os.popen('nslookup {}'.format(ip))
        Output = Outputfileobject.read()
        Outputfileobject.close()
        label3 = tk.Label(newWindow, text=Output, font=('helvetica', 10))
        label3.pack()
    except:
        print('Command does not work')
    print('-'*60)
```

A ping is a signal sent to a host and requests a response. It serves two purposes; to check if the host is available and to measure how long each response takes. This part of the code pings a given IP address with a default packet and the output is shown on the GUI and also saved separately in a text file.

4. Traceroute

```
# Command for Window user
def tracert(route_to_test):
    print('-'*60)
    try:
        # if it is not != 0 then something is wrong and raise Exception
        if os.system('tracert {}'.format(route_to_test)) != 0:
            raise Exception('Route does not exist')
    except:
        print('Command does not work')
    print('-'*60)

# Command for Mac user
def traceroute(route_to_test):
    print('-'*60)
    try:
        # if it is not != 0 then something is wrong and raise Exception
        if os.system('traceroute {}'.format(route_to_test)) != 0:
            raise Exception('Route does not exist')
    except:
        print('Command does not work')
    print('-'*60)
```

Traceroute or tracert are commands used to display possible paths and measuring transit delays of packets across an IP network. This part of the code is split into 2. One is for users running the application on Windows OS, the other is for users running Mac OS. It executes a “traceroute <ip address>” or “tracert <ip address>” with a given IP address. Output is displayed on the GUI as well as save to a text file.

5. NSLookup

```
def nslookup(Host_name):
    print('#'*70)
    # Host_name = raw_input('Enter the host name: ')
    print('-'*70)
    os.system('nslookup {}'.format(Host_name))
    print('-'*70)
    Outputfileobject = os.popen('nslookup {}'.format(Host_name))
    Output = Outputfileobject.read()
    Outputfileobject.close()
    label99 = tk.Label(newWindow, text='-'*70, font=('helvetica', 10))
    label99.pack()
    label3 = tk.Label(newWindow, text=Output, font=('helvetica', 10))
    label3.pack()
    label99 = tk.Label(newWindow, text='-'*70, font=('helvetica', 10))
    label99.pack()
```

The nslookup command is used to find the IP address for a specific host, or vice versa. This code is used to run a nslookup on a given IP address. The output is shown on the GUI and saved to an external text file.

Output to text file

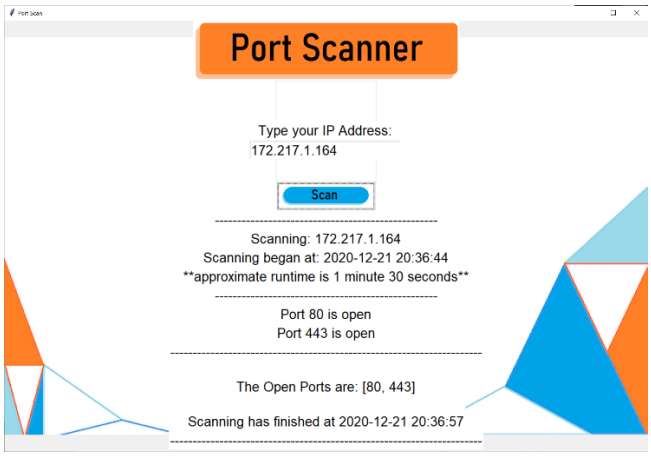
We have a logfile.txt, where all the output of our tool gets written (appended). Users will have to clear the file after each execution of main.py to avoid confusion. It will work even if you do not clear it. The log file will be in the same directory as main.py .

Experiment and Result (Test cases)

Here, we will show you the test cases we used to check if our code worked.

1. **Test 1: Find open ports for 'www.google.com' [172.217.1.164]**

   Output on UI:



   Text file content:

   None
   Scanning: 172.217.1.164
   None
   Scanning began at: 2020-12-21 20:36:44
   None
   approximate runtime is 1 minute 30 seconds
   None

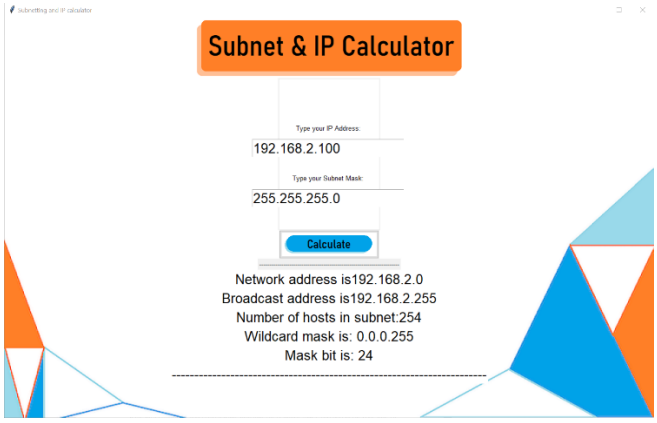
   None
   Port 80 is open
   None
   Port 443 is open
   None

   The Open Ports are: [80, 443]
   None

   Scanning has finished at 2020-12-21 20:36:57
   None

2. **Test 2: Calculate the subnet for IP address= “192.168.2.100” and Subnet mask= “255.255.255.0”**

   Output on UI:

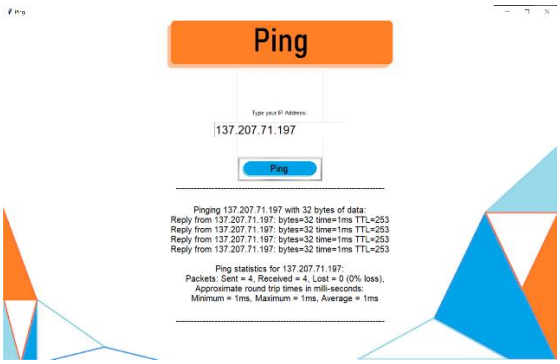


   Text file content:

   The result of subnet calcualtion:
   Network address is: 192.168.2.0
   None
   Broadcast address is: 192.168.2.255
   None
   Number of hosts in subnet: 254
   None
   Wildcard mask is: 0.0.0.255
   None
   Mask bit is: 24
   None

3. **Test 3: Send a ping to “www.uwindsor.ca” [137.207.71.197]**

   Output:



Text file content:

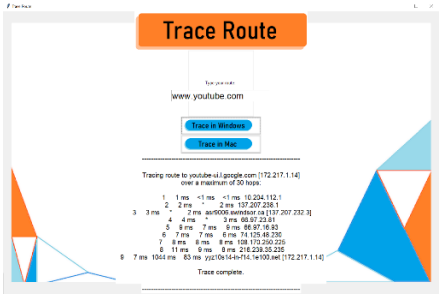
The result of ping is:

Pinging 137.207.71.197 with 32 bytes of data:
Reply from 137.207.71.197: bytes=32 time=1ms TTL=253
Reply from 137.207.71.197: bytes=32 time=1ms TTL=253
Reply from 137.207.71.197: bytes=32 time=1ms TTL=253
Reply from 137.207.71.197: bytes=32 time=1ms TTL=253

Ping statistics for 137.207.71.197:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 1ms, Average = 1ms

4. Test 4: Traceroute "www.youtube.com" [172.217.165.14]

Output:



Text file content:

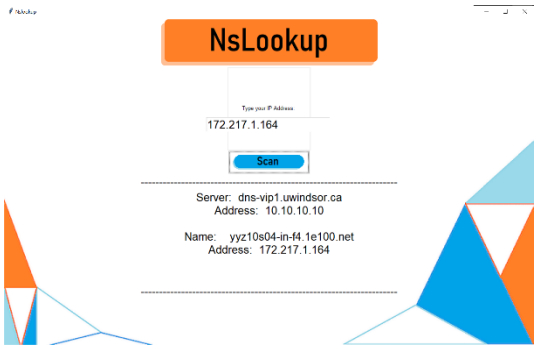
The result of traceroute is:
Tracing route to youtube-ui.l.google.com [172.217.1.14]
over a maximum of 30 hops:

1	1 ms	<1 ms	<1 ms	10.204.112.1
2	2 ms	*	2 ms	137.207.238.1
3	3 ms	*	2 ms	asr9006.uwindsor.ca [137.207.232.3]
4	4 ms	*	3 ms	66.97.23.81
5	9 ms	7 ms	9 ms	66.97.16.93
6	7 ms	7 ms	6 ms	74.125.48.230
7	8 ms	8 ms	8 ms	108.170.250.225
8	11 ms	9 ms	8 ms	216.239.35.235
9	7 ms	1044 ms	83 ms	yyz10s14-in-f14.1e100.net [172.217.1.14]

Trace complete.

5. **Test 5: nslookup “www.google.com” [172.217.1.164]**

Output:



Text file content:

The result of nslookup is
Server: dns-vip1.uwindsor.ca
Address: 10.10.10.10

Name: yyz10s04-in-f4.1e100.net
Address: 172.217.1.164

NOTE: *Execution times will vary with different types of system. If the window says “not responding”, just wait for 30 seconds. Do not close the window.*