

Table of Contents

Aims.....	2
Description.....	2
Background research	2
Tool Implementation	3
Technologies	3
Class Diagram.....	4
Screens.....	5
Main Menu.....	5
Interfaces	5
Scan	7
Snapshot history	9
Challengers.....	11
Other Similar tools	12
IP Address Tracker	12
Angry IP Scanner (Free).....	12
NETDISCOVER.....	12

Aims

- Scan Network for connected devices
- Save scan results
- View or compare saved logs

Description

It is always better to be aware about the network which we are connected and about other devices which are connected to the same network. “Man, in the middle” attack is one of common attack vectors nowadays in public networks. To be aware about the network I have prepared a small tool to scan a network and obtain device NIC information. The main Features are,

- Select network interface card from the list and start scan. Ip range will be automatically calculated using the IP information.
- Save scan results into files
- View or compare saved logs.

Background research

To scan a network, we mainly need two points of information.

1. Network ID – “is the portion of an IP address that identifies the TCP/IP network on which a host resides. The network ID portion of an IP address uniquely identifies the host's network on an internetwork, while the host ID portion of the IP address identifies the host within its network.” Also, the first address of the IP range
2. Broadcast ID – “A broadcast address is a network address at which all devices connected to a multiple-access communications network are enabled to receive datagrams, which comprise UDP and TCP/IP packets, for instance. A message sent to a broadcast address may be received by all network-attached hosts.” Also, the last address of the IP range

With this information we can calculate available host range within the connected network. Network ID can be calculated using bitwise AND operation between any random IP address and the subnet mask. Example,

```
10000010.00101101.00100010.00100100 (ip address)
AND
11111111.11111111.11110000.00000000 (subnet mask)
=
10000010.00101101.00100000.00000000 = 130.45.32.0 (the resulting network address)
```

To obtain the broadcast address we have to perform bitwise OR operation between Network ID and Inversed Subnetmask. Example,

```

10000010.00101101.00100000.00000000 (netaddress)
OR
00000000.00000000.00001111.11111111 (inverted subnet mask)
=
10000010.00101101.00101111.11111111 = 130.45.47.255 (broadcast address)

```

With this information we can obtain the host range of the network.

To find the active device of a network, we can use two methods,

- Address Resolution Protocol – containing the IP addresses, MAC addresses, and allocation type
- Ping (ICMP Protocol) – This will enable you to further narrow down what devices are active

But depends on the network implementation and firewall rules the efficiency of the above methods may vary.

The tools main finality will enhance these methods to obtain active device infections.

Tool Implementation

Technologies

During the initial research I have identified many languages which can perform required operations to obtain information. Top selections were Python3 and .net Framework. But we chose to .net framework for the flexibility of windows form implementations. But I had to sacrifice the amount of code lines which we could use if we went with Python3 which is less.

.Net Framework (C#)	Python3
Windows Only (Newer Versions)	Platform Independent
Minimum configurations on deployment	Must configure the environment
Flexible GUI Designer in Visual Studio	Best on CLI operations
Both good at data structure implementation	
Compile code	Interpreter code
Closed source	Open source

Class Diagram

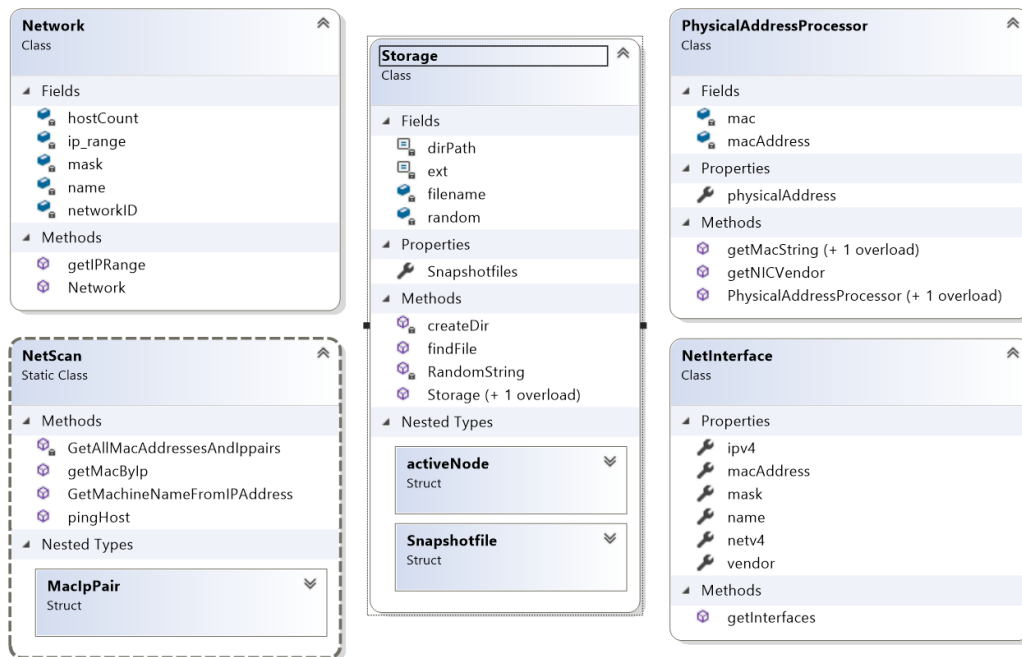


Figure 1 - Model Classes

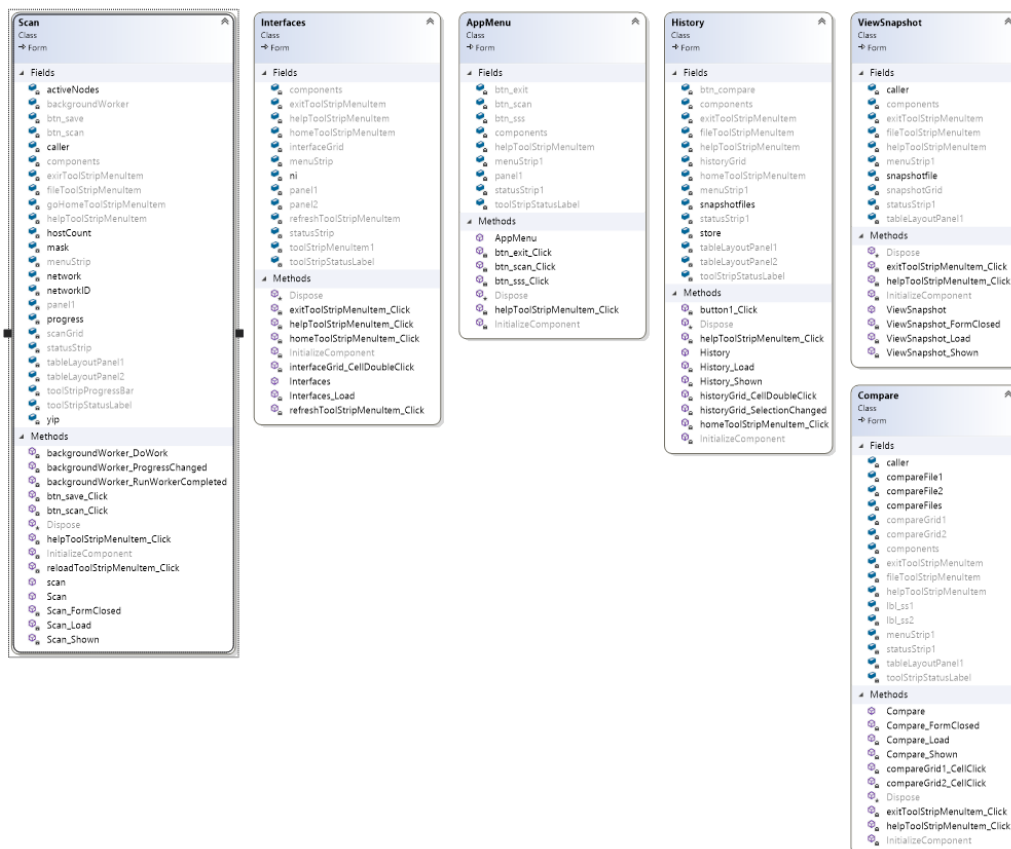


Figure 2 - Form Classes

Screens

Main Menu

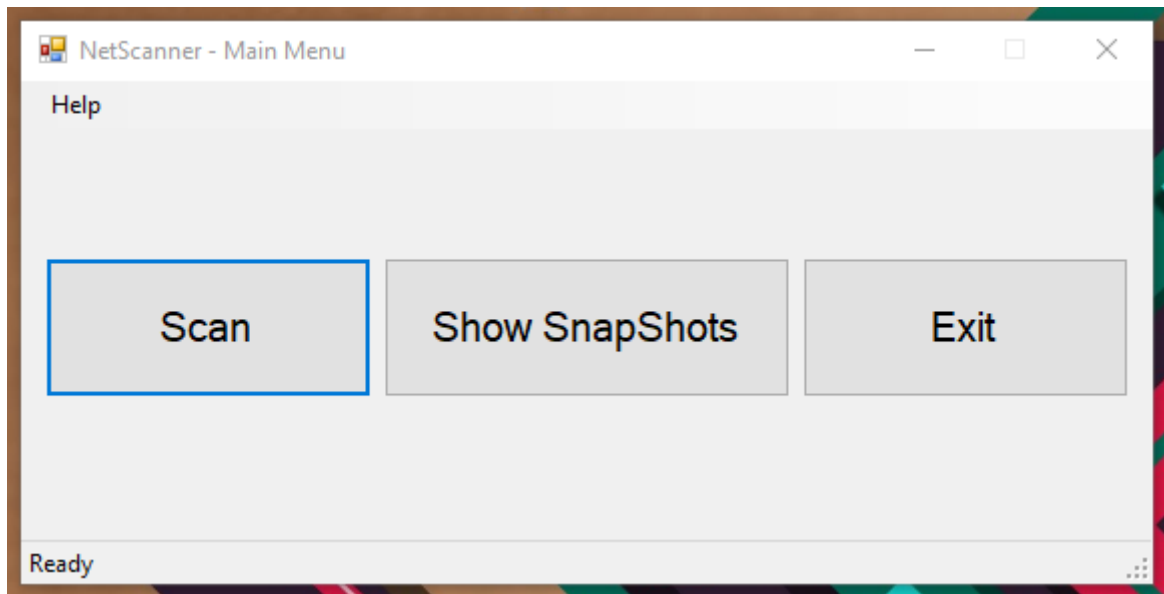


Figure 3 - Main Menu Form (APPMENU)

This screen presents the user with the main functionalities. Which are,

- Scan initiation
- View and compare snapshots

Interfaces

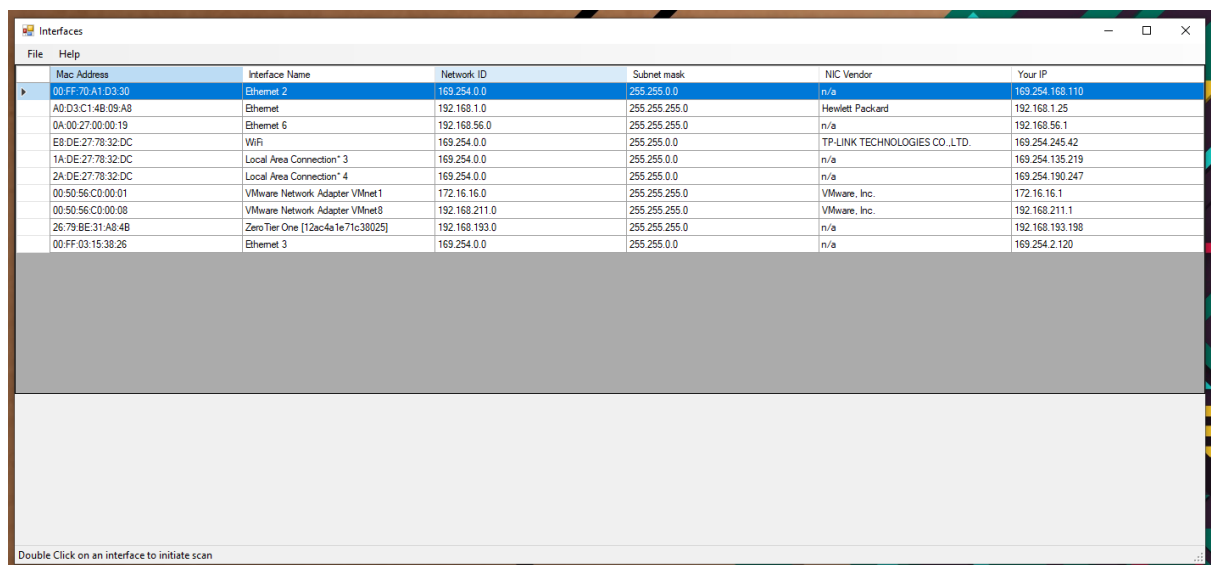


Figure 4 - Interface Selection

Nowadays most of the computers contain more than one network interface card. Therefore, during the initial designing I saw a reequipping of interface information screen

before beginning the scan. On this screen user will be presented with a list of interfaces with this information,

- NIC Physical Address (MAC Address)
- Human readable interface name
- Network ID
- Subnet Mask
- NIC manufacture (Vendor) – this has been identified using an inbuilt database of manufacture information corresponding to MAC address manufacture specific portion. I have provided this method inside the “PHYSICALADDRESSPROCESSOR CLASS”.

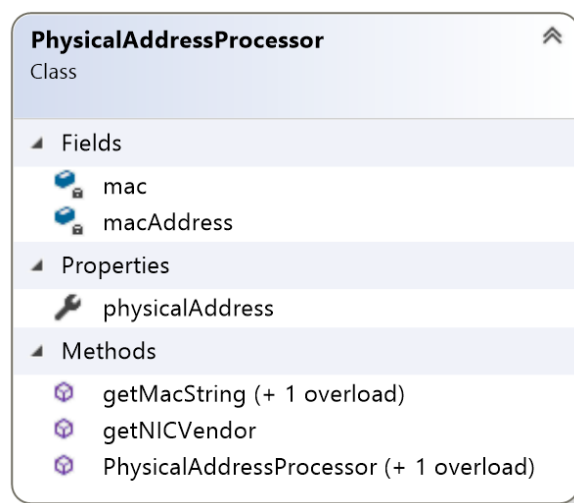


Figure 5 - PHYSICALADDRESSPROCESSOR Class

```
public string getNICVendor()
{
    var MacAddress = string.Join("-", this.physicalAddress.GetAddressBytes().Select(b => b.ToString("X2")));
    try
    {
        string mac = MacAddress;
        string vendor = "n/a";
        var assembly = Assembly.GetExecutingAssembly();
        var resourceName = "NetScanner.Resources.maclist.txt";

        using (Stream stream = assembly.GetManifestResourceStream(resourceName))
        using (StreamReader reader = new StreamReader(stream))
        {
            foreach (string result in reader.ReadToEnd().Split('\n'))
            {
                var macinfo = result.Split('|');
                if (macinfo[0] == mac.Substring(0, 8))
                {
                    vendor = macinfo[1];
                    break;
                }
            }
        }
        return vendor;
    }
    catch (Exception)
    {
        return "n/a";
    }
}
```

Figure 6 - PhysicalAddressProcessor.getNICVendor

Scan

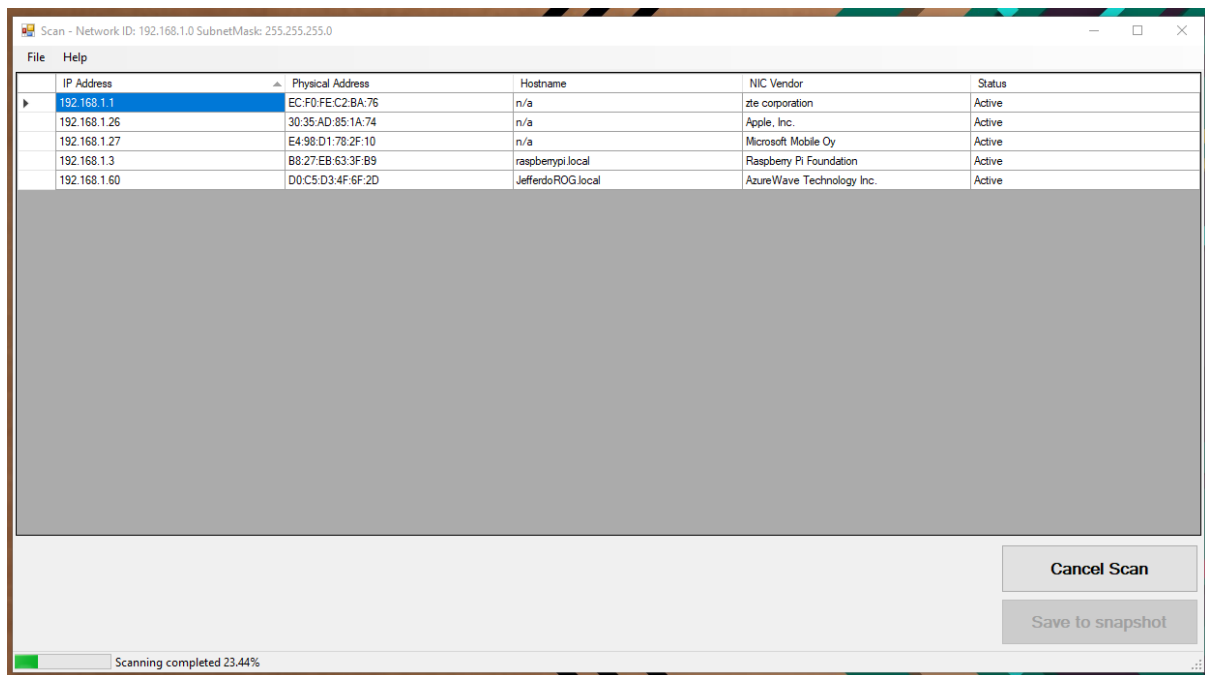


Figure 7 - Scan

On Shown event of this screen, scan job will be initiated, and the progress status is displayed status strip on the bottom of the window. User has two buttons to perform specific functions.

1. Scan Start and Cancel
2. Save to snapshot – will be disabled after saving snapshot.

```
public Scan(string networkID, string mask, string yourIP, Form caller)
{
    InitializeComponent();
    backgroundWorker.WorkerReportsProgress = true;
    backgroundWorker.WorkerSupportsCancellation = true;
    this.networkID = IPAddress.Parse(networkID);
    this.mask = IPAddress.Parse(mask);
    this.yip = IPAddress.Parse(yourIP);
    this.caller = caller;
}
```

Figure 8 - Scan Form Constructor

On form initialisation, caller should provide,

- Network ID
- Subnet Mask
- Host Machine IP Address
- Form caller

With this information, on form's on Shown event Scan methods will be called on a Background worker class

```
public IList<Storage.activeNode> scan(IList<IPAddress> ip_range, DoWorkEventArgs e)
{
    this.activeNodes = new List<Storage.activeNode>();
    hostCount = ip_range.Count;
    int i = 1;
    foreach (var ip_ in ip_range)
    {
        if ((backgroundWorker.CancellationPending == true))
        {
            e.Cancel = true;
            break;
        }
        if (ip_.ToString() == this.yip.ToString())
        {
            Debug.WriteLine("Own IP. Skipping..");
            continue;
        }
        try
        {
            var hostname_ = NetScan.GetMachineNameFromIPAddress(ip_);
            if (NetScan.pingHost(ip_.ToString()))
            {
                Debug.WriteLine(hostname_ + " Available ");
                var mac_o = NetScan.getMacByIp(ip_);
                var mac_p = new PhysicalAddressProcessor(mac_o);
                var mac_ = PhysicalAddressProcessor.getMacString(mac_o);
                var vendor_ = mac_p.getNICVendor();
                activeNodes.Add(new Storage.activeNode
                {
                    ip = ip_,
                    hostname = hostname_,
                    mac = mac_o,
                    vendor = vendor_
                });
                scanGrid.Invoke((Action)(() => scanGrid.Rows.Add(new string[]
                { ip_.ToString(), mac_.ToString(), hostname_, vendor_, "Active" })));
            }
        }
        catch (NullReferenceException ex)
        {
            Debug.Write(ex.Message);
        }
        catch (Exception ex)
        {
            Debug.Write(ex.Message);
        }
        finally
        {
            progress = ((double)100 / (double)hostCount) * (double)i;
            backgroundWorker.ReportProgress((int)Math.Ceiling(progress));
            Debug.WriteLine(ip_.ToString() + " " + Math.Ceiling(progress).ToString("F2"));
            i++;
        }
    }

    return this.activeNodes;
}
```

Figure 9 - Scan Form Scan Function

This function updates the DATAGRIDVIEW as the it identifies active devices and update a list of active nodes to use on save function if user chose to save the results.

On save button's Click event, it initializes storage object with active device on parameters. Storage class is responsible for saving provided information in file form in a predefined folder

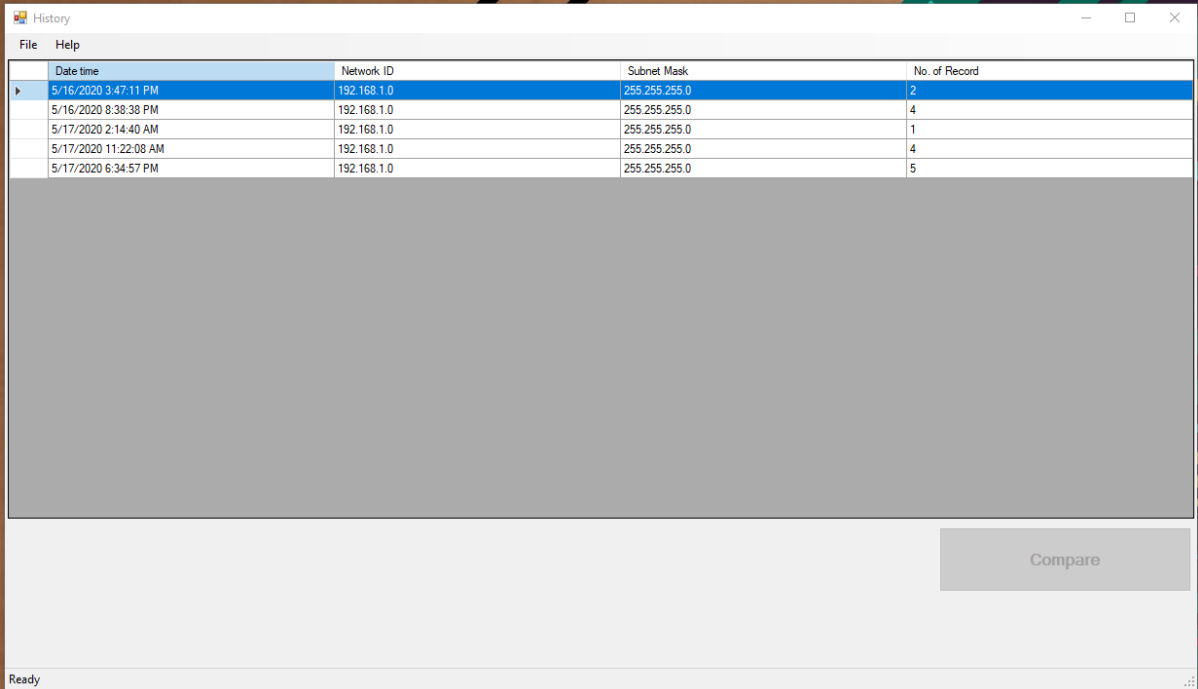

```

public Storage(string datetime, IList<activeNode> activeNodes, IPAddress netid, IPAddress mask)
{
    filename = datetime + "_" + netid.ToString() + "_" + mask.ToString() + '_' + RandomString(5) + ext;
    createDir();
    var myFile = File.Create(dirPath + "/" + filename);
    myFile.Close();
    TextWriter tw = new StreamWriter(dirPath + "/" + filename);
    foreach (var line in activeNodes)
    {
        tw.WriteLine(line.ip + "," + line.hostname + "," + new PhysicalAddressProcessor(line.mac).getMacString());
    }
    tw.Close();
}

```

Figure 10 - Storage Class (Save)

Snapshot history



The screenshot shows a window titled 'History' with a menu bar containing 'File' and 'Help'. Below the menu is a table with four columns: 'Date time', 'Network ID', 'Subnet Mask', and 'No. of Record'. The table contains five rows of data. The first row is highlighted with a blue background. Below the table is a large gray rectangular area, and at the bottom right is a button labeled 'Compare'. The status bar at the bottom left says 'Ready'.

Date time	Network ID	Subnet Mask	No. of Record
5/16/2020 3:47:11 PM	192.168.1.0	255.255.255.0	2
5/16/2020 8:38:38 PM	192.168.1.0	255.255.255.0	4
5/17/2020 2:14:40 AM	192.168.1.0	255.255.255.0	1
5/17/2020 11:22:08 AM	192.168.1.0	255.255.255.0	4
5/17/2020 6:34:57 PM	192.168.1.0	255.255.255.0	5

On this screen, users can view the previously obtained/saved scan results. Users can either view the results by double click on a specific recode of select 2 results and perform a comparison. Storage class we used to save snapshots previously is also responsible for obtaining information for this screen as well.

```

public Storage()
{
    createDir();
    Snapshotfiles = new List<Snapshotfile>();

    DirectoryInfo d = new DirectoryInfo(dirPath);
    FileInfo[] Files = d.GetFiles("*" + ext);
    foreach (FileInfo file in Files)
    {
        Debug.WriteLine(file);
        try
        {
            var date = file.Name.Split('_')[0].Split('-');
            var netid_ = file.Name.Split('_')[1];
            var mask_ = file.Name.Split('_')[2];
            var key_ = file.Name.Split('_')[3].Split('.')[0];
            var datetime = new DateTime(Convert.ToInt32(date[0]), Convert.ToInt32(date[1]), Convert.ToInt32(date[2]),
            Convert.ToInt32(date[3]), Convert.ToInt32(date[4]), Convert.ToInt32(date[5]));
            var filepath = dirPath + @"\" + file.Name;
            var lines = File.ReadLines(filepath);

            Snapshotfile snapshotfile = new Snapshotfile();
            snapshotfile.netid = IPAddress.Parse(netid_);
            snapshotfile.mask = IPAddress.Parse(mask_);
            snapshotfile.datetime = datetime;
            snapshotfile.recodeCount = lines.Count();
            snapshotfile.activeNodes = new List<activeNode>();
            snapshotfile.key = key_;

            foreach (var line in lines)
            {
                Debug.WriteLine(line);
                try
                {
                    var seg = line.Split(',');
                    var mac_ = new PhysicalAddressProcessor(seg[2]);
                    snapshotfile.activeNodes.Add(new activeNode
                    {
                        ip = IPAddress.Parse(seg[0]),
                        hostname = seg[1],
                        mac = mac_.physicalAddress
                    });
                }
                catch (Exception)
                {
                }
            }
            Snapshotfiles.Add(snapshotfile);
        }
        catch (Exception ex)
        {
            Debug.WriteLine(ex.Message);
            continue;
        }
    }
}

```

Figure 11 - Storage Get Files

Scan History - Network ID: 192.168.1.0 SubnetMask: 255.255.255.0 on: 5/17/2020 11:22:08 AM

IP Address	Physical Address	Hostname	NIC Vendor
192.168.1.1	EC:F0:FE:C2:BA:76	192.168.1.1	zte corporation
192.168.1.3	B8:27:EB:63:3F:B9	192.168.1.3	Raspberry Pi Foundation
192.168.1.25	A0:D3:C1:4B:09:A8	JefferdoBeast	Hewlett Packard
192.168.1.26	30:35:AD:85:1A:74	Jeewakas+iPhone.local	Apple, Inc.

Figure 12 - View single record

Compare Snapshots: 5/17/2020 6:34:57 PM 32Q5P vs 5/17/2020 11:22:08 AM NXH44

5/17/2020 6:34:57 PM (ID: 32Q5P)

IP Address	Physical Address	Hostname	NIC Vendor
192.168.1.1	EC:F0:FE:C2:BA:76	n/a	zte corporation
192.168.1.3	B8:27:EB:63:3F:B9	raspberrypi.local	Raspberry Pi Foundation
192.168.1.26	30:35:AD:85:1A:74	n/a	Apple, Inc.
192.168.1.27	E4:98:D1:78:2F:10	n/a	Microsoft Mobile Oy
192.168.1.60	D0:C5:D3:4F:6F:2D	JefferdoROG.local	AzureWave Technology I...

5/17/2020 11:22:08 AM (ID: NXH44)

IP Address	Physical Address	Hostname	NIC Vendor
192.168.1.1	EC:F0:FE:C2:BA:76	192.168.1.1	zte corporation
192.168.1.3	B8:27:EB:63:3F:B9	192.168.1.3	Raspberry Pi Foundation
192.168.1.25	A0:D3:C1:4B:09:A8	JefferdoBeast	Hewlett Packard
192.168.1.26	30:35:AD:85:1A:74	Jeewakas+iPhone.local	Apple, Inc.

Figure 13 - Compare 2 records

On comparison form, on row click event it shows matching record from the other file. The perform this action on click event look for similar mac addresses from two files.

Challengers

- Firewall rule may affect the results of the Software
- To resolve hostname of each device the program depends on the DNS server of the network. Therefore, availability of hostname is may vary on different network. To

mitigate this issue, I have included NIC manufacture identifier. With it we can get somewhat better idea about the device.

- Most of the devices on power save mode does not replay for ICMP (PING) protocol requests. Therefore, program may skip those devices even though they are actively connected to the network. Especially mobile devices.

Other Similar tools

IP Address Tracker

By far the most powerful tool on the list of free clients, SolarWinds IP Address Tracker is a standalone solution, available for free download, that works on its own but is further enhanced by the SolarWinds IPAM suite when integrated. This makes it an excellent first step if you are considering a premium option but looking for a fully functional address tracker in the meantime.

Angry IP Scanner (Free)

Widely hailed as one of the first and most popular free IP address scanners, Angry IP Scanner is open-source software, deployable across operating systems. Windows, Linux, and Mac OS X users will find this tool handy for its non-existent price tag.

NETDISCOVER

NETDISCOVER is an active/passive ARP reconnaissance tool, initially developed to gain information about wireless networks without DHCP servers in wardriving scenarios. It can also be used on switched networks. Built on top of LIBNET and LIBPCAP, it can passively detect online hosts or search for them by sending ARP requests.

Furthermore, it can be used to inspect your network's ARP traffic, or find network addresses using auto scan mode, which will scan for common local networks.