Skype Traffic Detector

Atwin O. Calchand, Van T. Dinh, Philip Branch, Jason But
Centre for Advanced Internet Architectures, Technical Report 090128A
Swinburne University of Technology
Melbourne, Australia
pbranch@swin.edu.au, jbut@swin.edu.au

Abstract—**This paper describes the need for the Skype Traffic Detector, a GUI designed for the purpose of filtering and displaying VoIP calls made from the Skype software.**

I. Introduction

Real-time detection of network traffic types is of recent but increasing interest for law enforcement agencies and telecommunications companies alike. With increased criminal and terrorist activities in the past few years, law enforcement agencies have realised that the use of data networks for communication is making their job of identifying imminent threats more difficult. Freely available VoIP (Voice over Internet Protocol) software, which offers free PC-to-PC calls, is being used by more and more people around the globe to communicate. Skype has captured the attention of the aforementioned authorities since it is the most widely used VoIP software in the world. Due to its closed-source code, research institutes are also taking a keen interest in Skype to better understand how it works.

Lawful Interception of IP packets, e.g. VoIP calls, can help law enforcement agencies track criminal and terrorist activities to ensure a country's security. Telecommunications companies are interested in identifying Skype traffic since they are facing losses on their traditional voice networks. Understanding how Skype is being used would help them better shape their business strategies. Internet Service Providers (ISPs), on the other hand, can use data gathered from detected Skype traffic flows to generate statistics and hence provide additional resources for increased Quality of Service on real-time VoIP calls. Skype relays are also a concern to ISPs since Skype makes use of nodes within networks to route packets without the nodes' owners knowing about it.

We have used machine learning techniques to analyse network traffic traces to detect Skype traffic. However this causes a problem in the amount of data that needs to be interpreted. Going through and trying to make sense of all this data is virtually impossible, specially when quick results are needed for the purposes of lawful interception, call tracking and so forth. Furthermore, it would be better to collate and display this information using a simple and intuitive graphical user interface which would make it easier to check for Skype calls. We have developed such an interface that would allow the user to quickly locate and filter Skype traffic flows.

The rest of the paper is structured as follows. Section II describes the tools required to make the detector work. It also covers the components making up the Detector and describes the function of each. Section III elaborates on how to set up the machine learning tools developed previously by Fokus Institute, Germany and CAIA. The manual pages of those tools need to be checked for a thorough understanding of how they work. Section IV describes how to start the tools and make a trace.

II. The Tools

Machine Learning (ML) is part of the broad area of Artificial Intelligence. ML mostly has to do with the design and development of algorithms that allow computers to learn from provided data sets to improve their performance at a specific task. Datasets can be in the form of databases and data gathered from sensors. Using those datasets a Machine Learning algorithm then builds models based on patterns it detects. Due to these characteristics, Machine Learning is closely related to applications in data mining, statistics, inductive reasoning, pattern recognition, and theoretical computer science [1].

For the purpose of our research our dataset consisted of traffic dumps (stored in *.pcap* format) [2] which were composed of both Skype and other network traffic. The tools used were:

- NetAI - uses ML algorithms to generate statistics based on traffic flows
- Netmate - classifies traffic flows
- WEKA - machine learning software used by NetAI

- Skype Traffic Detector - for filtering and displaying results

WEKA (Waikato Environment for Knowledge Analysis) is a collection of machine learning algorithms. The WEKA software suite was developed at the University of Waikato in New Zealand [3]. NetAI makes use of WEKA's capabilities to generate statistics on data acquired from NetMate and classifies that data based on patterns and defined rules (depending on the machine learning algorithm being used).

The Skype Traffic Detector was written in python and consists of three main parts: the filter, the graphical user interface (GUI) and a database. The filter's function is to read data from netAI's output file and store it in the database. As data is read, it is compared to existing records to see if it matches previous entries in the database. If an entry matches, the record in the database is updated so that there are no repeating records. Also, a timeout period and "call end" flag determines whether a record is kept or removed from the database. The GUI's function is to display all calls in the database and update the display at fixed intervals to be in sync with the contents in the database. The GUI is the main program the user interacts with whilst the filter runs in background in another thread. The GUI also allows a user to display specific records from the database. The user has to enter desired IP addresses and the GUI will update the display for matching records.

By default, when the detector is launched, all the appropriate fields are displayed. Using command line arguments, the user can choose to display distinct fields.

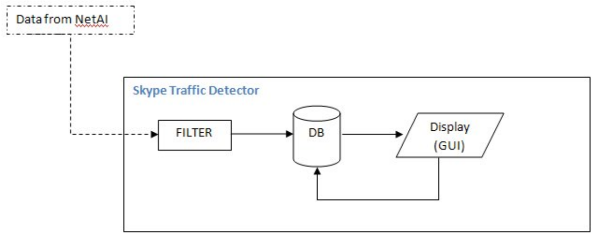


Fig. 1. Block Diagram of the Skype Traffic Detector

III. Making It All Work

In this section we describe how to install the various tools in order to make the detector work. The documentation of those tools needs to be examined before they are installed. The platforms on which they are installed needs to be taken into consideration as well. Netmate and netAI only work under UNIX-like environments and their dependencies need to be installed on these systems before they are set up. In our research, we have used FreeBSD 7.0 as our test platform. Installation procedure may vary on other operating systems.

A. Quick Start Guide

The netAI package can be obtained from the caia website at http://caia.swin.edu.au/urp/dstc/netai/. Once the tarball is downloaded, extract it to a directory of your choice then cd into that directory. Issue the following command: `./install-sh`. This command will download and install WEKA and netmate automatically. Below are the steps if the tarball file is downloaded onto the Desktop. [4]

```
cd /root/Desktop/
tar -xvzf netAI-0.1.tar.gz
cd netAI-0.1
./install-sh
```

B. Alternative Installation Procedure

It might occur in some cases that the automated install from the netAI package will not work as it should. In which case, a manual install of the different packages is required. Downloading the individual packages and then putting them into the *install directory* of netAI and re-issuing the command `./install-sh` as demonstrated above also worked in some cases. Follow the instructions below to install each package.

First of all, WEKA needs to be installed. To do so, cd into the WEKA FreeBSD port tree; `cd /usr/ports/textproc/weka/` and issue the command `make && make install` and wait for the installation to complete. To check whether the installation was successful, issue the command `weka` in a command-line interface, such as a terminal or shell, and it should display the user-interface of WEKA. If an x-server is not available, the command-line version of WEKA should show up. In our research, WEKA was tested on FreeBSD 7.0 running X11 server with KDE 3.5.

Once WEKA is installed, netmate and netAI can be set up - it does not matter in which order they are installed. Netmate can be downloaded from http://sourceforge.net/projects/netmate-meter/ and NetAI can be obtained from http://caia.swin.edu.au/urp/dstc/netai/.

1) Installing NetMate: Once downloaded, extract the tarball and cd into it. Then issue the following commands [5]:

```
./configure
gmake
gmake install
```

Notice that *gmake* was used instead of *make* (as described in the original installation guide). It was noted that *gmake* handles the installation process much better than *make* when we tried to install the packages under FreeBSD 7.0.

2) Installing netAI: The installation for netAI is more challenging. After the tarball for netAI has been downloaded and extracted, cd into the extracted directory and issue the following commands:

```
./configure −−prefix=/root/Desktop/netAI-0.1/
−−with-weka=/usr/local/bin/
−−with-netmate=/root/Desktop/netmate-0.9.4/
−−with-netmate-src=/root/Desktop/netmate-0.9.4/
gmake
gmake install
```

We are assuming here that both netmate and netAI have been extracted to the root user's desktop and installed there itself in their respective directories.

C. Installing the Skype Traffic Detector

The Skype Traffic Detector can be downloaded from http:// ... . Its installation is relatively simple; cd into the directory where the tarball has been downloaded and execute

```
tar -xvzf Skypetd.tar.gz
```

This will extract the Skype Traffic Detector (Skype.pyw) and the best feature file (BestFeaturesA.arff). Before running the Skype Detector, make sure that Python 2.5.2 or higher is installed and that the TCL/TK libraries are available as well. If the TCL/TK libraries are not present they can be installed via the FreeBSD ports tree. Issue the following commands to install TCL/TK for Python.

```
cd /usr/ports/x11-toolkits/py-tkinter/
make && make install
```

A directory where the software will access files from netAI's output needs to be created as well. To accomplish this, become the root user and execute

```
cd /usr/
```

TABLE I
COMMANDS FOR CAPTURING NETWORK TRAFFIC

Capture Type	**Command**
Live Capture	netmate -i em0
Trace File	netmate -f /root/Desktop/tcp.pcap

```
mkdir Skype
```

The Skype Traffic Detector can only be run by root due to privacy concerns.

IV. Starting a Trace

A training dataset is required in the *.arff* format to start with. The **manual pages** of netmate and netAI can be referred to for a complete list of arguments that can be used to run the trace. Both netAI and netmate have to be running simultaneously for the trace to work. The most efficient way to achieve this is to execute netmate in one shell window and netAI in another. There will be no results if they are run independently of each other. In our research, we mostly used trace files to run our simulations since we were building upon already accomplished work by other CAIA staff.

They had already built the training files needed to identify Skype traffic. The C45 algorithm, or J48 as it is known in WEKA, was used to determine the best features that could be used to identify Skype traffic flows and the training file was generated using that algorithm. The Naive Bayes algorithm was also tested but the C45 algorithm proved to be more accurate in classifying Skype traffic flows.

A. Start netmate

Netmate can be used to capture data from a trace-file (*-f <filename>*) (e.g. a tcp-dump) or from a live capture (*-i <interface>*) directly from a connected network interface. If capture from a tracefile is chosen, run netAI first (see below for details). Tables I and II demonstrate what commands need to be issued and what output should be expected.

The *-r <filename>* can be used to define any rules that the user wishes to process while running the capture such as `netmate -f /root/Desktop/tcp.pcap -r /root/Desktop/netAI-rules.xml`, depending on the location of your files, the commands would vary accordingly. The designation of your network interface card, whether emX or ethX, can be found by running `ifconfig` command.

TABLE II
SAMPLE RESULTS FROM NETMATE CAPTURE

For Live Capture
Listening on em0 Up and Running
For Trace File
NetMate version 0.9.4, (c) 2003-2004 Fraunhofer Institute FOKUS, Germany netmate logfile is: /usr/local/var/log/netmate.log Listening on: /root/Desktop/tcp.dump Up and running. End of capture file reached Classifier Dump packets: 18591 bytes: 2564968 rules: 2 NetTapPcap dump: packets: 18604 bytes: 2565656 dropped packets: 0 Terminating netmate.

TABLE III
ARGUMENTS REQUIRED TO START NETAI

-c	the class index
-t	the training file
-A	the attributes you want to run the algorithm on
-o	needed to write output to a file
-Y	the attributes you want in the output

B. Start netAI

netAI can also be configured to capture either from a tracefile or directly from an interface. A basic understanding of how WEKA works is a must and a knowledge of how data is stored in the *.arff* files is also essential to get netAI to function properly. netAI requires more arguments than netmate.

Arguments needed to make netAI work are shown in Table III, and an example command is demonstrated in Table IV. Table V shows the output after executing the command.

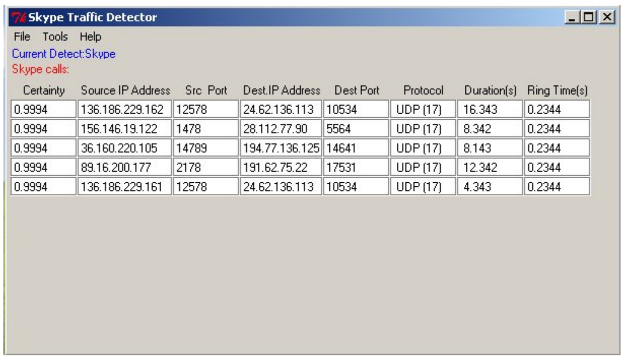
In this scenario, the results from netAI are written to a file, *Skype.test*, and we are using the training data file *BestFeaturesA.arff*. *weka.classifiers.trees.J48* is the algorithm being used to train on the data obtained from netmate.

TABLE IV
EXAMPLE OF HOW TO START NETAI FROM THE COMMAND LINE

```
java -jar /root/netAI-0.1/src/netAI_CL/netAI_CL.jar
weka.classifiers.trees.J48
-c 1 -t /usr/Skype/BestFeaturesA.arff
-A 4,5,6,7,8,9,10,11,12,13,14,15,16,17,18
-o /usr/Skype/Skype.test
-Y 0,1,2,3,4,5,6,8,9,10,11,12,13,14,15,16,17,18,19,20,
21,22,23,24,25,26,27,28,29,30
```

TABLE V
A SAMPLE OUTPUT FROM NETAI AFTER IT IS SUCCESSFULLY STARTED

```
netAI CL Version 0.1
Outputting features: 0 1 2 3 4 ... 16 17 18 19 20
Output File: /usr/Skype/data.test
Training File: /usr/Skype/BestFeaturesA.arff
Classification Algorithm: weka.classifiers.trees.J48
Model created in 1.99 seconds
Ready to receive from: TCP export
```



Current Detect:Skype
Skype calls:

Certainty	Source IP Address	Src Port	Dest.IP Address	Dest Port	Protocol	Duration(s)	Ring Time(s)
0.9994	136.186.229.162	12578	24.62.136.113	10534	UDP (17)	16.343	0.2344
0.9994	156.146.19.122	1478	28.112.77.90	5564	UDP (17)	8.342	0.2344
0.9994	36.160.220.105	14789	194.77.136.125	14641	UDP (17)	8.143	0.2344
0.9994	89.16.200.177	2178	191.62.75.22	17531	UDP (17)	12.342	0.2344
0.9994	136.186.229.161	12578	24.62.136.113	10534	UDP (17)	4.343	0.2344

Fig. 2. Skype Traffic Detector

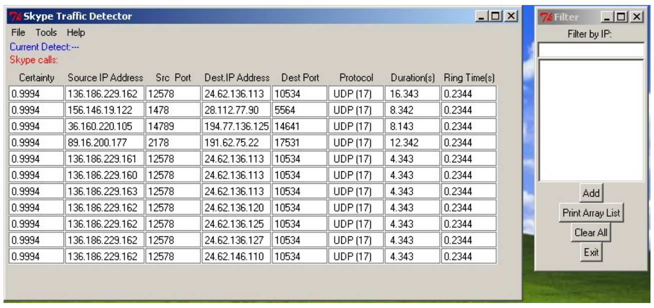
C. Skype Traffic Detector

The main function of the Skype Traffic Detector is to display and filter out the results output from netAI on a clean graphical user interface, making it easier for the user to interpret and work on. **Note that it is important that netAI outputs its results to the correct location and uses the proper filename for the Skype Traffic Detector to work.** The first thing that needs to be done to run the detector is to change its file mode to executable so that it can be run. To do so, issue the following commands in sequence:

```
cd into directory where program is stored
./Skype.pyw (to execute it)
```

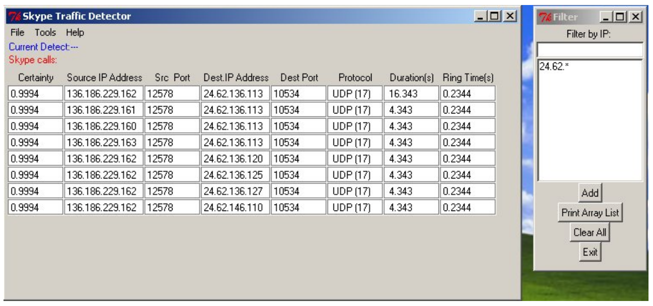
The output file from netAI needs to be stored in the directory */usr/Skype/* and should have the filename *Skype.test*.

Once the program is up and running the results can immediately be seen on the screen as illustrated in Figure



Certainty	Source IP Address	Src Port	Dest.IP Address	Dest Port	Protocol	Duration(s)	Ring Time(s)
0.9994	136.186.229.162	12578	24.62.136.113	10534	UDP (17)	16.343	0.2344
0.9994	156.146.19.122	1478	28.112.77.90	5564	UDP (17)	8.342	0.2344
0.9994	36.160.220.105	14789	194.77.136.125	14641	UDP (17)	8.143	0.2344
0.9994	89.16.200.177	2178	191.62.75.22	17531	UDP (17)	12.342	0.2344
0.9994	136.186.229.161	12578	24.62.136.113	10534	UDP (17)	4.343	0.2344
0.9994	136.186.229.160	12578	24.62.136.113	10534	UDP (17)	4.343	0.2344
0.9994	136.186.229.163	12578	24.62.136.113	10534	UDP (17)	4.343	0.2344
0.9994	136.186.229.162	12578	24.62.136.120	10534	UDP (17)	4.343	0.2344
0.9994	136.186.229.162	12578	24.62.136.125	10534	UDP (17)	4.343	0.2344
0.9994	136.186.229.162	12578	24.62.136.127	10534	UDP (17)	4.343	0.2344
0.9994	136.186.229.162	12578	24.62.146.110	10534	UDP (17)	4.343	0.2344

Fig. 3. Skype Traffic Detector and filter dialog



Certainty	Source IP Address	Src Port	Dest.IP Address	Dest Port	Protocol	Duration(s)	Ring Time(s)
0.9994	136.186.229.162	12578	24.62.136.113	10534	UDP (17)	16.343	0.2344
0.9994	136.186.229.161	12578	24.62.136.113	10534	UDP (17)	4.343	0.2344
0.9994	136.186.229.160	12578	24.62.136.113	10534	UDP (17)	4.343	0.2344
0.9994	136.186.229.163	12578	24.62.136.113	10534	UDP (17)	4.343	0.2344
0.9994	136.186.229.162	12578	24.62.136.120	10534	UDP (17)	4.343	0.2344
0.9994	136.186.229.162	12578	24.62.136.125	10534	UDP (17)	4.343	0.2344
0.9994	136.186.229.162	12578	24.62.136.127	10534	UDP (17)	4.343	0.2344
0.9994	136.186.229.162	12578	24.62.146.110	10534	UDP (17)	4.343	0.2344

Fig. 4. Skype Traffic Detector filtering

2. The program can be set to display records from a chosen list of IPs only. To achieve this, click on the "Tools" menu and then select the "Filter" command. A window identical to that shown in Figure 3 appears. Type in the IP addresses desired and click on the "Add" button to add them to the filter list and the program will start filtering immediately, as illustrated in Figure 4. In case an entry mistake is made, the whole list can be cleared and the filtering criteria must be re-entered. It is also possible to use wildcard masks to filter out IPs. Suppose IPs starting with 24.62 need to filtered; 24.62.* should be input in the entry textbox and added to the list. The detector will then filter all IPs in the range 24.62.XXX.XXX.

Specific fields can be displayed by entering command line arguments, such as `./Skype.pyw 0 1 2 3`. This will cause the GUI to display four fields. The arguments for the various fields in the program are shown in Table VI.

V. Conclusion

In this technical report, we have illustrated how machine learning tools can be set up and how we can filter out VoIP calls made on Skype using the Skype Traffic Detector. Our objective to design a system that could display and update records from netAI in real-time was successfully achieved. We were able to implement functions to eliminate repeated and redundant information all by providing a way to filter out records based on their IP address. Even though the Skype Traffic Detector was designed to filter out Skype VoIP calls, its code can be altered to suit other applications where network traffic capture is involved. Further development in machine learning techniques means that we will be able to create more reliable and powerful filtering systems.

Future development on the Skype Traffic Detector can include more filtering options and the ability to print to softcopy or hardcopy. A next release can also include loading netAI files which have been saved before and therefore conducting the trace on those files instead of having to run netAI and netmate at the same time as the Detector.

TABLE VI
Fields and their associated numbers

Field	Argument
certainty	0
flowID	1
srcIP	2
srcPort	3
desIP	4
desPort	5
protocol	6
datetimeStart	7
datetimeEnd	8
timeOut	9
startTime	10
endTime	11
duration	12
ringTime	13
end	14

References

[1] Wikipedia, "Machine learning — Wikipedia, the free encyclopedia," 2009, [Online; accessed 19-January-2009]. [Online]. Available: http://en.wikipedia.org/wiki/Machine_learning
[2] ——, "Pcap — Wikipedia, the free encyclopedia," 2009, [Online; accessed 27-January-2009]. [Online]. Available: http://en.wikipedia.org/wiki/Pcap
[3] T. U. of Waikato, "Weka 3: Data mining software in java," 2009, [Online; accessed 19-January-2009]. [Online]. Available: http://www.cs.waikato.ac.nz/~ml/weka/
[4] S. Zander and N. Williams, "netai install," 2006, [Online; accessed 19-January-2009]. [Online]. Available: http://caia.swin.edu.au/urp/dstc/netai/INSTALL
[5] C. Schmoll and S. Zander, "Netmate — user and developer manual," 2004.