Information Security

EINDVERSLAG

PACKETANALYZER & INJECTOR MET C#

Ziggy Streulens

Jasper Van Gestel

3EA1

Voorwoord

Dit project werd gemaakt in opdracht van de heer Tim Dams, voor het vak 'Information Security'.

Tijdens het project hebben we ook dit jaar weer veel bijgeleerd. Als we zeggen 'veel' dan heeft dit vooral betrekkingen tot het gebied netwerking met c#. Maar ook 'threading' was een enorm pijnpunt om optimaal te krijgen tijdens de vele uurtjes die aan het project zijn verslonden.

In dit verslag volgt een kort overzicht van alle belangrijke code, alsook de gebruikte libraries en technieken die het project maken tot wat het is. Mits wat meer tijd (en wat minder projecten) had dit natuurlijk nog beter kunnen zijn dan het momenteel is. Al mogen we zeker niet ontevreden zijn na al de tegenvallers die we onderweg zijn tegengekomen.

Als laatste nog een bedankje aan de heer Tim Dams zelf, waar we altijd terecht konden als we met vragen zaten.

Inhoud

Inleiding - Opdracht

Je schrijft een eigen applicatie die kan dienen om eenvoudige pakketinspectie te doen (**). Vervolgens zorg je ervoor dat pakketten kunnen aangepast worden (of van scratch gemaakt worden) die dan terug in het netwerk kunnen worden geïnjecteerd (****) . Doel van de tool is om een zeer elementaire 'Wireshark' te maken die eerstejaars in de labo's netwerkcomponenten zouden kunnen gebruiken voor kleine experimenten zonder overdonderd te worden door de uitgebreide eigenschappen van 'Wireshark'.

De applicatie is een WPF applicatie met een verzorgde front-end

Chapter 1: De Hoofdcode (MainWindow.xaml.cs)

De code is opgedeeld in 3 delen, elk onderverdeeld in hun belangrijke sub-delen.

Hieronder een kort overzicht via print-screen:

```
13   namespace PacketSniffer2
14   {
15       // Interaction logic for MainWindow.xaml
         4 references | Viraptorus, 46 minutes ago | 2 authors, 28 changes
16       public partial class MainWindow : MetroWindow
17       {
18           // All variables
19           Variables
55
56           // All initialization methods / definitions
57           Initialization
151
152          // All methods that don't handle initialization
153          #region Methods
154
155          Sniffing Methods
437
438          Injecting Methods
635
636          #endregion
```

Variabelen

Zoals de titel het ongetwijfeld al doet zeggen. Simpelweg de variabelen (per type) onder elkaar staan gedefinieerd. Omdat variabelen meestal genoeg zeggen, ga ik hier niet veel verder over uitweiden.

Initialisatie

De initialisatie methoden stellen het programma in voor eerste gebruik en behandelen de methoden omtrent het halen van de netwerkadapters van uw lokale machine. Hier is ook nog niet veel code noemenswaardig. Al zal de belangrijkste methode hier: 'GetSelectedDevice()' zijn (screenshot)

```
138      //Get the adaptor the user wants to use
         1 reference | Viraptorus, 7 days ago | 1 author, 5 changes
139      private void GetSelectedDevice()
140      {
141          for (int i = 0; i != listAllDevices.Count; ++i)
142          {
143              //Get devicelist
144              LivePacketDevice device = listAllDevices[i];
145              if (DeviceListBox.SelectedItem.ToString() != null)
146              {
147                  //Check if name of device is in the devicelist
148                  if (DeviceListBox.SelectedItem.ToString().Contains(device.Name))
149                  {
150                      pSelectedDevice = device;
151                      PacketList.ItemsSource = ocPackets;
152                  }
153              }
154              else
155              {
156                  //Tell user if he did'nt pick a device
157                  MessageBox.Show("Select a device and press 'capture' to start.");
158              }
159          }
160      }
```

Methoden

Here is where the magic happens.

Hierin onderscheiden we 2 delen code zoals al reeds gezien:

- Code die de 'Sniffing' behandelt
- Code die de 'Injecting' behandelt

Oorspronkelijk was er ook een stuk code dat het 'Editing' behandelde, maar aangezien het editing deel praktisch gezien net hetzelfde doet als het injecteren van pakketten hebben we dit stuk code gewoon geïntegreerd bij het 'Injecting' gedeelte.

Sniffing

Het grootste deel van de code vinden we terug bij onze sniffer. De packethandler moet zowat de grootste methode en van het project zijn. De methode creëert namelijk een nieuw virtueel pakket van het originele netwerkpakket dat wordt gevangen via de netwerkadapter

Deze methode maakt gebruik van een zelfgemaakte klasse (deels herbruikt van vorig jaar). Deze klasse overlopen we wat verder in dit verslag.

De packethandler gaat dus de data uit het originele pakket binden aan de datavelden van het nieuwe virtuele pakket.

```
168         //Create a virtual packet from the original packet
            1 reference | Viraptorus, 1 day ago | 1 author, 8 changes
169         private void PacketHandler(Packet packet)
170         {
171             //Create virtual packet
172             var ArrivedPacket = new PacketAPI();
173
174             //Bind original packet data to virtual packet
175             ArrivedPacket.Timestamp = packet.Timestamp.ToString();
176             ArrivedPacket.MacSource = packet.Ethernet.Source.ToString();
177             ArrivedPacket.MacDestination = packet.Ethernet.Destination.ToString();
178             ArrivedPacket.IpSource = packet.Ethernet.IpV4.Source.ToString();
179             ArrivedPacket.IpDestination = packet.Ethernet.IpV4.Destination.ToString();
180             ArrivedPacket.Length = packet.Length;
181             ArrivedPacket.Ttl = packet.Ethernet.IpV4.Ttl;
182             ArrivedPacket.Id = packet.Ethernet.IpV4.Identification;
```

Het gaat ook (gebaseerd op meerdere criteria) zoeken naar welk protocol er juist gebruikt wordt. Deze methode kon een pak beter worden, maar wegens tijdgebrek is ze gebleven zoals ze er momenteel uitziet.

```
184             //Find out what protocol the original packet is using and bind as well
185             //(This part is not optimal yet)
186             if (packet.Ethernet.EtherType == EthernetType.IpV4)
187             {
188                 ArrivedPacket.Ipv4 = true;
189                 ArrivedPacket.Protocol = "IPV4";
190                 if (packet.Ethernet.IpV4.Icmp != null && packet.Ethernet.IpV4.Protocol.ToString()
191                     == "InternetControlMessageProtocol")
192                 {
193                     ArrivedPacket.Icmp = true;
194                     ArrivedPacket.Protocol = "ICMP";
195                 }
196                 else
197                 {
198                     ArrivedPacket.Icmp = false;
199                 }
200
201                 if (packet.Ethernet.IpV4.Udp != null && packet.Ethernet.IpV4.Protocol.ToString()
202                     == "Udp")
203                 {
204                     ArrivedPacket.Udp = true;
205                     ArrivedPacket.Protocol = "UDP";
206                     ArrivedPacket.PortSource = packet.Ethernet.IpV4.Udp.SourcePort;
207                     ArrivedPacket.PortDestination = packet.Ethernet.IpV4.Udp.DestinationPort;
208                     if (ArrivedPacket.PortDestination == 53 || ArrivedPacket.PortDestination > 1023
```

```
209                     || ArrivedPacket.PortSource == 53 || ArrivedPacket.PortSource > 1023)
210                 {
211                     ArrivedPacket.Dns = true;
212                     ArrivedPacket.Protocol = "DNS";
213                 }
214                 else
215                 {
216                     ArrivedPacket.Dns = false;
217                 }
218             }
219             else
220             {
221                 ArrivedPacket.Udp = false;
222             }
223
224             if (packet.Ethernet.IpV4.Tcp != null && packet.Ethernet.IpV4.Protocol.ToString()
225                 == "Tcp")
226             {
227                 ArrivedPacket.Tcp = true;
228                 ArrivedPacket.Protocol = "TCP";
229                 ArrivedPacket.PortSource = packet.Ethernet.IpV4.Tcp.SourcePort;
230                 ArrivedPacket.PortDestination = packet.Ethernet.IpV4.Tcp.DestinationPort;
```

Op het einde gaat de methode ook gebruik maken van de ingebouwde filter en zo de juiste pakketten door te geven.

```
251             //Check the filter to see which packets are allowed to be shown
252             if (bIcmpCheck && ArrivedPacket.Icmp)
253                 Dispatcher.Invoke(new UpdateTextCallback(UpdatePacketText), ArrivedPacket);
254             else if (bUdpCheck && ArrivedPacket.Udp)
255                 Dispatcher.Invoke(new UpdateTextCallback(UpdatePacketText), ArrivedPacket);
256             else if (bTcpCheck && ArrivedPacket.Tcp)
257                 Dispatcher.Invoke(new UpdateTextCallback(UpdatePacketText), ArrivedPacket);
258             else if (bDnsCheck && ArrivedPacket.Dns)
259                 Dispatcher.Invoke(new UpdateTextCallback(UpdatePacketText), ArrivedPacket);
260             else if (bHttpCheck && ArrivedPacket.Http)
261                 Dispatcher.Invoke(new UpdateTextCallback(UpdatePacketText), ArrivedPacket);
262             else if (bIPV4Check && ArrivedPacket.Ipv4)
263                 Dispatcher.Invoke(new UpdateTextCallback(UpdatePacketText), ArrivedPacket);
```

De ‘StartThreadAnalyze()’ methode behandelt het grootste deel van de threading. Zo wordt de nieuwe thread (in het project ‘tCapture’ genoemd) opgestart en krijgt de thread het geselecteerde apparaat mee als argument om in de thread te kunnen gebruiken. (krijgt het argument mee via het ‘delegate’ commando) Klein detail, wanneer we de ‘PacketCommunicator’ gebruiken starten we deze steeds in ‘Promiscuous’ mode. Deze modus stelt ons in staat om ook pakketten die niet voor onze lokale machine bestemd zijn, toch te kunnen opvangen. Dit start dus het eigenlijke sniffen naar pakketten.

```
266         //Start the analyze thread that will handle incoming packets
            1 reference | Viraptorus, 7 days ago | 1 author, 3 changes
267         private void StartThreadAnalyze()
268         {
269             //Pass the selected device on to the new thread
270             Dispatcher.Invoke((ThreadStart)delegate { pSelectedDevice.ToString(); },
271                 DispatcherPriority.Normal, null);
272
273             if (pSelectedDevice.ToString() != null)
274             {
275                 while (true)
276                 {
277                     ePause.WaitOne(Timeout.Infinite);
278                     if (eShutdown.WaitOne(0))
279                         break;
280
281                     //bCapture = true;
282
283                     //Start capturing with selected device
284                     using (PacketCommunicator communicator = pSelectedDevice.Open(65536,
285                         PacketDeviceOpenAttributes.Promiscuous, 1000))
286                     {
287                         communicator.ReceivePackets(0, PacketHandler);
288                     }
289                     //bCapture = false;
290                 }
291             }
292             else
293                 MessageBox.Show("Please choose a networkadapter");
```

Verder bevat de 'Sniffing methoden' regio nog de booleans en methode voor de filter, de functies achter de verschillende knoppen en de methoden achter de verschillende lijsten om de juiste data weer te geven op het juiste moment.

Injecting

De 'Injecting' code bevat slechts 2 (weliswaar grote) methoden

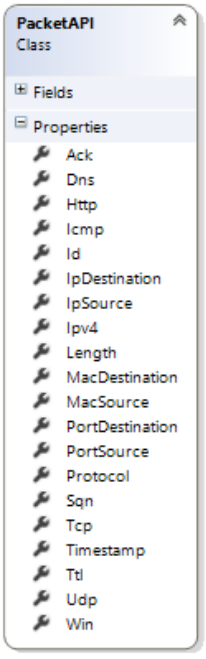
De code achter de verzendknop bepaalt welk type pakket moet worden opgebouwd in de 'PacketCommunicator' en zo in het netwerk verstuurd zal worden. Via de switchcase wordt het type bepaalt. Er wordt ook een extra controle gedaan opdat de noodzakelijke velden wel degelijk ingevuld zijn (Het had eventueel nog verbeterd kunnen worden met een template voor de invulvelden, maar daar was helaas (alweer) niet genoeg tijd meer voor). Het voorbeeld voor ICMP is ook bij de screenshot weergegeven.

```
495         //Send button is pressed
            1 reference | Viraptorus, 4 hours ago | 2 authors, 19 changes
496         private void btnSendPacket_Click(object sender, RoutedEventArgs e)
497         {
498             //Loop that defines how many times the packet should be resent
499             for (int i = 0; i <= Convert.ToInt16(xTimes.Text.ToString()); i++)
500             {
501                 // Open the output device
502                 using (pCommunicator = pSelectedDevice.Open(100, PacketDeviceOpenAttributes.Promiscuous, 1000))
503                 {
504                     //Get the protocoltype and go through the switchcase in order to build the right packet
505                     int stringProtocol = ProtType.SelectedIndex;
506                     switch (stringProtocol)
507                     {
508                         //If its an ICMP packet do this
509                         case 1:
510                             if (MACsrc.Text != "" && MACdst.Text != "" && IPsrc.Text != "" && IPdst.Text != ""
511                                 && IpId.Text != "" && TTL.Text != "" && Identifier.Text != "" && SQN.Text != "")
512                             {
513                                 pBuildIcmpPacket = new ICMPSendPacket(MACsrc.Text, MACdst.Text, IPsrc.Text,
514                                 IPdst.Text, IpId.Text, TTL.Text, Identifier.Text, SQN.Text);
515                                 pCommunicator.SendPacket(pBuildIcmpPacket.GetBuilder());
516                             }
517                             else
518                             {
519                                 MessageBox.Show("Please fill in all required (open) fields");
520                             }
521                             break;
```

De andere methode die aanwezig is behandelt de visuele weergave om de juiste invulvelden open te stellen. Zo is er minder kans op fouten.

Chapter 2: Klassen

Er zijn verschillende klassen gemaakt om het programma vlotter te laten werken en om het project een beetje object-oriënted te houden. Alle klassen worden gebruikt om pakketten op te stellen. Zo is er een klasse die gebruikt wordt als de virtuele voorstelling van echte packeten (PacketAPI) en een klasse die de basis vormt van al de overige klassen (via overerving) om effectief echte netwerkpakketten te creëren (BaseSendPacket). Deze twee gaan we dan ook overlopen.



De PacketAPI klasse bevat alle 'properties' die nodig zijn om de juiste pakketten virtueel weer te geven. Via deze klasse worden de originele pakketten in de 'ObservableCollection' doorgegeven, die op zijn beurt alle 'properties' visueel zal weergeven bij de gebruiker.

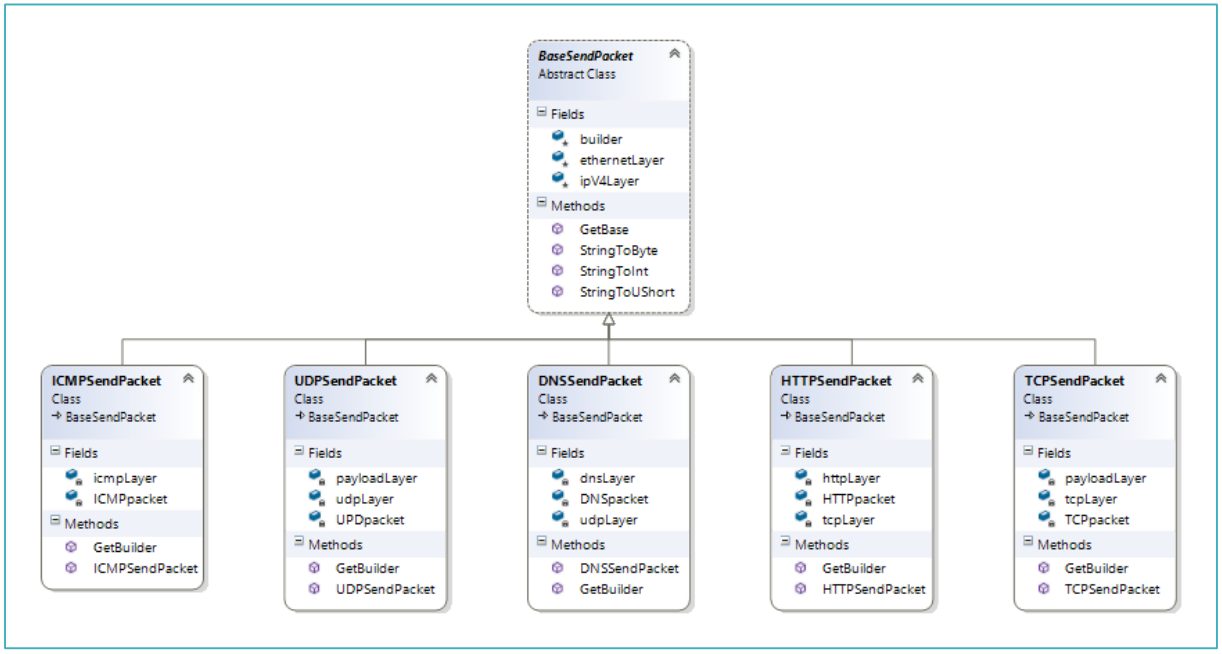
De 'BaseSendPacket' klasse definieert de basis-layers voor alle internetpakketten. Zo zijn er de ethernet en IpV4 laag die al reeds gecreëerd worden in deze basisklasse. Ook de builder wordt hier al gedefinieerd. De builder zal uiteindelijk zorgen dat het pakket juist wordt samengesteld met de nodige netwerklagen.

```
5 references | Viraptorus, 19 days ago | 1 author, 6 changes
abstract class BaseSendPacket
{
    //Define layers
    protected EthernetLayer ethernetLayer;
    protected IpV4Layer ipV4Layer;
    protected PacketBuilder builder;

    //Create a base packet to use as parent for the other classes
    5 references | Viraptorus, 19 days ago | 1 author, 1 change
    public virtual void GetBase(string MACsrc, string MACdst, string IPsrc, string IPdst,
        string IpId, string TTL)
    {
        // Ethernet Layer
        ethernetLayer = new EthernetLayer
        {
            Source = new MacAddress(MACsrc),
            Destination = new MacAddress(MACdst),
            // Set ethernet type
            EtherType = EthernetType.None
        };

        // IpV4 Layer
        ipV4Layer = new IpV4Layer
        {
            Source = new IpV4Address(IPsrc),
            CurrentDestination = new IpV4Address(IPdst),
            Fragmentation = IpV4Fragmentation.None,
            HeaderChecksum = null, // Will be filled automatically.
            Identification = StringToUShort(IpId),
            Options = IpV4Options.None,
            Protocol = null, // Will be filled automatically.
            Ttl = StringToByte(TTL),
            TypeOfService = 0,
        };
```

De andere 5 klassen zijn overervende klassen. Deze stellen per protocol het juiste pakket samen om dan het netwerk in te sturen en zijn dan ook een aanvulling op de basisklasse 'BaseSendPacket'. Hieronder zien we een screenshot van het klasse diagram waar we een beter zicht krijgen op deze 5 klassen



Chapter 3: Front-end (MainWindow.xaml)

Er zijn 2 grote elementen in het xaml-bestand.

- Sniffing
- Injecting

Algemeen

Het programma bestaat dus uit 2 'tabs'. Elke tab heeft 'een navigatiebalk' Dit is de blauwe balk aan de linkerkant. De balk bevat de enkele interactieve knoppen per tab. De rest van het gebied zijn listboxen die de informatie duidelijk en gestructureerd moeten weergeven.

Sniffing

In de 'sniffing' tab hebben we in onze navigatiebalk 3 knoppen:

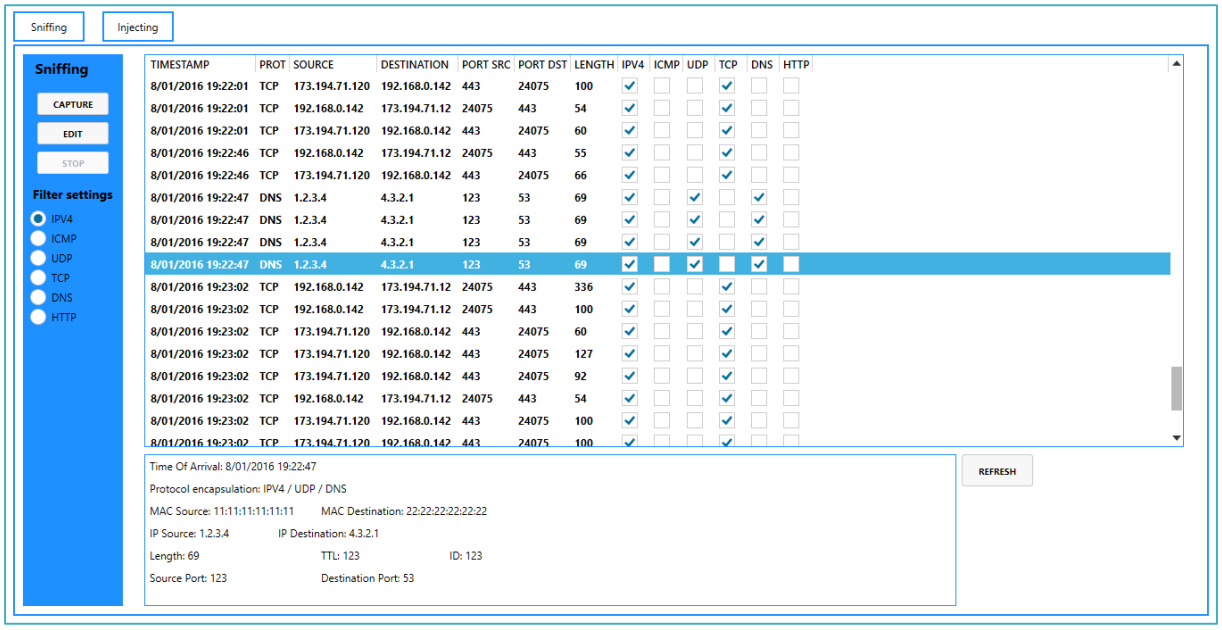
- Capture
- Edit
- Stop

Deze dienen respectievelijk om te starten met pakketten op te vangen, pakketten aan te passen en te stoppen pakketten op te vangen. Meer info over de werking vinden we in het hoofdstuk 'werking'.

Nog in de werkbalk vinden we een pakket-filter die ons toelaat om enkel pakketten met een bepaald protocol op te vangen.

Initieel zien we onderaan een lijst met alle gevonden netwerkadapters waarmee men pakketten kan opvangen en bovenaan een lijst die extra informatie verschaft over deze adapters. We zouden immers niet willen dat we een nutteloze of verkeerde adapter gebruiken.

Wanneer een adapter gekozen is en er op 'capture' gedrukt wordt maken deze 2 lijsten plaats voor 2 andere lijsten. Bovenaan hebben we dan de lijst met opgevangen pakketten waarin we de basis informatie en het tijdstip van aankomst kunnen zien. De onderste lijst zorgt dan, wanneer we een pakket selecteren, dat we meer geavanceerde informatie krijgen over het betreffende pakket.

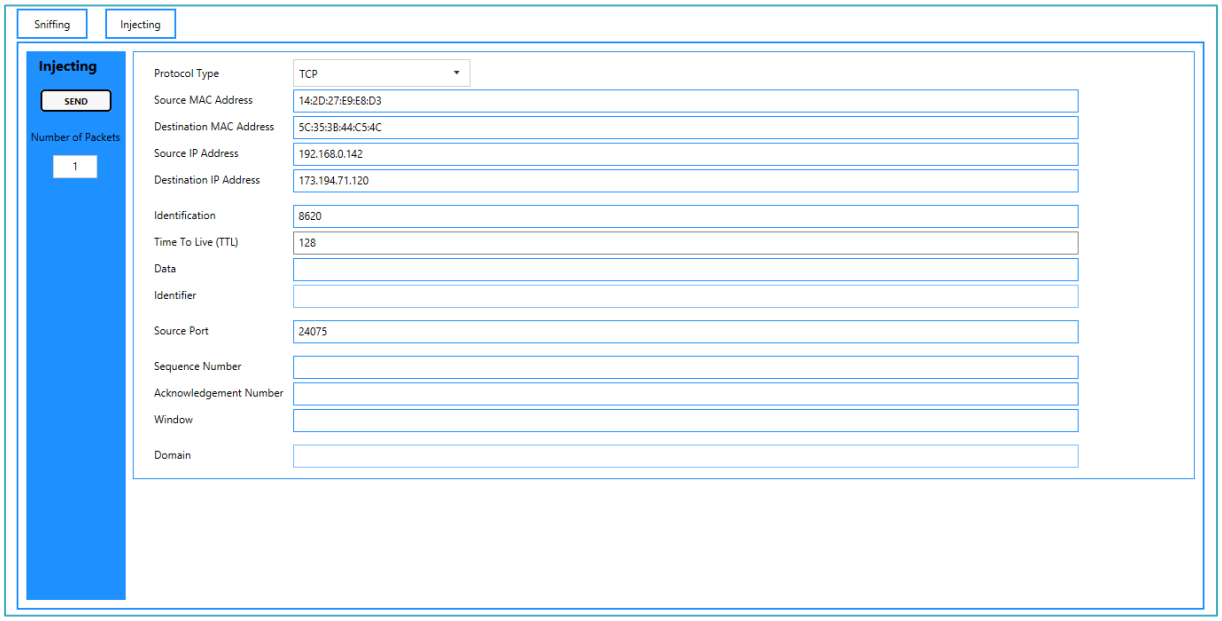


Editing

Wanneer men de 'editing' tab opent kan men (als een adapter is geselecteerd) hier een eigen pakket bouwen om het netwerk in te sturen.

In de navigatiebalk vinden we de 'send' knop terug. Deze zal het pakket versturen mits alle velden zijn ingevuld. Daaronder is er een veld voorzien waarin de gebruiker kan bepalen hoeveel keer hij het pakketje wil versturen.

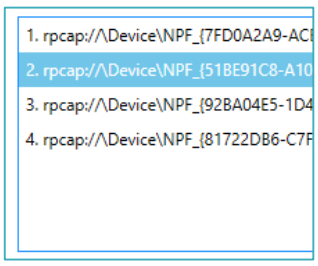
Wanneer er bij de 'sniffing' tab op de 'edit' knop was gedrukt zal er al een groot deel van de informatie ingevuld zijn en is het zelfs mogelijk deze informatie aan te passen en opnieuw los te laten in het netwerk.



Chapter 4: Werking

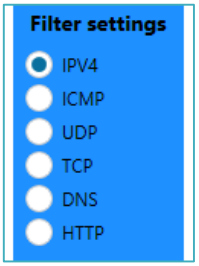
Sniffing tab

Het is de bedoeling om na de opstart van het programma in de tab 'sniffing' naar de onderste lijst te gaan om hier eerst uw adapter te selecteren die gebruikt wordt tijdens deze sessie (afbeelding rechts). Je kan meer info over de adapter terug vinden in de lijst erboven. Wanneer er dan op 'capture' gedrukt wordt zal het programma da adapter aanspreken en de pakketten die rondvliegen beginnen op te vangen. Deze pakketten worden dan in de lijst daarboven weergegeven.



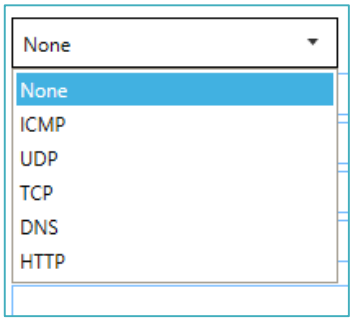
De 'capture'-knop start eveneens een aparte thread die de pakkethandling doet (zoals eerder vermeld). De thread wordt onderbroken door de 'stop' knop of wanneer het programma wordt afgesloten.

De 'edit' knop zal alle gegevens van een geselecteerd pakket overzetten naar de injecting-tab zodat de gebruiker dit pakket aan kan passen.



Tot slot is er de filter. Deze zorgt ervoor dat je enkel pakketten in de lijst terecht ziet komen die overeenkomen met dat specifieke protocol. Het zal al de andere pakketten ook opvangen, maar zal deze niet tonen. (zie afbeelding rechts)

Editing tab



Men kan in deze tab pakketten creëren. Dit doen we simpelweg door eerst een protocol te kiezen dat we willen gebruiken voor ons pakket. Het programma zal dan de velden die ingevuld moeten worden openstellen voor bewerking. (afbeelding rechts)

Wanneer er echter op de 'edit' knop gedrukt is bij de vorige tab zal er in de 'editing' tab al een groot deel informatie van het pakketje ingevuld zijn. Deze is hier dan ook manipuleerbaar.

Zorg er steeds voor dat alle noodzakelijke velden ingevuld zijn, anders zal je het pakketje niet kunnen maken. Wanneer dan op de 'send' knop gedrukt wordt zal het pakketje gebouwd en verzonden worden.

Er is ook een tekst vak waarin men kan typen hoeveel keer men het pakketje zou willen genereren.

Chapter 5: Ontwikkeling, libraries en problemen

Geen project zonder de nodige ergernissen en problemen. Hier dus een korte oplossing van de ontwikkelingen die uiteindelijk niet in het project geraakt zijn, de problemen die zich zowat overal hebben voorgedaan en de verschillende libraries die de revue passeerden

Ontwikkeling en libraries

Bij de start van het project was het ons idee om met een betere library te werken dan de reeds gebruikte library van vorig jaar. Zo kwamen we Sharppcap eerst tegen. Deze bleek, na heel wat testen, niet geschikt voor onze doeleinden.

Een tijdje later kwamen we bij een zeer uitgebreide, en geavanceerde library terecht: de eEx Network Library. Na een hoop tests met deze library kwamen we tot de conclusie dat deze iets te ingewikkeld en misschien zelfs te geavanceerd was.

Ondertussen zaten we al een maand verder. Er moest dus dringend een oplossing gevonden worden. We keerden ons dus naar de Pcap.net library die ook vorig jaar werd gebruikt. Na heel wat zoeken en testen hadden we een goed basisprogramma om op verder te bouwen. We baseerden ons op talloze tutorial 's, voorbeelden en projecten om uiteindelijk aan deze werkende versie te komen.

Problemen

Buiten het probleem met de libraries dat we eerst hadden, zijn er natuurlijk ook nog een hoop andere problemen geweest.

Typfouten in de invulvelden maakten enorme problemen. Daardoor is er een kleine beveiliging opgezet tegen het openlaten van velden en het invullen van de juiste velden.

De 'Refresh' knop geeft nog problemen, dit heeft vooral te maken met de 'slection_changed' methoden. Deze fout stond al even op het lijstje om te herstellen.

De threading is ook nog steeds niet optimaal. Wanneer men lang aan het capturen is, begint het programma soms te haperen.

De ACK, SQN, en WIN velden konden niet uit de originele pakketten worden gehaald, waardoor de velden een beetje nutteloos worden.

Oorspronkelijk was er ook een IPV4 pakket dat kon worden opgebouwd. Het pakket kon enkel verstuurt worden via UDP waardoor automatisch DNS wordt aangesproken (→ is dus identiek aan DNS pakket)

En als laatste is de identificatie van de protocollen niet echt optimaal. Zo zal er nog wel de kans bestaan dat er het verkeerde protocol toegewezen wordt. (Niet genoeg tijd om alle netwerkprotocollen deftig te bekijken)

Chapter 6: Bronnen

Informatief

- https://msdn.microsoft.com/en-us/library/bew39x2a(v=vs.110).aspx?cs-save-lang=1&cs-lang=csharp#code-snippet-1
- https://www.youtube.com/watch?v=QWY5pDlrZp0
- https://pcapdotnet.codeplex.com/wikipage?title=Pcap.Net%20Tutorial&referringTitle=Pcap.Net%20Tutorial%20-%20Obtaining%20the%20device%20list
- https://github.com/chmorgan/sharppcap
- http://www.tamirgal.com/blog/page/SharpPcap.aspx

Zeer nuttig:

- http://www.codeproject.com/Articles/17031/A-Network-Sniffer-in-C (link van 2008…)
- https://msdn.microsoft.com/en-us/library/system.net.sockets(v=vs.110).aspx
- https://github.com/PcapDotNet/Pcap.Net/wiki (wiki pcap)

ICMP implementatie:

- http://www.winsocketdotnetworkprogramming.com/clientserversocketnetworkcommunication8n.html

Nieuwe Library: eEx Network Library:

- http://www.eex-dev.net/fileadmin/user_upload/apidoc/NetworkLibrary/index.html
- https://eex.codeplex.com/
- http://network.eex-dev.net/index.php?id=64&tx_drwiki_pi1%5Bkeyword%5D=Your%20first%20Router

Pcap.Net:

- https://pcapdotnet.codeplex.com/wikipage?title=Pcap.Net%20Tutorial%20-%20Sending%20Packets