## Background

My project’s main aim is to visualise and demonstrate network packet ‘sniffing’ in the context of an encrypted chat room. I initially did some preliminary work with sockets and an extended packet analysis program however, later I decided to move my project towards the educational context of a packet capturing & reading program. The second part of my program actively observes the effectiveness of packet sniffing whilst in the context of encryption. It is a message application that allows the user to enable and disable encryption whilst communicating. The packet sniffer will focus on this chat and therefore demonstrate the effectiveness of encryption and packet sniffing when they coincide.

## Target Audience

The project is fundamentally an educational one and therefore my aim was to create an application with which members across the educational community could use to support their studies. Therefore my project’s target audience are interested in network programming, cryptography and networking in general. Initially, before efforts were made to develop this application, I discussed how I should solve this problem with my Computer Science teacher who is a part of my target audience. I asked him questions, a few of which were: “*How should I visualise network packet sniffing?*”, “*What kind of technology would be the most effective to capture packets?*”, “*How can I allow my target audience to interact with my application?*”, “*What should the main functions of this application be?*” and “*How can I maximise the educational aspect of my application?*”.

After some discussion, a few points were raised. A number of issues were first made clear that I had to create solutions to, alongside numerous main functions and points that I should endeavour to include within my application, of which I will discuss in more detail throughout my research and documentation.

Significantly, an idea with combined thought from myself, Mr Quantrill and a Computing professional was that I should demonstrate the technology and concepts of packet capturing via the programming of an encrypted messaging system. It should demonstrate packet sniffing in both the context of encryption and no encryption. This was a very important concept that met most of my earlier questions.

## Research

My research started by analysing the industry standard packet sniffer ‘WireShark’. I downloaded the application and started reading through the manual and the documentation. After watching and reading numerous videos and papers and websites explaining the technology of packet sniffing, I wanted to first give myself the opportunity to program a socket connection in Java as that would form the base for my application.

## Section 1: Network Sniffer

## Language Suitability

Below is a simplified diagram of how my initial program worked:



A massive issue I encountered whilst socket programming in Java, is that the sockets were not picking up *all* of the network data passing through a port. After doing some research into socket implementation in Java, I concluded that it is impossible to open what is known as *promiscuous* sockets in Java. Java sockets are end-to-end implementations, they cannot listen on the network port for *all* traffic. However, Python allows entirely promiscuous sockets to be opened by use of the Windows winsock2.h WSAIoctl function (written in C).

Here is where I made the decision to change my language to Python, as it best suited this project.

## The OSI Model and Ethernet Frame

All networks reflect a layered, logical progression that is based on the 7-layered OSI model. I researched heavily into this layered architecture as it helps identify the specific area in which I will be capturing my packets from. I evidently won’t be capturing packets at the Physical layer as it’s entirely abstracted. The most important layer for me in this project is the Data Link layer. It contains a **reliable** **transmission** of **data frames** between two nodes.



The OSI Model

From the Wireshark documentation, I learned that network packets exist in the Ethernet Frame which is a data link layer protocol data unit, in other words, a data unit on an Ethernet link transports Ethernet frames as it’s payload data. I thought the use of a diagram really helped me understand it.



The ethernet frame looks like this.

* The preamble is a 56 bit pattern of alternating bits, ending in 11, allowing devices on a network to understand that an ethernet is about to be received.
* The recipient MAC & sender MAC are addresses for NICs to identify devices.
* The Type is referring to which Ethernet frame which is being used, Ethernet II is the most common in use today as it's often used directly as the Internet Protocol. (Ipv4 or Ipv6).
* **The data section is the most important to us, it houses IP, ARP etc data as payload data. It’s an IP packet, it contains the data we want to capture and manipulate. It must contain between 46 and 1500 bytes of information.**
* The Frame Check Sequence is a cyclic redundancy check (CRC) that allows for the detection of corrupted data in the entirety of the frame (bar the preamble).

This level of information is really important with the designing of my algorithms as they must be able to manipulate the byte stream in correspondence with how an Ethernet frame is constructed in order to properly intercept and read the data.

## Wireshark Analysis



Wireshark Program Diagram

Wireshark is the industry standard network packet sniffer and so researching how it works is very beneficial to the development of my application.

**GUI**

Handles all user input and output.

**Core**

Main code that holds and connects the other blocks.

**Epan**

Enhanced packet Analyser - the engine that analyses each packet, it also provides the

following APIs:

Protocol Tree (dissection information for an individual packet)

Dissectors (Various protocol dissectors)

Dissector Plugins (Support for implementation of dissectors individually)

Display Filters (Display filter engine)

**Wiretap**

Wiretap library is used to read and write capture files in libpcap.

**Capture**

Interface to the capture engine.

**Dumpcap**

Capture engine - this is the only function block that executes with elevated privileges.

**Libpcap**

External libraries that provide packet capture and filtering support across many

Platforms.

The main components that will make up my application are a custom programming of the Core, GUI, Capture and parts of the Epan.

## Professional Feedback

I decided to seek professional help from both a Digital Forensics law enforcement officer, and a private professional Digital Forensics investigator. The following is a transcript of the conversation I had with both, under aliases.

**Smorgan 22/08/2020**

*In general I would study the osi model and look at how wireshark identifies a given packet as an IP, HTTP, etc.*

*I would study encapsulation of packets and so forth*

**Kr Today at 11:15**

***@bailey*** *You'll most likely want to reconstruct TCP to be able to follow the many protocols built on top of it, otherwise analyzing each packet independently is very limited.*

*There are some quick & dirty workarounds of course, like most HTTP requests fit within one TCP packet but technically one could (and malware would) send "GE", "T / H" and "TTP/1.1" as separate packets for example instead of the usual "GET / HTTP/1.1", which would work perfectly fine on the receiving end but a firewall/IDS can be fooled by this.*

*You can see what I mean in Wireshark with the right click > Follow TCP Stream feature, or with tcpflow some.pcap on the CLI, and with LibNIDS as a library for C; I can't seem to find Java bindings for it but hopefully there is an equivalent.*

**bailey Today at 17:10**

*I've looked at the OSI model and done some prerequisite research into how wireshark works, the ethernet frame, how to identify packet structures and stuff, I'm trying to program this using vanilla libs where possible*

***@Kr*** *I really appreciate the feedback, though, what do you mean by "reconstruct TCP"?*

**Kr Today at 17:35**

***@bailey*** *Basically all protocols fall into one of two categories: datagram-based or stream-based; the former means that a single packet contains everything it needs (since the MTU is usually 1500 bytes there is plenty of room to fit things like a DHCP discovery, a DNS query, an ICMP ping, etc) while the latter allows the payload to stretch across packets (e.g. most images nowadays are more than a kilobyte, if not megabytes, so they will have to be transported piece by piece).*

**Kr Today at 17:45**

***@bailey*** *This introduces quite a few additional things to be considered: what if the pieces arrive in the wrong order, what if one is lost, what if they are being sent more quickly that the network or receiver can handle, etc. TCP addresses all of that.*

*A good approach to understanding this is to study the differences between UDP, DCCP, TCP and SCTP.*

*Basically:*

*1. All UDP does is put you in touch with a particular port, it doesn't handle any of the considerations above and its checksumming feature is pretty much useless. But this is great when you do not want flow control for example, which is the case of VPNs, hence the saying that "TCP over TCP is a bad idea".*

*2. DCCP is UDP plus flow control: it won't retransmit anything lost but at least it will try to regulate the speed, which is great for video streams for example: doesn't matter if you missed part of an image, you mostly care that it keeps going on instead of getting stuck (which is what TCP would do until it managed to retransmit that piece of image you don't care about anymore) and that the quality adjusts to make the most of your connection speed.*

*3. TCP as explained above.*

*4. SCTP is TCP plus the capability to handle multiple streams within one connection, which is especially interesting when you want to do multiple (see MPTCP) but that gets complicated quickly and is not very common.*

**bailey Today at 17:50**

*I see, so it would be a good idea for me to try to handle both datagram and stream based protocols within my application*

*in order to make my application efficient and actually functioning*

*So, a good idea would be for me to unpack data according to the protocol and category it belongs to, and to process data through my code that way*

*as to make it most sensical to the user*

*I think personally I want to start just by capturing and dissecting ICMP, TCP and UDP data*

A number of key points were discussed here:

* I should study packet encapsulation in order to properly implement algorithms to deal with the concept.
* It could be a good idea to implement TCP streams in order to demonstrate datagram vs stream based protocols and how they work.
* I should study the differences between UDP, DCCP, TCP and SCTP.

## Section 2: Encrypted Message Application

I have decided the encryption method I will be custom programming and using within my application will be RSA encryption. RSA is a public-key cryptosystem that relies on the practical difficulty of factoring the product of two large prime numbers. There are no published methods of defeating RSA encryption if a large enough key is used.

## Public Key Encryption

RSA uses what’s known as a private key and a public key. A public key is non-secret and is utilised to encrypt messages. The messages encrypted with a public key are only decrypted in a reasonable amount of time by its corresponding private key. The encryption and decryption method is best illustrated.



## The Factoring Problem

Every positive integer has a unique prime factorisation, by principle, each integer can be factored down into its constituent prime factors by repeated application of a factoring algorithm.

Fermat’s method is an example of a factorisation method, it’s based on the understanding that the difference of two squares, and therefore can represent an odd integer. A full explanation of Fermat’s method is not relevant, however, it’s clear with the aid of an example that the factoring problem is a fundamentally complex one.

If where are large enough prime numbers, trial division will quickly produce the prime factors 3 and 19, but will take divisions to find the next factor. In contrast, if is the product of prime numbers , and where , Fermat’s factorisation method after one application will start with which is evidently so much more complex.

No algorithm has actually been derived to handle this process in a time where b is the amount of bits in the number, and some constant of k.

## Mathematical Operation

Principally, the RSA algorithm employs the practical difficulty of deriving three very large integers, , , and , so that the modular exponentiation of all integers where :

where knowing and , or even , it can be *extremely* difficult to find .

The corresponding public key is represented by the integers and . The corresponding private key is represented by the integer . represents the message.

Simply put, the RSA algorithm has four key steps.

* **Key generation**
* **Key distribution**
* **Encryption**
* **Decryption**

**Key generation** for the RSA algorithm occurs the following way:

1. Two distinct prime numbers are chosen, and .
   1. These two numbers are chosen at random and in a secure fashion. They should, for efficiency and difficulty to penetrate, be similar in magnitude but differ in length by a few digits. The *Primality test* is the most efficient way of finding these prime integers.
   2. and are kept secret.
2. is computed as
   1. is used as the modulus for both the public and private keys. Therefore it is released as part of the public key.
3. is computed where is Carmichael’s totient function.
   1. is kept secret.
4. Integer is chosen such that and the greatest common divisor of and is equal to . Or that they are *coprime*.
   1. should have a short bit-length and a small *Hamming weight* to make encryption more efficient. Most commonly, is chosen to be , it can be as small as however it has been found to be less secure.
   2. is released as part of the public key.
5. Determine as
   1. This can be computed efficiently by using the *Extended Euclidean Algorithm*
   2. is kept secret as the private key exponent.

**Key distribution** occurs the following way:

Person A (sender) wants to send data to Person B (receiver). Using RSA, Person A must know Person B’s public key in order to encrypt the message. Person B will use their own private key to decrypt it. Therefore, Person B transmits their public key (, ) to Person A via a reliable but not necessarily secret path. It’s worth noting Person B’s private key is never shared.

**Encryption** works like this:

After obtaining Person B’s public key, Person A can send a message to Person B.

Initially, turns into via an agreed padding scheme. The ciphertext is computed using Person B’s public key in accordance to

is then transmitted to Person B.

**Decryption** works this way:

Person B can algebraically arrange for from by using her own private key by computing

Once deriving , the original message, can be originated by reversing the universally-agreed padding scheme.

## Data and Algorithms

The algorithms that I will be using in my project can be split into two sections, those used for the network packet sniffing part, and those used for the encrypted messaging part.

## Section 1: Network Sniffer

The main algorithms for the network sniffer will be those that deal with the unpacking of bytestreams, byte manipulation and data processing.

* Handling incoming data:

My algorithms will be able to accept, manage, manipulate and interpret all data incoming from a network point of access so that it is maintainable and accessible.

This involves extracting important and useful data from a large amount, and then refining it so that it is easily maintained and accessed.

* Interpreting data:

After handling all incoming data, my algorithms will be able to interpret that maintainable data in a human-readable format. The algorithm will also make the data more manageable to program with.

* Displaying data:

My algorithms will be able to properly display information that has been previously handled and interpreted, in a user-friendly format and properly sequenced according to filters used by the user. Most of the customizability in my program will derive from these algorithms.

## Section 2: Encrypted Message Application

Mainly, the algorithms surrounding this part of my application will be highly mathematical and will involve complex number theory as is the case with encryption algorithms.

* Handling a connection:

My application will contain an algorithm that handles a reliable data connection between two endpoints.

* Encryption:

My algorithms will be able to secure a data connection by reliably and visibly encrypting the data transmission. Proper encryption standards will be upheld in my algorithms, however the details of encryption will be available to the user as to satisfy the educational objective.

* Switching Encryption:

To enhance my target audience’s educational experience, my algorithms will be customisable, allowing certain variants of encryption. It will be able to use configurable variables to customise the outcome of the demonstration.

## Objectives

1. The user should be presented with an intuitive interface with clickable buttons and changeable screens.
   1. The interface should always display information on which technologies are being used and where to learn more about them.
   2. The screens should clearly state at which stage the user is capturing packets. (e.g home screen, select adapter, capturing).
   3. There should be a home screen when the user initially launches the application. It should contain a brief explanation of the program and its functions.
2. The application should be customisable and entirely interactive.
   1. Each screen should display clickable buttons as well as changeable variables to enhance the way the user can control the application. (e.g changing protocol being scanned & changing filter options).
3. The user should be able to capture network traffic both intended and not intended for their current device.
   1. The application should, at base level, be able to capture network traffic in packet form.
   2. The application should be able to capture packets that are intended for the user’s device, as well as capture packets that are not intended for the user’s device.
   3. The application should be able to capture packets and save them in local storage.
   4. The user should be able to choose one or many protocols to capture from at any given time.
4. The application should be able to interpret network traffic.
   1. The user should be able to select which network adapter to capture traffic from.
   2. Network traffic should be, when encapsulated in packets, be interpreted as human-readable data.
   3. If the adapter 4.1 cannot perform 3.3 and by extension 3.4, an error message must prompt the user with further information.
5. The application should be able to connect to another application or itself via a client-server architecture.
   1. The user should be able to type messages to another user, or to an automated server.
   2. The user should have the option to encrypt their message.
      1. The user should be able to customise their encryption choice, with either symmetric or asymmetric encryption.
   3. The user should have the opportunity to capture traffic on the same port as the chat, and therefore demonstrate the capability of packet sniffing.
      1. The user should have the opportunity to capture this traffic whilst it is encrypted, and demonstrate the capability of packet sniffing within the context of encryption.