Hochschule Worms
University of Applied Sciences

Bachelor Thesis

DESIGN AND IMPLEMENTATION OF CCEAP USER INTERFACE

Bachelor Thesis Submitted as Partial Fulfilment of the Requirements for the Degree of B.Sc. in Applied computer science at the University of applied sciences (Hochschule Worms)

Feugang Kemegni Fabrice

Submitted Summer, 2017
supervised by Prof. Dr. Stephen Wendzel

Applied Computer Science

Software construction

Declaration of Authenticity

I hereby declare that all material presented in this document is my own work. I fully and specifically acknowledge wherever adapted from other sources.

I understand that if at any time it is shown or proven that I have intentionally misrepresented material present here, any decree or credits awarded to me on the basis of the material may be revoked.

I declare that all statements and information contained here is true, correct and accurate to the best of my knowledge and belief.

Ludwigshafen, Summer 2017.

………………………………..

Feugang Kemegni Fabrice

Abstract

This Bachelor thesis discusses the development of a graphical user interface program as an interface to a command line program CCEAP.
CCEAP (Covert Channel Educational Analysis Protocol) is a networking protocol in the form of a command line program to help learn/teach covert channels and hiding patterns.
The cceap program is currently executed from the command line, by starting a cceap-server with a given port number of a computer, then on another command line starting the cceap-client program with the IP address and port number of the computer where the server is running. The client is started with some other parameters that correspond to the cceap's protocol settings.

The first section of this thesis is a discussion about what cceap is and the related concepts connected to it. Additional background information required to understand the purpose of cceap are also discussed in this section. Then some limitations of the cceap that lead to the need for a complementary graphical user interface are presented.

The second section discusses the implementation phase of the Graphical User interface tool. Then a complete discussion on the implementation of the GUI tool with some of the important portions of the code followed by a discussion of some problems/issues faced during the implementation of the program as well as the solutions to some of the problems.

In the last section, extensions to the GUI tool i.e. are presented. This refers to the features that could be added to the program during future development potentially by other students that might be involved in the evolution of CCEAP.

Introduction

What is CCEAP

To simplify the understanding of the term **CCEAP**, it is indispensable to firstly define some of the concepts that are related to it, in other words we must first discuss the core concepts behind cceap. These concepts include covert channels, hiding patterns, network steganography.

Covert Channels and Hiding patterns

Also called network steganographic hiding methods are techniques used to conceal communications within network traffics. In other words covert channels involve hiding communications in plain sight by using public media(in this case the network traffic) to practice secret communication. Secret communication in the sense that the fact that the communication is even taking place is unknown. There are more than a hundred of such network steganographic methods for concealing communication however as shown in (Wendzel et al., 2015) these can be summarized into 11 so called **Hiding Patterns**.

CCEAP

With such a brief understanding of what a covert channel in the domain of networking means without deeply diving into the broad subject, we can now jump to what CCEAP stands for and why are its purposes.

CCEAP(**Covert Channel Educational Analysis Protocol**) is a command line tool(program) that emulates an application layer protocol and is designed to help in teaching/learning covert channels.

It is an application layer protocol that uses Ethernet packets payloads as covert channel(the term covert channel is explained above).
The idea behind CCEAP is to provide such a protocol that will allow learners to model/manipulate this protocol's structure with the aim of creating a covert channel.

The main features of cceap is that it is based on hiding patterns (Wendzel et al., 2015), which is summarizes more than 120 network steganographic methods to only 11 hiding patterns. Thus providing an economic approach to learn only these 11 hiding patterns which summarize the hundred methods.

Additionally, cceap combines all methods of hiding data in network communications into a single protocol instead of using each separate protocols i.e. IP, UDP, HTTP, TCP etc. to demonstrate how covert channels work for each of them. [1]

1 https://github.com/cdpxe/CCEAP/tree/master/documentation
https://www.researchgate.net/project/Network-Information-Hiding-Patterns
https://www.researchgate.net/publication/316215336_Network_Information_Hiding_and_Science_20_Can_it_be_a

How does CCEAP function

Now that the term CCEAP has been explained, a discussion about how it functions should is in order.
The CCEAP is a cross platform command line tool that runs on windows, Unix Linux and MAC OS. Detailed explanations on the installation steps and other Instructions on running the CCEAP tool together with some practical exercises to demonstrate the functioning of the tool can be found on this webpage CCEAP documentation.
However here is a shorthand oh how to run the tool on the most widely used operating systems is provided below, but it is highly recommended to visit the CCEAP's websit.

After downloading cceap from the CCEAP's website, https://github.com/cdpxe/CCEAP.git into a folder on the computer, follow the steps below depending on the architecture.

Running CCEAP on windows

- open a command line tool **cmd**
- navigate to the folder where the CCEAP tools are stored i.e cd /path/to/CCEP/Folder/
- start server.exe -P 1234
- start client.exe -P 1234 -D 127.0.0.1

Running CCEAP on Unix

- open shell (Terminal)
- with the terminal, navigate to folder where CCEAP program is stored
- start server ./server -P 1234
- start client ./client -p 1234 -D 127.0.0.1

In the example above, **-P** is the Port number, which must be the same for client and server,

-D is the IP address of the computer where the server is running, if you are running the client and server on different computers, you need to replace the 127.0.0.1 with the actua address of the computer where the server is running.

Visit the CCEAP website for more details about the parameters and explanations [User CCEAP command Line tool](https://github.com/cdpxe/CCEAP/tree/master/documentation)

Match

Limitations of the CCEAP Tool

After haven tested the cceap tool, its relevance and support in understanding covert channel becomes obvious as provides a practical way to visualize some of the concepts of hiding patterns and covert channels.For non computer users or even the novice ones that are not familiar or those who find the use of the command line or **cmd** or terminal, it becomes apparent that something could be done to increase the usability of cceap.

Below are some aspects of cceap that could reduce the motivation of learners when using cceap to learn hiding patterns and covert channels.

CCEAP is a not a graphical program but instead a command line tool. For the computer novices or non programmers, or even some computer people but who are not in love with working with the command line, it becomes a drawback to work with cceap.

Another point is the fact that it becomes difficult to follow the flow of the program from begin till end and to interpret the programs output from the command line.
Even advanced users will find it difficult to interpret the output of the cceap server when it is displaying a hundred packets.
The outputs in the server are displayed one line after the other which is simple to understand when only about 10 to 20 packets are being analysed of 20 packets when the screen offers enough room for that. However when the servers output is a large number of packets and parameters, it becomes strenuous to follow.

Another drawback of the command line is the inability to undo and redo when entering parameters.
The command line reads parameters lines after lines, so it is difficult to undo/redo un such environment.

The program parameters are not very understandable for novice users even when reading the help section. This tool is easily understandable by experienced users, however casting out the novice who are part of the intended audience.
The command line tool is relatively complex and each system has its own independent command line tool that is used differently from other systems. Thus having knowledge or being experienced with the windows command line tool does not guarantee the same knowledge could be applied to a Unix command line tool.

Above are only some points that describe the main limitations of the CCEAP. Faced with these, a need to extend features to complete these holes have generated a need for a graphical user interface to interface this command line tool facilitating the use of the cceap.
The section below describe my solution to this, in the form of a graphical user interface for the command line tool.

What is my contribution to solve the problem

The above section discussed the little limitations of the CCEAP tool that need to be addressed in some way. Faced with these limitations of the tool, I will attempt to provide a durable solution in the form of a **GUI** tool.
This is a graphical program that offers an interface between the command line tool(CCEAP) and the user. The user interact with the GUI and the GUI translates the user inputs into program parameters for the cceap tool.

Such a tool is quite straight forward in requirements and specifications as it is an interface in-between users and the program, permitting users to understand the program parameters and facilitating the understanding and interpretation of the program’s output.

Below is a list of the specifications that the finished program should fulfil among other specifications.

The program should be highly usable graphical program, that is intuitive to use so as to facilitate users understanding and interaction.
The GUI should also present the cceap’s program output in a clear, understandable format to users.

The program should be highly transportable and usable. This means that the program should run on possibly several platforms i.e. Unix, windows MAC OS etc. without requiring any effort. This implies a meticulous choice of programming tools.

The program should be smart and robust in terms of dealing with users inputs. User’s inputs that do not make any sense should not lead to system fail but instead the program should identify the erroneous input and notify the user with the appropriate action to take. In this way helping in the learning process of covert channels and hiding patterns.

The users should be presented the program’s output in a format that will be easy to understand and interpret, thereby helping in teaching and increasing the usability of cceap.

Related Work

Cryptography- the science of communicating secret messages in public media is a broad topic with multiple branches and concepts.

Steganography on the other hand focusses on hiding the content of a communication in a public medium. In other words it is the science of privately communicating in public.

One of the main branches of steganography being covert channels focusses on the usage of public innocent communication media to communicate secret data in a manner that is unnoticeable if not unidentifiable manner.

One simple way of using covert channel is in the domain of networking(computer networking). Here an underlying protocol for Ethernet communication namely TCP/IP is been used as a covert channel to communicate information illegally using Enet Packet headers. This is exactly what cceap intends to address, i.e. teaching concepts of covert channels using Ethernet traffic.

Design and Implementation

This chapter discusses the implementation phase of the GUI tool.

Programming Requirements

To implement this GUI-Tool, I have made use of a variety of concepts, tools and other programs. Among some of these

- QT creator as IDE: for developing the layouts and to organize and structure the code. The reasons for selecting QT creator among the hundreds of other tools are discussed below.

- Git for version controls and history management.

Git is the most widely used version control framework today and is taking over the tasks of versioning, document and clouding programs.

Together with its associated tool GitHub, provides an intuitive approach to even beginners in programming. There are several other frameworks that implement version controls but most of them in one way or another make use of concepts from git. An example of such a framework is source tree[source tree](http://sourcetree.org)

- gcc, qmake, cmake, to compile

- OS
  - ubuntu
  - windows

It was imperative to work with 2 operating systems because one of the main features of this tool is its portability and transportability.
Transportability in the sense that program, both source code and executable should be easy transportable among platforms.

As main operating system I used ubuntu 16. and then each time transported the program to my windows platform to generate windows executable .exe files for windows distribution.

In the section **deployment** below, I explain how this process of writing the program under platform and the generating a distributed version of the program for another platform.

Deployment is often done in the target platform, i.e. to get the windows executable can only be done on a windows platform and the Unix version on a Unix platform respectively.

Why Qt Creator is a cross-platform C++, JavaScript and QML integrated development environment.

QT creator comes with several useful programming resources including

- a visual debugger
- an integrated GUI layout and forms designer.
- syntax highlighting and autocompletion in editor's features.
- C++ compiler from the GNU Compiler Collection on Linux and FreeBSD.
- On Windows it can:
- use MinGW or MSVC
- Microsoft Console Debugger (when compiled from source code).
- Clang.

Due to these features, it became intuitive to choose it as the development tool.

Design decisions

I started writing the program in the Java programming language due to the fact that Java is a highly portable programming language, it is one of the languages that I know the most and I have gathered some experiences through homework and projects in school.

By saying a highly portable language, I refer to the fact that a program written using a Java language on a given system can be exported to other systems without any strenuous operation.

This is because Java programs are not technically compiled to produce a static compiled output, but instead programs written in the Java language are translated from the high level code of programming language into a java machine language(dynamic code). This java machine is called the Java virtual Machine (JVM).

To be able to transport a program from an architecture (source architecture) to another architecture(target architecture)) the only requirement is the presence of the JVM and JAVAC on the target architecture.

Both the JVM and JAVAC are open-source programs available
https://www.java.com/en/download/

http://www.oracle.com/technetwork/java/javase/downloads/index.html

the system architecture only requires a JVM (Java virtual machine) running in background

- javac JRE
- java JDK
- Eclipse
- Window Builder plug-in for Eclipse
- Windows System
- Ubuntu/any Linux distribution for testing
- virtual box for virtualisation in case only one architecture is available

After more than one week implementing the CCEAP-GUI program in java language, I came to a deadlock and realised that I could not implement such a program using java. The reasons being that the following.

In order to be able to place the UI controls on the screen in the desired positions, the container in which such UI controls would be placed needed an absolute layout such that items(UI-controls) can be placed on the container independent of the positions of the other ui controls already on the container.
However java GUI containers i.e. AWT and swing do not offer absolute layouts but relative layouts, such that an item can only be placed relative to the others, for example in a table or on bordered layout. This was a very serious problem that pulled me back on research.

Another reason for abandoning java in the project was the absence of a designer mode in the most Java programming IDEs. Using the open source eclipse, the only alternative was using plug-ins such as Windows builder, but this plugin was very slow and kept on freezing.

Faced with these problems, I was forced to switch to an alternative language, and after some research I came to choose C with QT creator. This solved my problem of UI designing.

Qt Creator

Qt Creator is a cross-platform IDE supporting the languages C/C++, JavaScript, QML. QT Creator offers features like a visual debugger, an integrated designer for GUI layout and forms. The editor's features include syntax highlighting and autocompletion.

Qt Creator provides support for building, deploying and running Qt applications for desktop environments including Windows, Linux, FreeBSD and Mac OS.
QT creator also provides programming applications for mobile devices including Android, BlackBerry, embedded Linux devices and several others. Build settings allow to switch between build targets, different Qt versions and build configurations.

CCEAP GUI is developed with QT creator, because as a result of prior research for the ultimate tool to develop the app as it has been described in the problem statement and program specifications, it appears that QT creator offered be the best alternative to develop such a GUI program. As mentioned above I had started implementing the app using java on eclipse and came to a death lock before coming to QT Creator.
Some of the reasons are explained below.

UI controls are easily placed on the window container as compared to the Java alternative. This is because QT creator allows drag and drop of UI Controls on desired positions of the container, as compared to Java where containers are constraint in Layouts, for example a Border layout only allows a UI control to be placed at the left of the previous controls.

Transporting apps between architectures are relatively simpler on QT as is the case with eclipse. It is to mention here that one of the main advantages of Java over other languages and frameworks is the fact that java code is not compiled for a given output, instead the source code is translated from high to java machine language, and the only requirement to reuse a code on different platforms is the presence of a java machine on the target machine.QT Creator is available under https://www.qt.io/ide/.

Main Code sections

In this section I discuss some of the most important parts of the GUI codes. Most of the functions here can be reused as a library, this is one of the main reasons that I have included in this documentation.
The functions are well commented and aligned in a self-explanatory order, thus I found it redundant to add explanations for the code here.

Server-GUI

```
/*The app's entry point simply starts the server-gui*/
int main(int argc, char *argv[])
{
    QApplication application(argc, argv);
    ServerGui serverGui;
    serverGui.show();

      ...

    return application.exec();
}
```

ServerGui

```
ServerGui::ServerGui(QWidget *parent) : QMainWindow(parent), ui(new Ui::CCEAP)
{

    //inig the GUI
    ui->setupUi(this);

    //hint the server's ipaddress
    ui->ip_lineEdit->setPlaceholderText("server listening to: "+getLocalIP());

    //init the model that formats the way data will be displayed in the ListView
    model = new QStringListModel(this);

    //welcome text that appears when server ui starts.
    //uses a stringlist so as to format each item to be clickable.
    QStringList List;
    List << ""
            "CCEAP protocol implementation. Copyright (C) 2016 Steffen 
Wendzel\n"
            "This program comes with ABSOLUTELY NO WARRANTY; for details see 
LICENSE file.\n"
            "This is free software, and you are welcome to redistribute it under 
certain conditions;\n"
            "for details see LICENSE file.\n"
            "CCEAP - Covert Channel Educational Analysis Protocol (Server)\n"
            "=> version: 0.5.2, written by: Steffen Wendzel, www.wendzel.de\n"
            "\n"
            "start the server to begin an analysis session:";
    //assigng the string list to the model and assign the model to the listview.
    //This is all that is required to display the hello world page.
    model->setStringList(List);
    ui->listView->setModel(model);

}



ServerGui::~ServerGui()
{
      …

    delete ui;
}

//This is the handle for the button 'start server'.
void ServerGui::on_startServerButton_clicked()
{
    //clear display
    stringList.clear();

    /*This handler instantiates the server thread.
     * and registers a handler so that the serveThread can then emit a signal
back to the
     * gui once it has completed its task.
    The serverThread just executes the cceap-server with th port provided by the
client.
    if client does not provide a port, a default port 4444 is used.

The server thread runs the server in background, waiting for any client
connection.
After receiving and processing a clients connection, the cceap returns a string
to the serverThread,
the serverThread then parses the returned string to the cceap gui in the form of
a stringList that will
be placed on the ListView.*/

    ServerThread *serverThread = new ServerThread(ui->port_lineEdit->text());

    /*This registers the slot(the gui's dataReceivedFromServer(QStringList)
function)
     * with a stringList parameter.
     * Once the serverThread has completed its task,
     * it will then emit a signal and return a QStringList to this gui
     * upon emiting the signal and QStringList, the function
dataReceivedFromServer(QStringList)
     * will be called to process the result  QStringList.
    */

    QObject::connect(serverThread,SIGNAL(signal(QStringList)),this,
SLOT(dataReceivedFromServer(QStringList)));

    //start the serverThread in background
    serverThread->start();

}
```

```
//This slot is called automatically when the serverThread emits its signal
void ServerGui::dataReceivedFromServer(QStringList list)
{

    /*The stringList returned by the thread is already formated,
     * so this gui only needs to display it in its listview.
     *
    */
    this->stringList << list;
    this->model->setStringList(stringList);
    this->ui->listView->setModel(model);
    this->ui->listView->update();
}

//This returns the ip address to display to user so that they can connect with
QString ServerGui::getLocalIP(){
    foreach (const QHostAddress &address, QNetworkInterface::allAddresses())
        if (address.protocol() == QAbstractSocket::IPv4Protocol && address !=
QHostAddress(QHostAddress::LocalHost))
             return address.toString();
    return "127.0.0.1";
}
```

ServerThread

```
void ServerThread::run()
{

    QStringList list;

    //veryfy that port number is valid, if not use defaul port number
    if(port.length()==4 &&
            port.at(0).isDigit()&&port.at(1).isDigit()&&
            port.at(2).isDigit()&&port.at(3).isDigit())
{
        list << "server started on port: "+port;
                "\nwaiting for clients ...";

        //signal GUI that server has successfully started

    }
    else{
        list << "Error !! The port number that you have provided is invalid.\n"
             "please check it.\n";
    }

    emit signal(list);
    qDebug() << list;

    /*QProcess is used to execute an external program.
     * An alternative would be using a system call,
     * but because we still need to process the output of the external prograam
     * we need to start a process instead and
     * waitFirFinish blocks the code to wait for cceapServer to exit and return
its
     * output before thisThread can proceed.
     * This is required because we need to process the output of the
cceapServer.
    */

    QProcess cceapServer;
    cceapServer.start("./server -P "+port);

    //forces this.Thead to wait for output of cceapServer
    cceapServer.waitForFinished(-1);

    /*read all cceapServer's outputs
     * both readAllStandardOutput and readAllStandardOutput
     * are returned in 'readAllStandardOutput'
     * so no need to read both of them
    */
    QString stdout = cceapServer.readAllStandardOutput();
    //QString stderror = cceapServer.readAllStandardOutput();

    qDebug() << "stdout: "+stdout.length() << stdout;

    /*The output of cceapServer is a string, howver the GUI needs alist of
packets
     * of list of error to display to users, so the stdout string must be split
into a StringList
     * before been returned.
     * In the cceapServer output, each packet begins with received Data ...
     * so we use such pattern as delimiter to split the String into an String
list
     *
    */
    QRegExp rx("received");
    list = stdout.split(rx);

    //return tha processed data(StringList) to the GUI then exit
    emit signal(list);

}

/* This function checks that the given port is valid,
 * if not valid, returns the default port
*/
QString validateIp(QString port){

    //veryfy that port number is valid, if not use defaul port number
    if(port.length()==4)
        if(port.at(0).isDigit()&&port.at(1).isDigit()&&
                        port.at(2).isDigit()&&port.at(3).isDigit()) ;
        else port="4444";
    else port="4444";
    return port;
}
```

Client-GUI

```
ClientGui::ClientGui(QWidget *parent) :QMainWindow(parent),ui(new Ui::CCEAP)
{
    //init the gui

    ui->setupUi(this);
    initMenuBar();

    //init th listview and its correspponding model.
    model = new QStringListModel(this);
    ui->listView->setModel(model);

    //hide the sequnce number fields, this will be displayed later by user
    ui->i_label->setVisible(false);
    ui->i_spinbox->setVisible(false);
    ui->s_label->setVisible(false);
    ui->s_plainTextEdit->setVisible(false);

    //regular expressions to validate ipaddress
    QRegExp rx( "[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}" );
    QRegExpValidator regValidator( rx, 0 );
    ui->D_lineEdit->setValidator( &regValidator );
...


}

/*function to convert a String into a char array.
 * This is required because QProcess accepts only
 * char array as üarameter*/

char* ClientGui::qStringToCharPtr(QString str){
    char*  cstr = new char[str.length()];
    for(int i =0; i < str.length(); i++)
        cstr[i] = str.at(i).toLatin1();
    return cstr;
}

void ClientGui::clcearScreen()
{

    //clear display
    result.clear();
    model->setStringList(result);
    ui->listView->setModel(model);
    ui->listView->scrollToBottom();
    ui->listView->update();
}

void ClientGui::display(QString data){

    result.clear();
    result << data;
    model->setStringList(result);
    ui->listView->setModel(model);
    ui->listView->scrollToBottom();
    ui->listView->update();
}
```

```
void ClientGui::display(QStringList data){

    result.clear();
    model->setStringList(data);
    ui->listView->setModel(model);
    ui->listView->scrollToBottom();
    ui->listView->update();
}

bool ClientGui::isIPAddress(QString ipaddr)
{
    qDebug() <<ipaddr;
    QRegExp rx( "[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}" );
    QRegExpValidator regValidator( rx, 0 );

    return true;
}

void ClientGui::initMenuBar(){
    //Initialize the menubar
    ui->menuHelp->addAction("about CCEAP", this, SLOT(aboutCceap()));
    ui->menuHelp->addAction("developers", this, SLOT(developers()));
    ui->menuHelp->addAction("participate", this, SLOT(participate()));
    ui->menuHelp->addAction("help", this, SLOT(help()));
}
/*This function hardcodes a brief description about cceap*/
void ClientGui::aboutCceap(){
    QStringList message;

    display(message);

}
//display information about people involved in the developement of cceap
void ClientGui::developers(){
    QStringList message;


...
    display(message);

}
//display information for users who may be interested in participating to cceap
void ClientGui::participate(){
    QStringList message;
    message << "Please contact Prof Wendzel to participate to CCEAP.\n";
    message << "www.wendzel.de";
    message << "wendzel@hs-worms.de";

    display(message);
}
//this function starts the thread in bg, the thread then calls the ccceap --help
and displays the output to user
void ClientGui::help(){
    Thead *thread = new Thead("./client -h");
    Qobject::connect(
thread,
SIGNAL(signal(QStringList)),
this,
SLOT(dataReceived(QStringList)));
    thread->start();;
}
void ClientGui::dataReceived(QStringList data)
{
    display(data);
}
```

on_sendDataClicked

```
/*This function parses the users inputs for each field corresponding to the
cceap parameters,
 * then starts a thread, the tread calls cceap server with the parameters, waits
for the result,
 * when the cceap output is available, the thread then emits a signal back to
ClientGui.dataReceived(),
 * the latter simply displays the output to the user
*/


void ClientGui::on_sendDataB_clicked()
{

    QString parameters="";
    QString list;

    QString temp;

    /*Parse port number.
     *A port number must be 4 charachters,
     * each character must be a digit,
     * if any of these conditions fails, use the default port number 4444.
    */
    QString port,tmp = ui->P_lineEdit->text();
    if(tmp.length()==4 && tmp.at(0).isDigit() && tmp.at(1).isDigit() &&
tmp.at(3).isDigit() && tmp.at(3).isDigit())
        port=tmp;
    else
        port= "4444";
    parameters += " -P "+port;
    list += "\n'-P' TCP port x to connect to: "+port;

    /*Parse Ip address.
     * An ip address must be atleast 6 charaters and at most 15
     * must contain 3 dots in which
     * the firs and last characters are digits,
     * a dot should not preceed another
    */
    QString ip = ui->D_lineEdit->text();
    if(ip.length()<6)
        ip="127.0.0.1";
    parameters += " -D "+ip;
    list += "\n'-D'  Destination IP x to connect to: "+ip;

    // is verbose enabled?
    if(ui->v_checkBox->isChecked()){
        parameters += " -v ";
        list += "\n'-v' verbose mode: true";
    }
    else list += "\n'-v' verbose mode: false";

    parameters += " -c "+QString::number(ui->c_spinBox->value());
    list += "\n'-c' Number of packets to send: "+QString::number(ui->c_spinBox-
>value());

    parameters += " -u "+QString::number(ui->u_spinBox->value());
    list += "\n'-u' Dummy' value in the main header: "+QString::number(ui-
>u_spinBox->value());

    /*User can only use predefined sequence numbers | a starting sequence
number.
     * Users choice will then make the appropriate input visible
    */
    if(ui->i_label->isVisible()){
        parameters += " -i "+QString::number(ui->i_spinbox->value());
        list += "\n'-i' Sequence number x to use for CCEAP:
"+QString::number(ui->i_spinbox->value());
    }
    if(ui->s_label->isVisible()){
        parameters += " -s "+ui->s_plainTextEdit->toPlainText();
        list += "\n'-s' pre defined sequence numbers: "+ui->s_plainTextEdit-
>toPlainText();
    }

    temp =ui->t_plainTextEdit->toPlainText();
    if(temp.length() > 1){
        parameters += " -t "+temp;
        list += "\n'-t' Inter-arrival times between packets: "+temp;
    }

    temp = QString::number(ui->p_spinBox->value());
    if(temp.length()>0){
        parameters += " -p "+temp;
        list += "\n'-p' Seq_No of packets to be duplicated : "+temp;
    }

    temp = QString::number(ui->x_spinBox->value());
    if(temp.length()>0){
        parameters += " -x "+temp;
        list += "\n'-x' Packets to be Excluded: "+temp;
    }

    temp = ui->d_lineEdit->text();
    if(temp.length()>0){
        parameters += " -d "+temp;
        list += "\n'-d' CCEAP destination address: "+temp;
    }

    temp = ui->o_lineEdit->text();
    if(temp.length()>0){
        parameters += " -o "+temp;
        list += "\n'-o' Optional header elements: "+temp;
    }

    //parse command
    char* command = qStringToCharPtr("./client "+parameters);

    Thead *thread = new Thead(command);
    QObject::connect(thread,SIGNAL(signal(QStringList)),this,
SLOT(parseData(QStringList)));
    thread->start();;

    /* Start a process.

    //display program's output in a formated layout, separating each output with
a horizontal line
    result <<  "\n"+parameters+
                "\n"+list+
                "\nyour data is been sent\n"
    ".................................................";

    display(result);
}

void ClientGui::on_clearScreenB_clicked()
{
    clcearScreen();
}

/*event handlre for changed in seqNumber options
 * this simply enables the ui cotrol corresponding to users choice
*/
void ClientGui::on_seqNoType_currentIndexChanged(int index)
{
    if(index==1){
        ui->i_spinbox->setVisible(true);
        ui->i_label->setVisible(true);

        ui->s_plainTextEdit->setVisible(false);
        ui->s_label->setVisible(false);
    }
    else if(index==2){
        ui->i_spinbox->setVisible(false);
        ui->i_label->setVisible(false);

        ui->s_plainTextEdit->setVisible(true);
        ui->s_label->setVisible(true);
    }
    else{
        ui->i_spinbox->setVisible(false);
        ui->s_plainTextEdit->setVisible(false);
        ui->i_label->setVisible(false);
        ui->s_label->setVisible(false);
    }
}

void ClientGui::parseData(QStringList data){
    display(data);
}
```

CCEAP Thread

```
Thead::Thead(){}
Thead::Thead(QString data):cmd(data){}

void Thead::run()
{
    qDebug() << "thread running";
    QStringList list;

    /*QProcess is used to execute an external program.
     * An alternative would be using a system call,
     * but because we still need to process the output of the external prograam
     * we need to start a process instead and
     * waitFirFinish blocks the code to wait for cceapServer to exit and return
its
     * output before thisThread can proceed.
     * This is required because we need to process the output of the
cceapServer.
    */

    //cmd = "./client -h ";
    QProcess thread;
    thread.start(cmd);

    //forces this.Thead to wait for output of cceapServer
    thread.waitForFinished(-1);

    list << thread.readAllStandardOutput() << thread.readAllStandardError();

    qDebug() << list;

    //return tha processed data(StringList) to the GUI then exit
    emit signal(list);

}
```

Problems encountered and Solutions

The program implemented during this project was a simple and quite easy understand. However the implementation was characterized by lots of problems encountered mostly due to the fact that I was familiar to the programming environment and my lack of experience with such types of programs.

At the end most of the problems were solved despite taking relatively much time to be dealt with. Among some of the problems that I faced, the following are the most prominent and are those that required much time.

I started writing the program in the Java programming language due to the fact that Java is a highly portable programming language, it is one of the languages that I know the most and I have gathered some experiences through homework and projects in school. Although I carried out some research on the portable languages to fit the tasks on hand but at some point this research was biased by the fact that I already knew java, so I jumped into implementation and spent about 15% of the whole time for this project on a wrong path.
It is extremely difficult to use Java SWING and AWT classes as means of developing complex User Interface programs.

Although the SWING and AWT widgets are are designed to create UI programs, they are better appropriate for some tasks than some others, for my task they were not appropriate. These classes are appropriate for UI such as calculator, video player, etc. where a.

For the CCEAP, the available containers apply useless because some items need absolute positioning on well defined coordinates, whereas AWT and SWING containers only offer Border layout and flow layouts.

Executing an application from QT, and making system calls. The second problem and one of the most time consuming problems was making a system call from the GUI program to starting both the CCEAP client and server.
I have stayed on death lock for more than a week trying to run the server and client program from the GUI app using the parameters that the users entered in the latter.
Each of the approaches that I used didn’t work and each time I ran the program nothing noticeable happened, at best when I closed the program a debug message said, client program started and is still running although the GUI app had already exited.

Dead-lock with server freezing while waiting for data. After the above problem was solved, I immediately found myself with another death-lock. This happens when the server GUI is started and waiting for clients, after waiting for some time without receiving any client data, the server freezes and only unfreezes once a client has connected and sent some packets. The server GUI then unfreezes and displays the programs output from the CCEAP.

The last but not the least problem was that of live communication between the CCEAP GUI and the CCEAP app. This is not considered as a problem because it was not included in the programs backlog. It was just intended to be a feature of the initial GUI app but instead it is an extension that I have left behind for the future students that will continue the development of this application.

The feature consists in that, the server GUI should display live output from the server program as it is been outputted. Currently, the GUI displays the programs output once the cceap server has exited. It would be a nice feature to have the cceap server’s output lively displayed on the GUI. I am not sure that it is possible to have such output but this research was out of the scope of my thesis, so I left this task for the future developers of the CCEAP.
Nonetheless, I have spent considerable time researching on how to implement this task.

My remark about the implementation

Despite all the planning that I went through, i have come to learn that, practice is a different topic from theory and contigency is always the best plan when experience is unavailable.

Deploying the app to windows

This section describes how to create an executable file for the windows platform. In order words how to run the program on windows in case the application does not provide a windows executable file. The following link offers more detailed information on how to deploy a program developed in QT creator to different target architectures [Deploy an Application on Windows] (http://wiki.qt.io/Deploy_an_Application_on_Windows).

Requirements

- The release version of your app works correctly when you build + run it from Qt Creator
- Qt is installed in C:\Qt\5.2.1\mingw48_32\
- This also applies to
- Qt 5.6 and mingw49_32

Steps (Hard Code)

1. Close Qt Creator.
2. Create a folder to hold the app eg ~\MyApp\
3. cp \QT\Projects\MyApp\MyApp.exe ~\MyApp\
4. cp C:\Qt\5.2.1\mingw48_32\bin\*.dll ~\MyApp\
5. cp from C:\Qt\5.2.1\mingw48_32\plugins\* ~\MyApp\
6. cp C:\Qt\5.2.1\mingw48_32\qml\* ~\MyApp\
7. Test if deployment worked

    1. mv (rename) C:\Qt C:\QtHidden\ (This turns your PC into a clean environment, just like one that doesn't have Qt installed).

    2. ~\MyApp\\MyApp.exe

    3. if it works then perfect

8. Cleanup

    - start ~\MyApp\MyApp.exe
    - while ie running,
    -rm ~\MyApp* and skip eroneous files.(this will delete only unnecessary dll files)

9. restart ~\MyApp\MyApp.exe

    - it it works, then the app can be distributed.

Extensions to the application

The problem statement of this thesis requires the development of a GUI tool that provides interface to interact with the CCEAP command line program.
The basic requirements are the simplicity and portability of the program. Portability in terms of the possibility and ease with which the program can be ran on different operating systems and architectures.
The requirements of the program are quite clear and the required features have totally fulfilled.

However, during and after the implementation, I have come to identify some features that could be added to this GUI tool. These features I have called extensions as it is often done in computing to mean features that could be added to an existing program to extend the programs functionalities. Below I have listed some of these functionalities, and these are intended to be developed in the future by students who will continue the development of CCEAP.

Features that could be added to the CCEAP-Gui tool

The first feature will be an Export command. This could be extended to the clients GUI so that once a session with all input parameters for a purpose will be used, these settings could be exported to an external file in JSON, XML, CSV, oData or any other format. , so that next time the user do not need to re enter such parameters but can open the file in which these settings were saved and directly copy from there into the GUI.
This will increase the overall usability of the program.

As a further extension to the above extension, could be the introduction of an import feature, this permits users to directly import settings from a file into the GUI program. This is very practical when for example a client starts an analysis session with 100 packets giving each packet a sequence number. Even though these 100 sequence numbers might have been saved into a file, typing them into the program is quite strenuous and time consuming. Export and Import features have been separated because each export might not absolutely require and import. The user might just need to save settings into a file without forcedly needing to import them.

Live communication between UI tools and programs: This feature should permit data to be displayed on the GUI while the cceap program is performing actions. Currently, the GUI starts the CCEAP program with the parameters entered by the user, then only display the returned output of the cceap after the latter has exited.
The idea here is to display the cceap’s output while the latter is running, not simply displaying the latter’s output after it exits.
I have not researched on this much because it was out of the scope of my thesis, so I left it behind for future developers.

A further feature but this time the starting point will be testing the GUI and CCEAP on load tests. The tests that were carried during and after implementation were functionality tests to ensure that the programs behave as expected independent of user's inputs and systems architectures.

All tests (both systems and functional) returned positive as the program recognises user's input errors and handles them as expected. After all tests, no know bugs have been identified.

The task here will be to test both cceap and the corresponding GUI tools on their behaviours when faced with overloads for example sending a thousand packets.
These have not been tested because thousand packets as parameters go out of the scope of CCEAP as students might not need to analyse a thousand packets in a session while learning covert channels.

The necessity to test appear now because as CCEAP becomes widely used, several users might have to connect to a single server, the behaviour of the server in response should be tested.

Conclusion

The aim of my thesis has been to implement a GUI (Graphical User Interface) program as a tool to complement an existing program (CCEAP tool).
In the introduction I have explained all useful terms and concepts related to CCEAP and the reasons and purpose of the CCEAP.

The importance of this work lies not only on the program, but also on the application of the concepts that I have acquired during my undergraduate studies. I would like to emphasize on the fact that the application developed in this project is only for an academic purpose, a lightweight GUI-tool with the sole purpose of providing a Graphical User Interface to the already existing command line program.
It is not intended for commercial purpose but instead to serve for academic purposes especially to contribute in learning/teaching network steganographic hiding patterns.

This program has not been extensively been tested, due to the fact that there is no immediate need for testing, as the development will be continued by other students like me, who should focus on testing before building from my foundations.

Developing this application has provided me the opportunity to carry out some amount of academic research. During this development, I have come across several difficulties and problems, but with some efforts I have been able to overcome them or find away around.

In the Chapter Related works, I describe some related works, projects and tools that till now provided alternative solutions to the problem that i solved in this project.

In the chapter Implementation, I try to describe the steps that I took in implementing the app. I also attached the Git repository where the project is hosted, with a detailed git log, that could be followed to understand the historic of the development.

In the section 'problems encountered, I list some of the difficult that I encountered followed by my solutions and some advices to other programmers that may come across similar problems.
The chapter Important code is a presentation of the most important code sections of the program. I tried to make it as modular as possible to encourage reusing of the code, thus the reason for providing such section.

The chapter extensions describe a list of tasks and features that students that will continue the development of CCEAP should first implement so as to keep

Abbreviations

- CCEAP: Covert Channel Educational Analysis Protocol
- CMD: Command Line tool
- CSV: Comma Separated Values
- Enet: Ethernet
- GUI: Graphical User Interface
- HTTP:
- IDE: Integrated Development Environment
- IP: Internet Protocol
- Javac: java compiler
- JDK: Java-Development Kit
- JRE: Java-Runtime Environment
- JSON: JavaScript Object Notation
- JVM: Java Virtual Machine
- oData: Open Data source
- OS: Operating System
- OS: Operating System
- QML: Qt Meta Language/Qt Modelling Language
- SDK: Software Development Kit
- TCP/IP: TCP+ Internet Protocol
- TCP: Transmission Control Protocol
- TCP: Transmission Control Protocol
- UI: User Interface
- UDP: User Datagram Protocol
- XML: Extensible Markup Language

Bibliography

[this publication (https://www.researchgate.net/profile/Steffen_Wendzel/publication/263773592_Hidden_and_Uncontrolled__On_the_Emergence_of_Network_Steganographic_Threats/links/53eb38eb0cf2dc24b3cea87a.pdf).

[Hidden and Uncontrolled - On the Emergence of Network Steganographic Threats] (https://www.researchgate.net/publication/263773592_Hidden_and_Uncontrolled__On_the_Emergence_of_Network_Steganographic_Threats).

[Hiding patterns](http://ih-patterns.blogspot.de/p/introduction.html)

[CCEAP-Master](https://github.com/cdpxe/CCEAP)

[An Educational Network Protocol for Covert Channel Analysis Using Patterns (http://dl.acm.org/citation.cfm?id=2989037)

[This Thesis] (https://github.com/fabrigeas/bachelor-thesis/)

[User CCEAP command Line tool](https://github.com/cdpxe/CCEAP)

[Javac] http://www.oracle.com/technetwork/java/javase/downloads/index.html

[JVM] https://www.java.com/en/download/

[QT Creator] https://www.qt.io/ide/