

## DOCUMENTATION FOR PACKET SNIFFER

This program sniffs packets from a user selected interface, analyzes the packets and sends an email to appropriate admin if user exceeds bandwidth or visits blacklisted IP addresses. The admin account is currently Email: testcat470@gmail.com Password: SWEProjCSC470

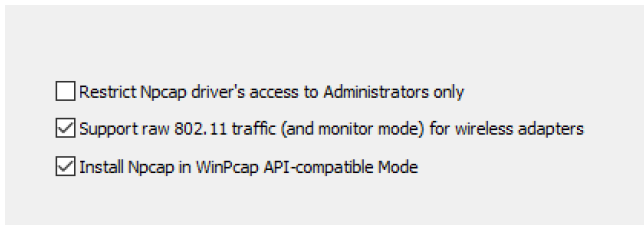
## Dependencies and Libraries:

Windows 10 Operating system  
NMAP  
PCAP  
Microsoft Visual Studio

## How to Compile and run

1. Download the required libraries and set up dependencies:
2. Create a project folder on local Drive
3. Go to <https://nmap.org/npcap/> The Npcap 1.60 installer will need to be downloaded and run
  - Npcap 1.60 installer for Windows 7/2008R2, 8/2012, 8.1/2012]
  - Npcap SDK 1.12 (ZIP).
  - Npcap 1.60 debug symbols (ZIP).
  - Npcap 1.60 source code (ZIP).

Download and install with these options:

- 
- A screenshot of the Npcap installation window showing three options with checkboxes. The first option, 'Restrict Npcap driver's access to Administrators only', is unchecked. The second option, 'Support raw 802.11 traffic (and monitor mode) for wireless adapters', is checked. The third option, 'Install Npcap in WinPcap API-compatible Mode', is also checked.
- ☐ Restrict Npcap driver's access to Administrators only
  - ☒ Support raw 802.11 traffic (and monitor mode) for wireless adapters
  - ☒ Install Npcap in WinPcap API-compatible Mode

4. Open Microsoft Visual Studio and clone the git repo from <https://github.com/KevinGranlund/SWEProj>
5. Make sure configuration manager is set to x86 in Visual Studio
6. Run program

## Functions:

### Header Files

- Blacklist.h:
  - This file defines the Blacklist class
- EmailFunction.h: Defines the EmailFunctionality functions
  - *resolveAddress(string testAddress)* - accepts a string which is either an ipv4, ipv6 or hostname and returns void, then resolves it to DNS, this function runs a powershell script with given DNS.
  - *sendEmail()* - sends email to users.
  - *buildDns()* - accepts no argument and returns void, function builds DNS using powershell scripting
  - *buildDataFiles()* - accept no argument and return void, function creates a blacklist file if none exist
  - *buildFiles()* - accepts no argument and returns void, functions calls three other functions: *buildDns()*, *buildEmail()*, and *buildDataFiles()*
  - *writesData()* - accepts a long integer and returns void. This function writes data to a text file called dataUsage.txt. The data includes the amount of bytes used and the timestamp of every 100,000 bytes used. Data in excess of 100,000 bytes is rolled into the next report. Upon reaching 100 reports an additional report will be generated with the email containing any left-over data. The trigger point of 100kb is variable and was chosen for quickness of testing.
- IP6Packet.h
  - Defines the IP6Packet class to access ipv6 addresses
- Packet\_structs.h: contains eight user defined struct data type used in the program
  - TCPheader
  - Ethernet\_header

- Udp\_header
  - Dummy\_struct
  - Ip6\_header
  - Ip\_header
  - Ip6\_address - sixteen(16) unsigned char byte values
  - Ip\_address - four(4) unsigned char byte values
- Packet.h: Defines the Packet class
- UserInfo.h: Defines the UserInfo class
- Resource.h: Visual studio automatically generated macros
- ComparisonOperatorOverloads.h: function compares ip addresses
  - operator() - accepts two const ipv4 addresses, compares them and returns true if they match and false if they don't
  - operator() - accepts two const ipv6 addresses, compares them and returns true if they match and false if they don't
- LocalLPHelperFunctions.h: declares IPv4 and IPv4 functions, it turns IPv4 and IPv6 address into useful format to be used in other parts of the code
  - *ipt6tos()* - accepts pointer to sockaddr struct, char pointer and int and returns an string address
  - *iptos()* - accepts an unsigned long int and returns an ip\_address struct of IPv4
- *Defines the Hexdump class. Sets the default length of the lineWidth to 64 (16 bytes)*
  - *str()* - output the string created within Hexdump

### C++ Files

- SWEProj.cpp: Contains functions and include files that initiates the program logic
- Blacklist.cpp: defines four function and a constructor
  - generateAddresses() - accepts a UserInfo and returns a void
  - checkBlackListIPv4() - accepts an IPv4 address and returns bool
  - checkBlackListIPv6() - accepts an IPv6 address and returns a bool
  - checkHostName() - accepts a string; resolve the hostname DNS, compare against blacklist IP's and return bool
- SWEProj.cpp
  - *CharToBinary()* - accepts char byte input and returns binary string of the byte input
  - *PortResolution()* - Refactored, not needed in the final design
  - *BinaryToDecimal()* - accepts three arguments(string input, int, int) returns a vector of two integers made out of the substrings that represents the binary digits from the bytes
  - *ipv4IPheaderToString()* - accepts an a pointer to ip\_header and returns a string value of the ip\_header
  - *ZeroPaddingHelper()* - accepts an unsigned char byte, returns a string, used for correctly formatting string representation of IPv6 addresses
  - *IP6addressToString()* - accepts an ip6\_address struct address and returns a formatted hex string representation of the address
  - *networkCheckIP6()* - accepts an ip6\_address struct address, checks if an IPv6 address is on a public or private network. It returns 0 for in network and 1 for out of network.

- *networkCheckIP4()* - accepts three ip\_address struct arguments and returns int; function checks if IPV4 packets originates in network, returns 0 for in network and 1 for out of network.
- *dataWatch()* - accepts no input and returns void. Monitors data usage for threshold limits, calls writeData() and sendEmail functions when usage is at 50%, 75% and 100 %.
- *packet\_handler()* accepts three arguments, unsigned char, a pointer pca\_pkthd struct and an unsigned char pointer to pkt\_data. Function analyses incoming packets
- *main()* - Program initiator, user interface is outputted here.
- **UserInfo.cpp:**
  - *UserInfo()* - A dummy constructor used to to make instance of the global version
  - *UserInfo(pcap\_if\_t\*)* - Constructor
  - *setUserName()* - Sets the username property to that of the user that logged into the computer
  - *setComputerName()* - returns the private property computer name
  - *setIP4Address()* accepts no arguments and returns void. Function correctly formats string representations of IPv4 addresses
  - *getLocalIPAddress():* - gets private IP address
  - *getSubnetAddress()* - gets private subnet address
  - *getBroadcastIPAddress()* - gets the private broadcast address
  - *toString()* - not utilized
- **Packet.cpp**
  - *toString()* - Not yet implemented. This idea was to have this output all of the relevant information in the packet header to a string.

- *HexDump()* - Is passed a pointer to a packet and outputs the hex of the packet header into a string.
- *Packet()* - Is passed a pointer of an ip4 packet. Parses information within the packet and stores pointers as ip4header, TCP header, ethernet header, and defines the packet type as 4.
- **IP6Packet.cpp**
  - *IP6Packet()* - Is passed a pointer of an ip6 packet. Parses information within the packet and stores pointers as ip6header, TCP header, ethernetHeader, and defines the packet as type 6.
  - *toString()* - Not yet implemented. This idea was to have this output all of the relevant information in the packet header to a string.
  - *HexDump()* - Is passed a pointer to a packet and outputs the hex of the packet header into a string.
- **Hexdump.cpp**
  - *Hexdump()* - Is passed a pointer to any continuous data, the size of the data in bytes, and the linewidth (how many bits of data per line). Writes a formatted string of the binary data converted to hex, and an ascii representation of this data to a string.

## Program Testing

There weren't many things to comprehensively test in this program. I disabled all of my network devices but couldn't figure out how to disable the loopback adapter on Windows 10 so I was unable to verify that the program would stop if there were no devices found.

```
1. \Device\NPF_Loopback (Adapter for loopback traffic capture)
Enter the interface number (1-1):
```

We were able to check all the points of the program where data was input either by the user or from file:

```
1. \Device\NPF_Loopback (Adapter for loopback traffic capture)
Enter the interface number (1-1):r
ERROR: a number must be entered: 5

Interface number out of range.
Enter the interface number (1-1):3

Interface number out of range.
Enter the interface number (1-1):
```

```
1. \Device\NPF_Loopback (Adapter for loopback traffic capture)
Enter the interface number (1-1):1
Invalid entry in ip6 blacklist 500,235,634,3
C:\Users\patri\source\Repos\SWEProj\x64\Debug\SWEProj.exe (process 21804) exited v
To automatically close the console when debugging stops, enable Tools->Options->D
```

The data warning worked and successfully stopped the program:

```
Data Warning Number: 100

C:\Users\patri\source\Repos\SWEProj\x64\Debug\SWEProj.exe (process 25972) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Auto
le when debugging stops.
Press any key to close this window . . .
```

Proper emails have been sent and documented for sites that were on test blacklists and data usage that goes over the set max. This has all been done manually by running through the program several times.

## Wrap Up

Overall, the program was a success. It was originally viewed as something that may not be either complex or take long enough to meet the requirements for the project. As development progressed, the largest time commitment was research. We were lucky enough to have a member of the team that is well versed in PowerShell and was able to set up all the output for the program beyond what was output to the screen. Another member of the team took most of the initiative and got a very good start on the program and was able to share documentation and knowledge with the rest of the group.

The difficulties came with abilities and communication. There was a communication breakdown in the final weeks that resulted in a bit of discord. This was exacerbated by the differing abilities of members of the group. In the end, everyone was able to contribute, although maybe not equally.

This was a great learning experience. Everyone in the group is at differing levels of ability with differing levels of time availability. Several members of the group stated there was at least 20 hours of research, with many hours of trial and error, spent gaining the knowledge needed to create a functional program that met the goals at the beginning of the project.