OpenPacketPro: A libpcap Extension Framework for Sniffing Outgoing Traffic

Jonathan Wald jwald@wustl.edu and **Jason Zigelbaum** jczigelb@wustl.edu (A project report written under the guidance of Prof. Raj Jain)

Download

Table of Content:

1 Introduction

1.1 What is OpenPacketPro?

OpenPacketPro is an open source framework that utilizes the libpcap library.

We want to give developers a platform to work with packets and the libpcap library in order to perform interesting and useful services. OpenPacketPro is intended to be a hub for fun and exciting experiments that involve packet capturing.

In order to demonstrate the power of the libpcap library we have developed two useful services.

OpenPacketPro's Demo Services :

1. Stopping outbound communication of a given program.
2. Sifting out MD5 encrypted packets and decrypting them in real time.

In the following sections, you will learn how to perform these tasks. To open the application, locate it in a terminal window and run the application in **privledged** mode (Details will be covered in the "how to use" section).

2 Design Documentation

This section documents the details of the application's design and some of the important decisions that we made. Specifically, we talk about the technologies that were leveraged in our implementation, including libpcap for packet capture and Nokia's QT library for the application's user interface and event handling. Also discussed is our hashing method of target detection, blocking target communication, our method for password detection and decryption, and multi-threading.

2.1 Languages and Frameworks

The application was implemented in C++ and made significant use of the libpcap and QT libraries. It should also be noted that we leveraged example code from tcpdump.org (the maintainer of libpcap) and QT tutorials.

2.1.1 Libpcap

The application's packet sniffing engine is built using the libpcap library, the same library used for popular tools Wireshark and Tcpdump. Libpcap works by inspecting packets that are placed in the kernel's packet filter by a network card driver. When a packet comes in, the network card checks to see if its own IP address is the destination address, and if so, signals for an interrupt. A signal handler in the form of the network card driver copies the data from the card to a buffer in kernel space for consumption by higher level protocols. The network card driver also stores a copy of the data in a buffer within the kernel called the packet filter; libpcap provides a framework for interacting with the packets stored in this buffer.

2.1.2 QT

We selected Nokia's open-source QT framework for the implementation of our user interface and event handling. QT is a popular, professional quality framework with a wide range of applications ranging from desktop applications, web development, and mobile application. We primarily used QT's user interface libraries, but also made use of its built-in threading functionality and event handling. QT is portable, meaning that our application and user interface will function on most platforms that support libpcap (we tested on Ubuntu and Mac OSX).

2.1.3 Rainbow Tables

A Rainbow Table is essentially a lookup table of decrypted MD5 hash values. With the help of a rainbow table, decrypting MD5 Hashes becomes a matter of looking up an entry in this large table. The larger the table the better. In our implementation, we use an API call to a free-to-use rainbow table at www.decrypt-md5.com.

2.2 Design Elements

This section outlines critical elements of the design and the decisions that went into them. The topics include our detection of target communication, communication blocking, password sniffing, and multi-threading.

2.2.1 Target Detection by Hashing and Reverse DNS

One of the primary services of the application is the ability to detect network communication of interest, without knowing a target's host name or IP address. This is useful, for example, when a user wants to crack the validation mechanism of a piece of software but has no idea with what host the software may attempt to communicate. We solved this problem by segmenting our packet sniffing into two phases - a training phase and a target detection phase.

At any given time, a modern computer user may be maintaining dozens of connections with hosts across the world. Many users now employ cloud storage or updating systems which are continuously communicating with their local machine. We want to filter out the benign packets whose destinations are not intended to be blocked. These unimportant packets may include current TCP connections, or other applications which must send outgoing data constantly and the user does not want blocked.To remove this noise, we require that the user first allow our application to undergo at 15 second training period, which acts as a sieve for network communication. During this period, the target application is turned off and we place all IP address with which we communicate into a map structure. Ideally this data structure would use hashing to minimize lookup time, we used the C++ Standard Template Library's map structure, which is actually implemented as a binary tree. After the training period, the target application is finally executed, and we ignore all "normal" communication by doing a lookup on each host that we communicate with and seeing if there is a collision. After this process, we have most likely narrowed the potential target IP address to a small set of less than a dozen suspects. We display these addresses, along with any host names that we can find through a reverse DNS lookup, to the user and allow them to select which ones they want to attack.

2.2.2 Host File and Routing Table Modification

The application offers two methods of blocking a software validation attempt. After a user selects the hosts that it wants to attack, he or she is presented with a dialog box asking them which of the methods that wish to employ. The first allows a user to block a specific host name by modifying the machine's host file. A host file allows a user to set how a specific host name is resolved to an IP address. When a program attempts to connect to a host, it first checks the machine's host file. If there is no IP address for that host name in the host file, the program then resorts to using DNS. Thus, in the case where a piece of software is authenticating itself with a hard-coded host name, our application will prevent packets from reaching that host by forcing that host name to resolve to the address of the local machine.

More likely, software attempting to authenticate will not try to connect with a host name but rather some sort of static IP address. In this scenario, we can prevent communication by identifying that IP address and modifying our routing table such that packets destined for that address are routed back to the local host.

2.2.3 Password Sniffing

Our application can also leverage packet sniffing as a means for stealing user passwords. If a user enables password sniffing, the application will attempt to detect password requests by parsing incoming HTML. Each incoming packet is searched variations of the word "password". If such a packet is detected, the all communication with that host is marked as relevant using a mapping scheme similar to that which was used in target detection. When the user then responds to that host in any way, we inspect the contents of the response packet. In the current implementation, we detect passwords simply by checking for simple form submission HTML that looks something like "password=". After identifying where the password is, we then make an API call to an online rainbow table database (we used www.decrypt-md5.com) in order to decrpypt the password. In the future, our application could be extended to support the decryption of more sophisticated password algorithms, such as SHA-2. Furthermore, this attack could be especially effective if used in conjunction with libpcap's promiscuous mode, which enables the sniffing of packets not just on the local machine but the entire local network.

In order to test our implementation, we developed a web application. The application takes in a username and a password field and encrypts the password client-side using an MD5 hash algorithm. The script then sends the password to a database so we can verify it was sent. This web-application served as a use-case for our MD5 decryption application.

2.2.4 Multithreading

We utilized QT's built in multi-threading library to make our user interface fast and responsive. When a user requests the use of the packet sniffing engine for software authentication cracking or password sniffing, the application initializes a new thread and delegates the responsibility of sniffing the packets to it. By decoupling this processing from the application's main thread, which is responsible for handling user events and making the interface responsive, we maximized both the application's performance and usability.

3. Using OpenPacketPro

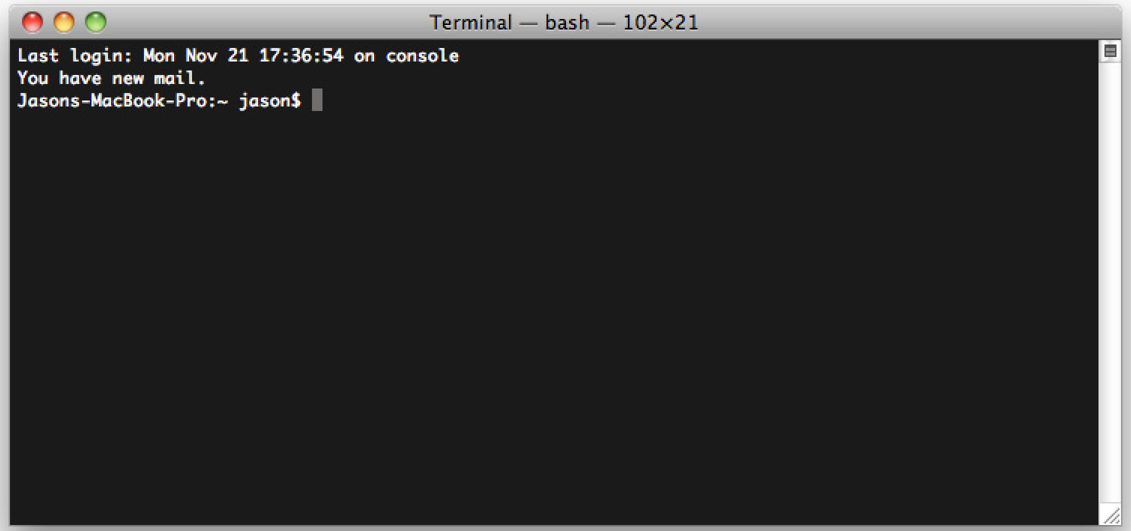
Download the source code

Once the source code is posted, it will be available online for download at our project page within Professor Raj Jain's website.

Open the folder in your terminal

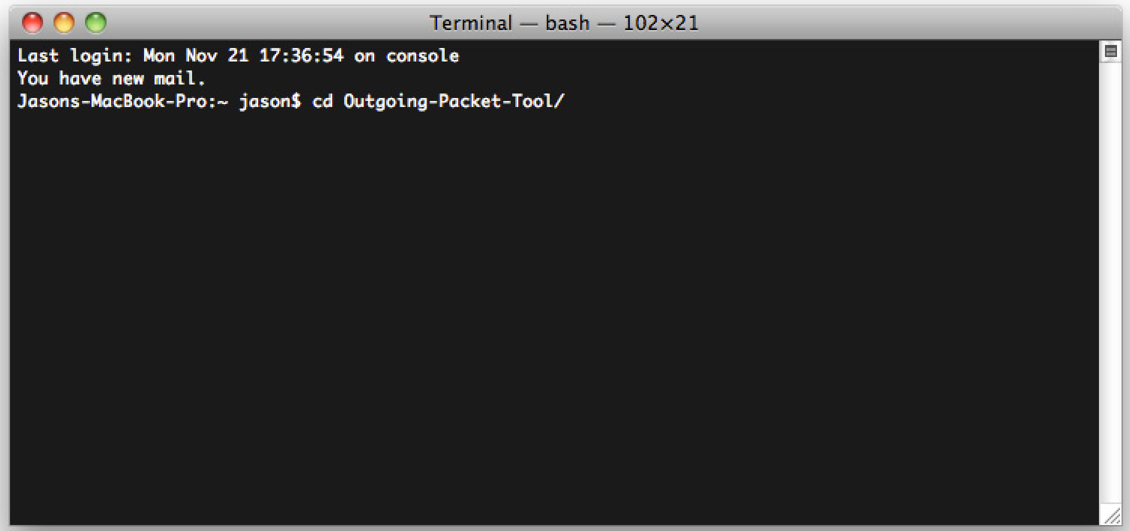
In order to open the application, you must navigate to the folder via the terminal. This can be done by dragging the folder into the terminal if you are in Mac OSX. If you are using some form of linux then you probably know how to navigate to the application's folder. But I will show you anyhow...

Open your terminal window of choice



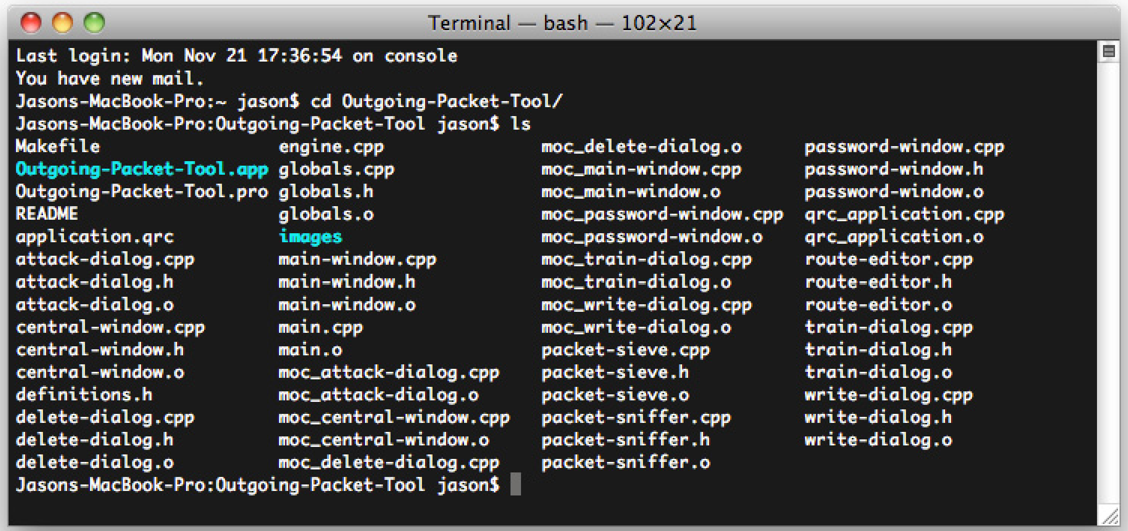
```
Last login: Mon Nov 21 17:36:54 on console
You have new mail.
Jasons-MacBook-Pro:~ jason$
```

Go to the application executable



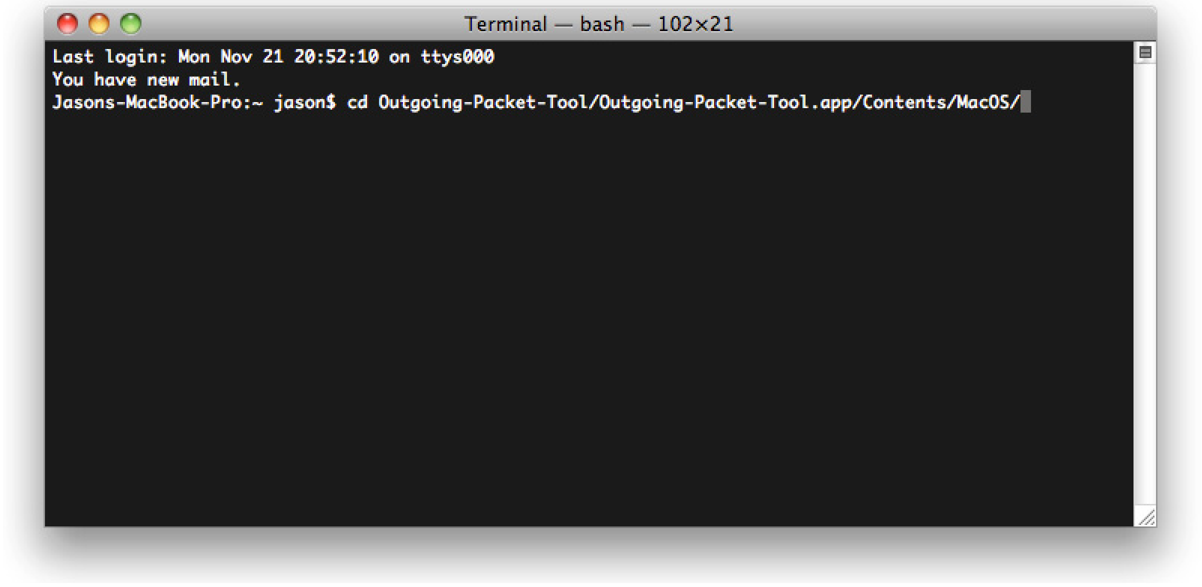
```
Last login: Mon Nov 21 17:36:54 on console
You have new mail.
Jasons-MacBook-Pro:~ jason$ cd Outgoing-Packet-Tool/
```

Once inside, you should see something like the image below inside of the directory. Type "ls" to see the contents for yourself



```
Last login: Mon Nov 21 17:36:54 on console
You have new mail.
Jasons-MacBook-Pro:~ jason$ cd Outgoing-Packet-Tool/
Jasons-MacBook-Pro:Outgoing-Packet-Tool jason$ ls
Makefile                 engine.cpp               moc_delete-dialog.o      password-window.cpp
Outgoing-Packet-Tool.app globals.cpp              moc_main-window.cpp      password-window.h
Outgoing-Packet-Tool.pro globals.h                moc_main-window.o        password-window.o
README                   globals.o                moc_password-window.cpp  qrc_application.cpp
application.qrc          images                   moc_password-window.o    qrc_application.o
attack-dialog.cpp        main-window.cpp          moc_train-dialog.cpp     route-editor.cpp
attack-dialog.h          main-window.h            moc_train-dialog.o       route-editor.h
attack-dialog.o          main-window.o            moc_write-dialog.cpp     route-editor.o
central-window.cpp       main.cpp                 moc_write-dialog.o       train-dialog.cpp
central-window.h         main.o                   packet-sieve.cpp         train-dialog.h
central-window.o         moc_attack-dialog.cpp    packet-sieve.h           train-dialog.o
definitions.h            moc_attack-dialog.o      packet-sieve.o           write-dialog.cpp
delete-dialog.cpp        moc_central-window.cpp   packet-sniffer.cpp       write-dialog.h
delete-dialog.h          moc_central-window.o     packet-sniffer.h         write-dialog.o
delete-dialog.o          moc_delete-dialog.cpp    packet-sniffer.o
Jasons-MacBook-Pro:Outgoing-Packet-Tool jason$ 
```

If you are running a mac, you must cd to "/Outgoing-Packet-Tool.app/Contents/MacOS" in order to reach the executable. This is because QT (our GUI framework) uses this convention.



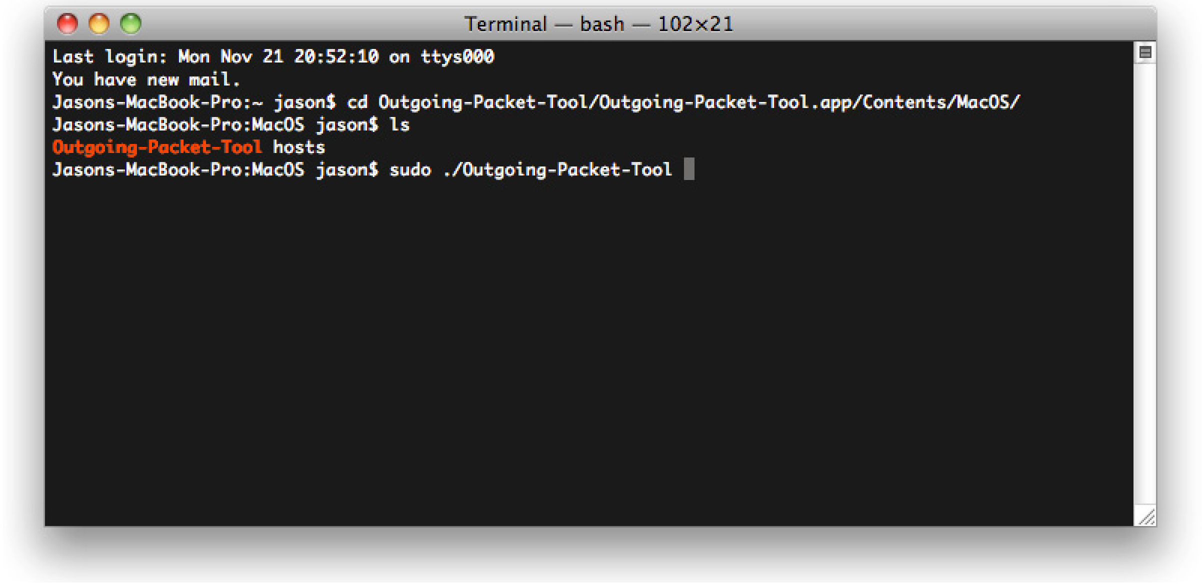
```
Last login: Mon Nov 21 20:52:10 on ttys000
You have new mail.
Jasons-MacBook-Pro:~ jason$ cd Outgoing-Packet-Tool/Outgoing-Packet-Tool.app/Contents/MacOS/
```

Run the executable in privileged mode.

Once inside the folder which contains the executable you will want to run it (obviously). To do this, you must use privileged mode.

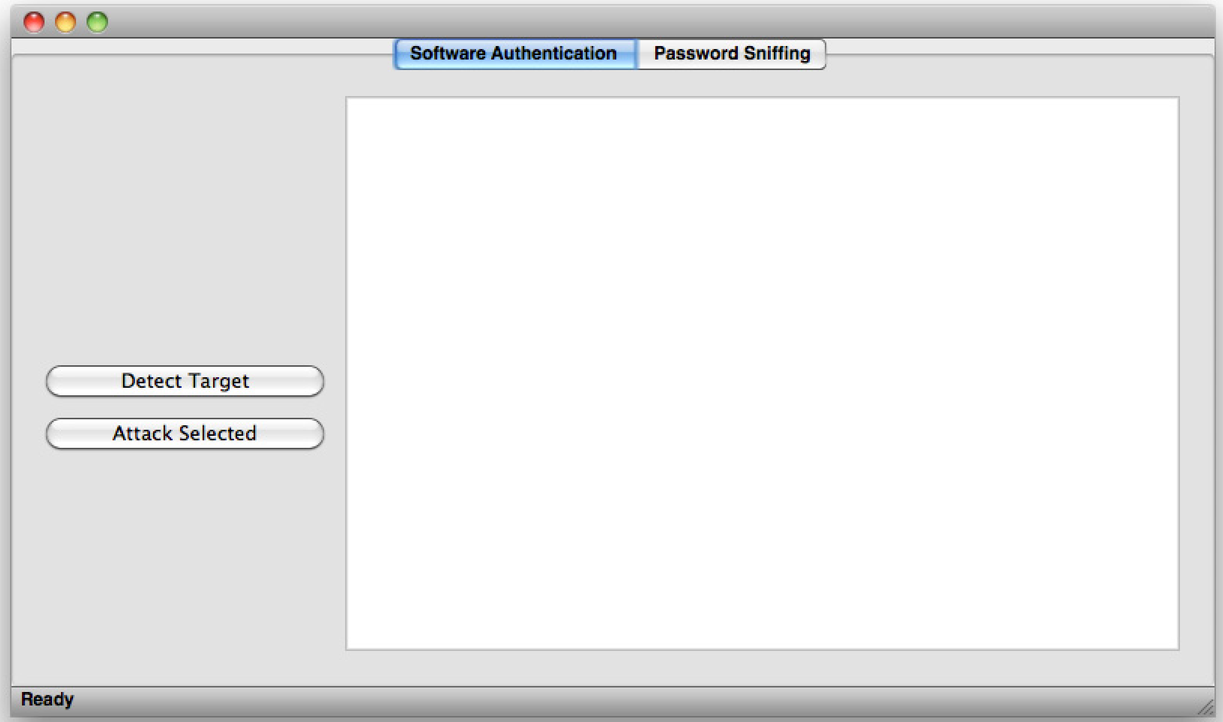
Technically, you do not have to run the application in privileged mode, but in order to edit your routing table (and effectively use the block communication service) you must be in privileged mode.

So, unless you want to limit the functionality of the app, you should run the program with privileges. To do this, once you are in the folder where the OpenPacketPro executable is located, type "sudo ./Outgoing-Packet-Tool". You will then be prompted for your password. Type it in and OpenPacketPro will start up.



```
Last login: Mon Nov 21 20:52:10 on ttys000
You have new mail.
Jasons-MacBook-Pro:~ jason$ cd Outgoing-Packet-Tool/Outgoing-Packet-Tool.app/Contents/MacOS/
Jasons-MacBook-Pro:MacOS jason$ ls
Outgoing-Packet-Tool hosts
Jasons-MacBook-Pro:MacOS jason$ sudo ./Outgoing-Packet-Tool
```

Once the program is running, you should see the following window pop open.



Decide which OpenPacketPro service you would like to use.

The available options are located on top of the main window pane.

3.1 OpenPacketPro Mode 1

Software Authentication

About

This service allows a user to block outgoing communications from his/her computer to a certain destination ip address (or multiple addresses). The practical use of this is to stop communication from certain software to their respective authentication server. This means, that although your software (which may be expensive...) can be run even though it is not exactly street legal. In fact, for most software you can easily get away with a less-than-perfect password.

Some bogus passwords will work because once **real** passwords get circulated (on google for instance) the software company may tag them on a "blacklist" after the product has shipped so that they cannot be used. The only catch is that these were once real passwords, meaning they work perfectly on a fresh version of the software and have to be marked by the black list in order to be invalid. This is where OpenPacketPro's Software Authentication (perhaps better dubbed software de authentication...) comes in handy. With this service, your software can no longer communicate to the outside world, and therefore cannot blacklist your easily found password.

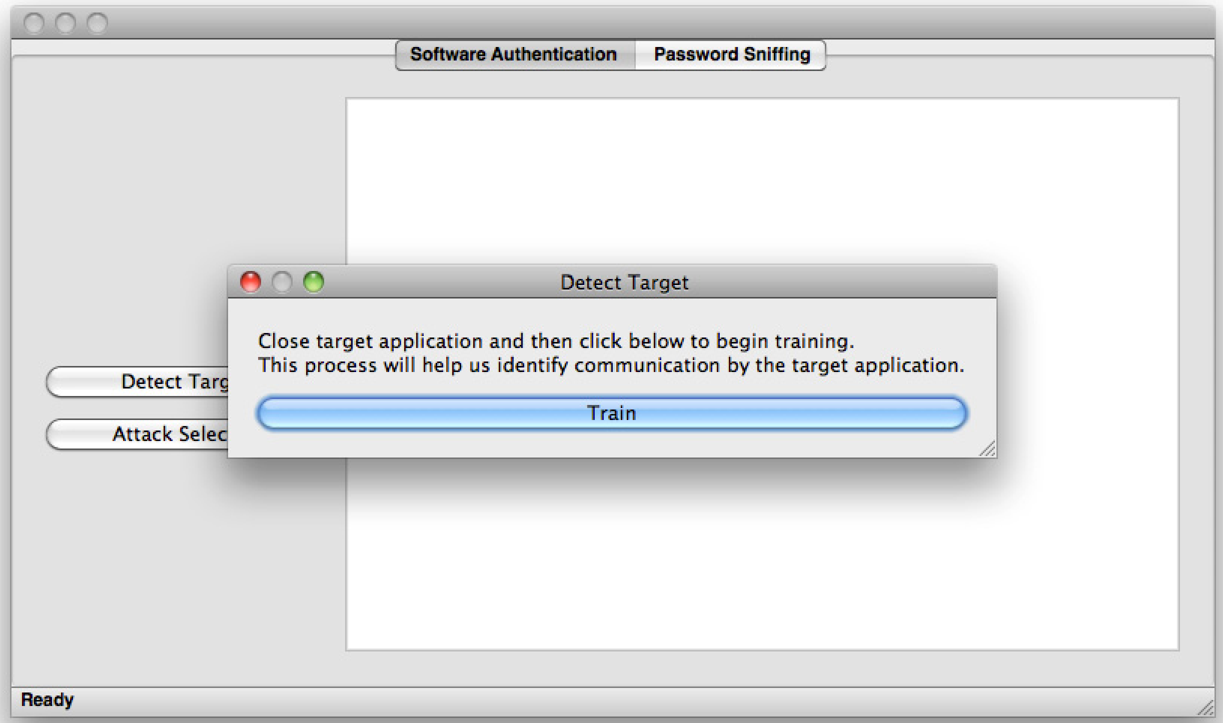
That may have been a lot to take in... so please follow along with an example. In a short time, you too can perform this attack.

How to Use

In order to block communication to an outgoing program, you will first need to know which program (or set of programs) you want to block. Once you have decided which program you would like to block, press the *Detect Target* button.

Step 1 : Detect Target

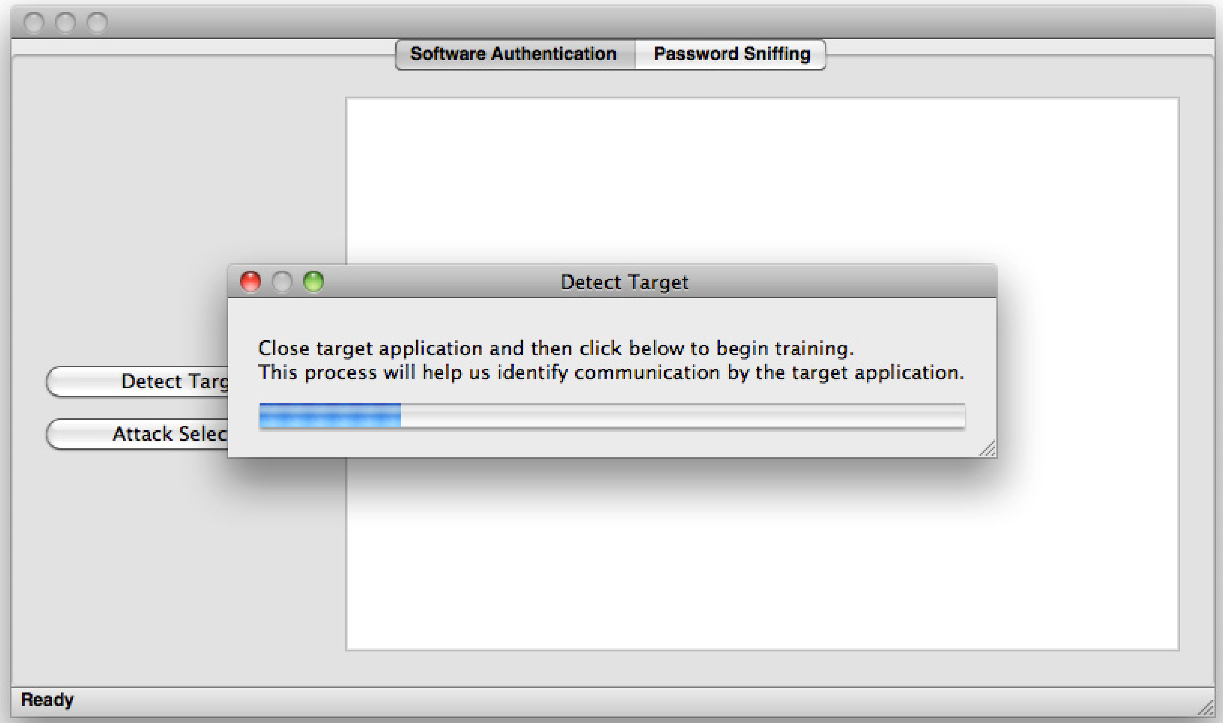
Press the "Detect Target" button. You will see a popup like the image below.



Close target application and then click below to begin training.
This process will help us identify communication by the target application.

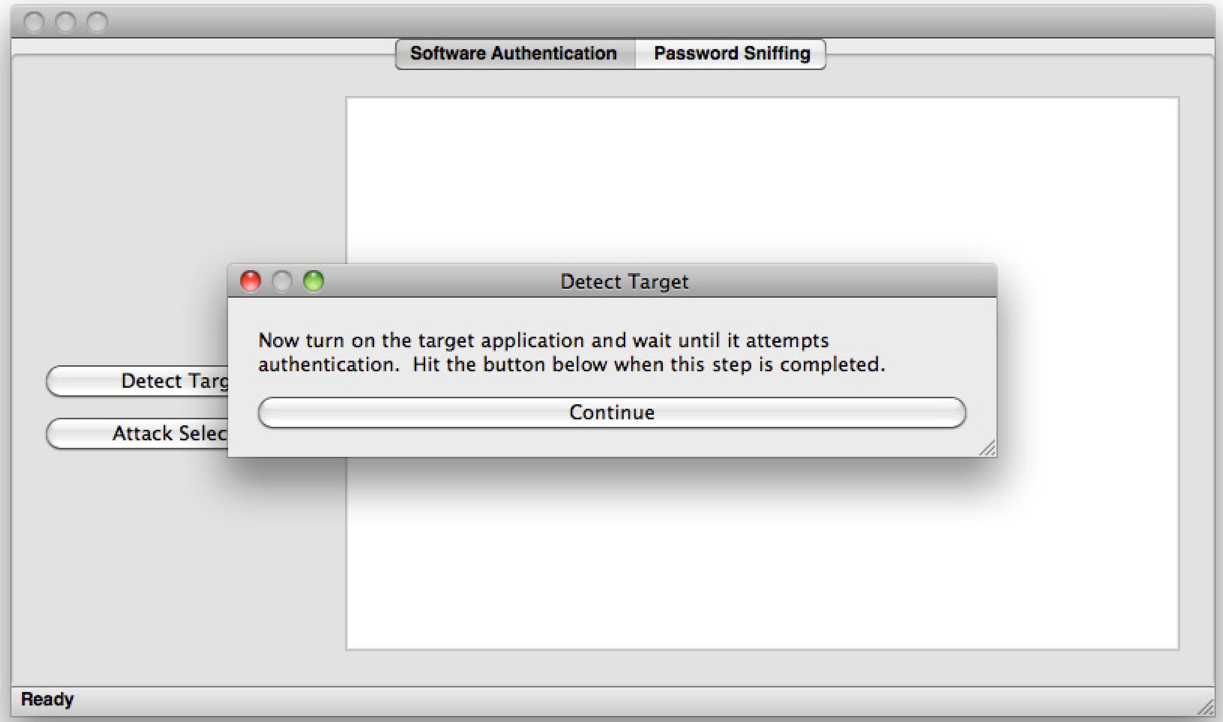
Step 1 : Training

When you press the train button, OpenPacketPro will listen to the packets you are sending out **before** you run your program (or set of programs). By listening for a designated period of time, we can sift out the packets which are being sent by programs that you do not want to block. It is crucial that you do not start your program during the training period or else it will be left out of the detection process. So, if your target software **is not running** press the train button. Once you do so, you will see something like the image below.



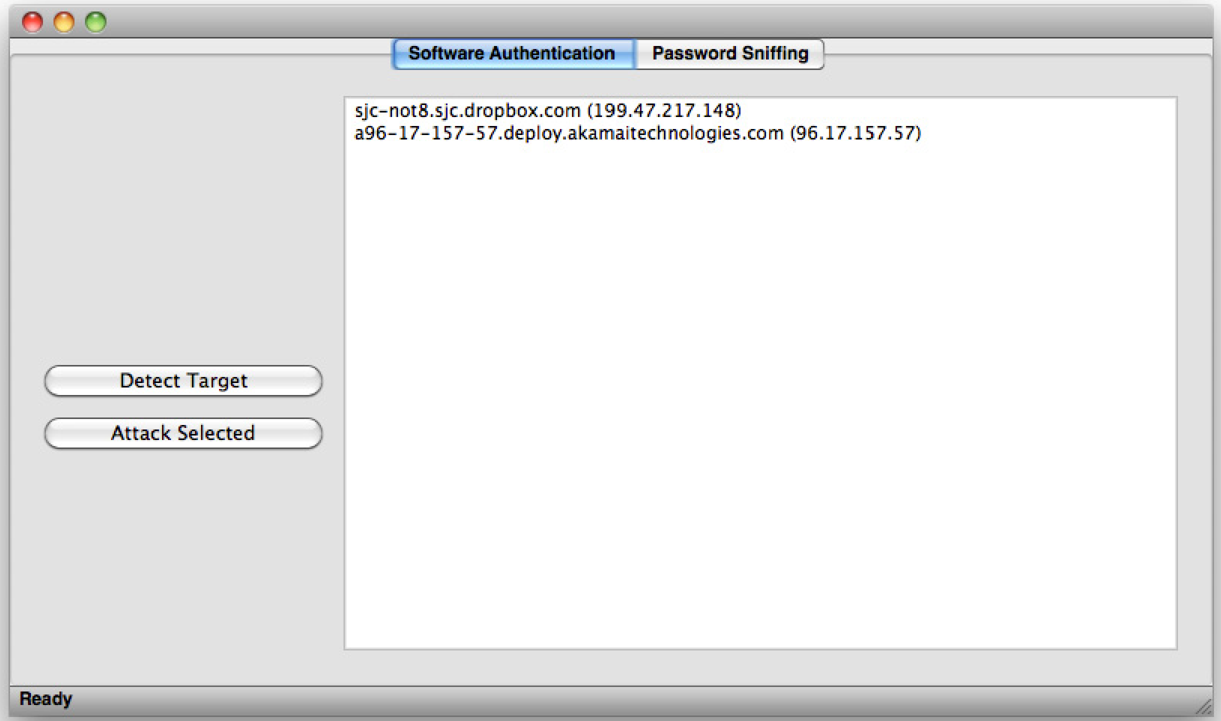
Close target application and then click below to begin training.
This process will help us identify communication by the target application.

Once the *training* has completed, you will be prompted to open your program. Please wait for your program to start up completely (so that it has time to contact the authentication server) before clicking continue.



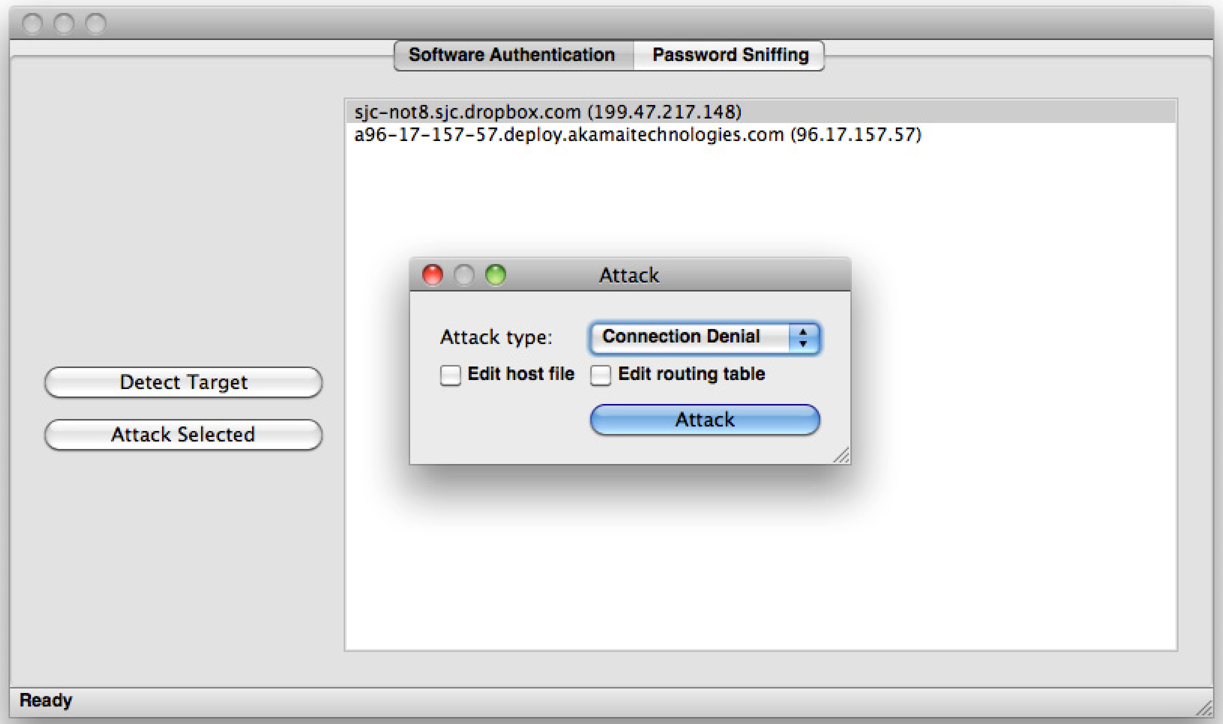
Now turn on the target application and wait until it attempts authentication. Hit the button below when this step is completed.

Once training has completed you will see a list of ip addresses that correlate to the destination addresses that the program is communicating with.



```
sjc-not8.sjc.dropbox.com (199.47.217.148)
a96-17-157-57.deploy.akamaitechnologies.com (96.17.157.57)
```

Select *all* the destination that correspond to your target software and select attack target. You will see a popup like the image below.



sjc-not8.sjc.dropbox.com (199.47.217.148)
a96-17-157-57.deploy.akamaitechnologies.com (96.17.157.57)

Attack type: Connection Denial
Edit host file
Edit routing table

Ready

Types of Attack

There are two types of attacks

1. Write the Domain Name Server entry to your hosts file.
2. Add route to routing table.

1. Writing to the hosts file will add a line to your hosts file. The route the selected DNS entry will be routed to your localhost (127.0.0.1) therefore denying all outgoing communication from that DNS address.

2. Adding a route to the routing table will add an entry to your routing table. The entry will route the selected IP address to your localhost (127.0.0.1) therefore denying all outgoing communication to that ip address. *Editing your routing table requires that you run OpenPacketPro in privileged mode.*

3.2 OpenPacketPro Mode 2

Cracking MD5 Hash Encrypted Values

About

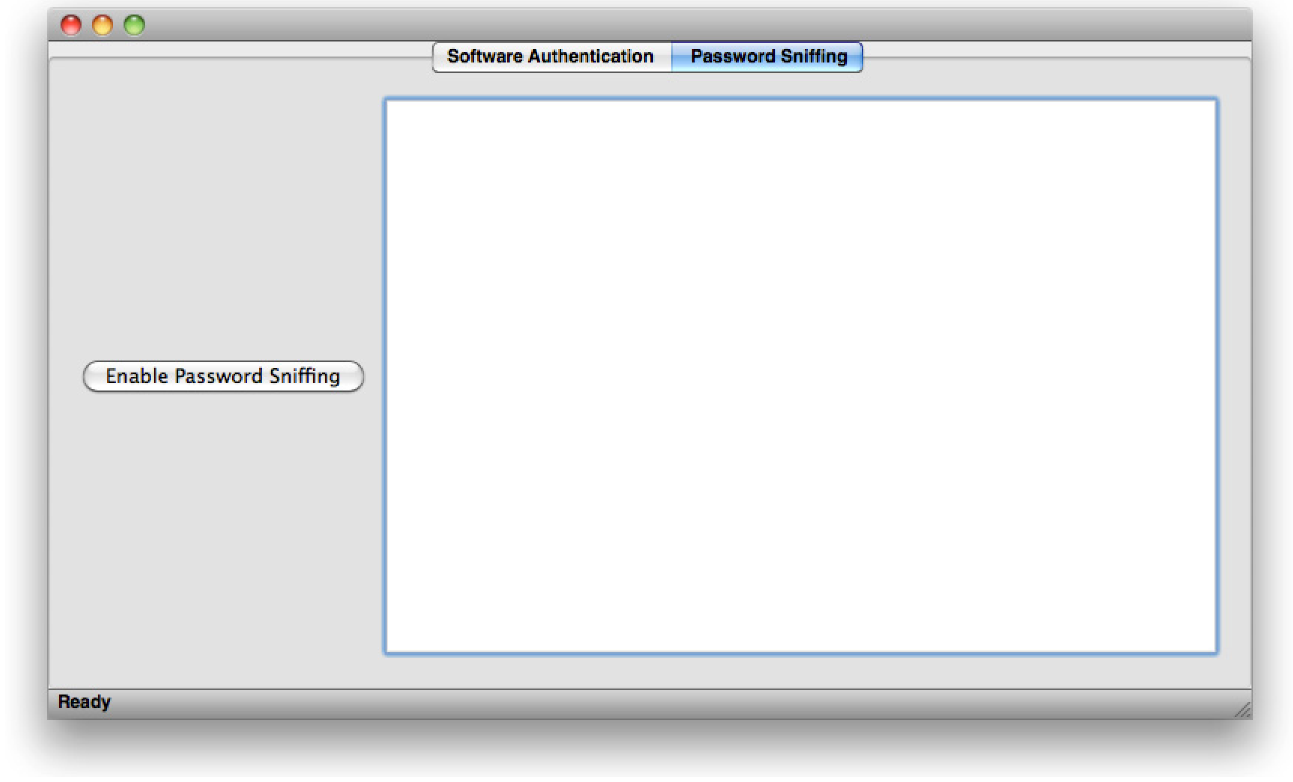
This service allows you to sniff and decrypt client side md5 hash encrypted passwords. This is done by selecting a key word which corresponds to a password field and honing in on the 32 bytes that follow that input word in the html packet. This will be explained further in our example.

Once found, the 32 bytes proceeding the key word are then decrypted through an API call to a rainbow table. A rainbow table is essentially a lookup table of known MD5 hashes and their corresponding plaintext values.

This methodology can be reused for different crackable encryptions (SHA-1 for instance) and for different key words (say variations of plaintext word "password" like "pass", "pw", etc...).

How to use

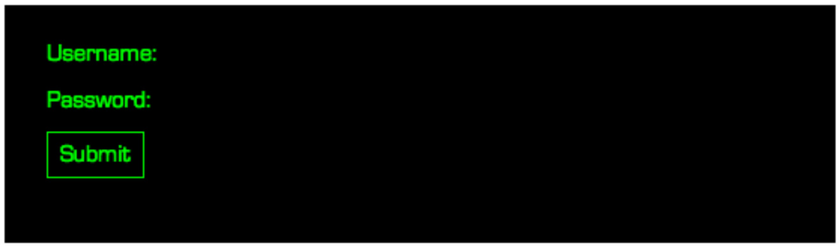
Select the Password Sniffing pane from the top of the main window. You will see the following pane.



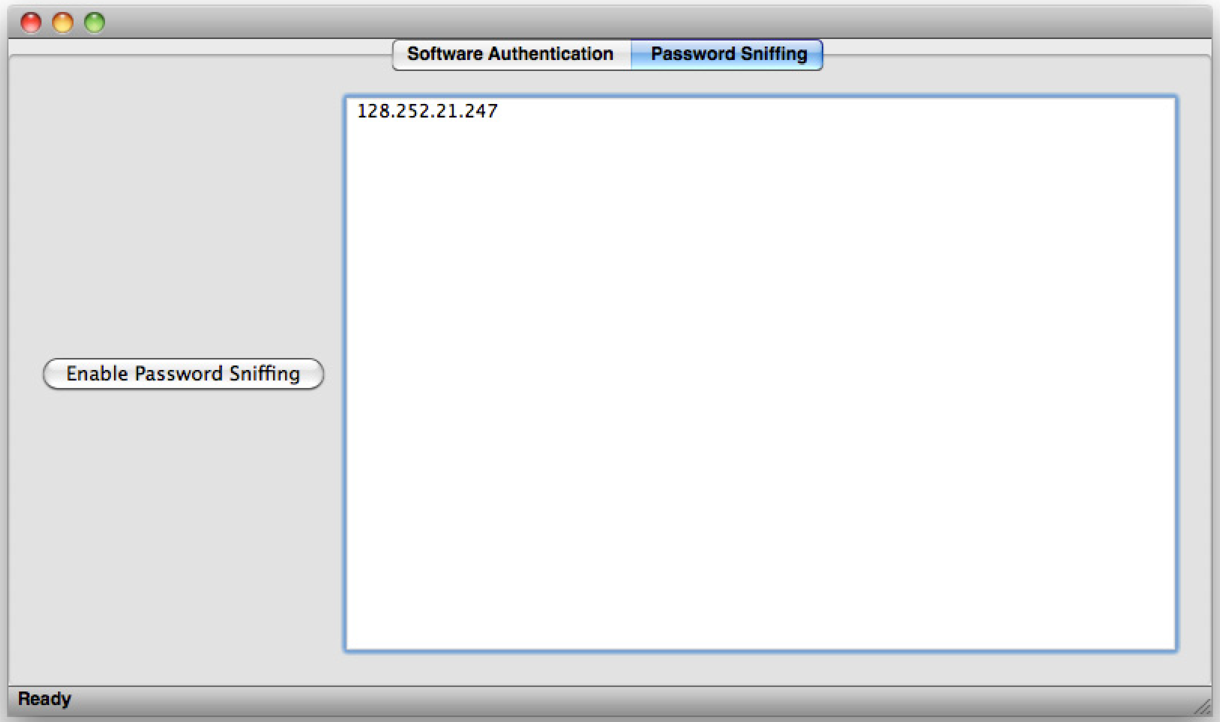
In order to use this part of the application you will want to first identify a webpage that uses MD5 encryption. For our example, we will be using a test website we produced. You can follow along by going here.

Once you have selected a page, go to the MD5 pane and select **Enable Password Sniffing**. Once pressed, the application will be listening for packets so you will want to clear your cache and open your target web-page now.

If you are using our home-made test page, you will see something like the image below.



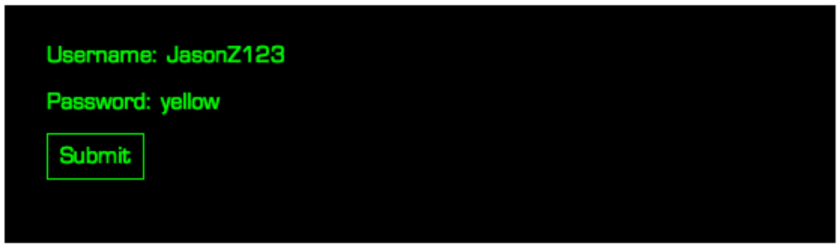
Once you have opened the page, you will see the ip address show up on the main window. This is the ip address of the webpage you will decrypt.



```
128.252.21.247
```

Ready

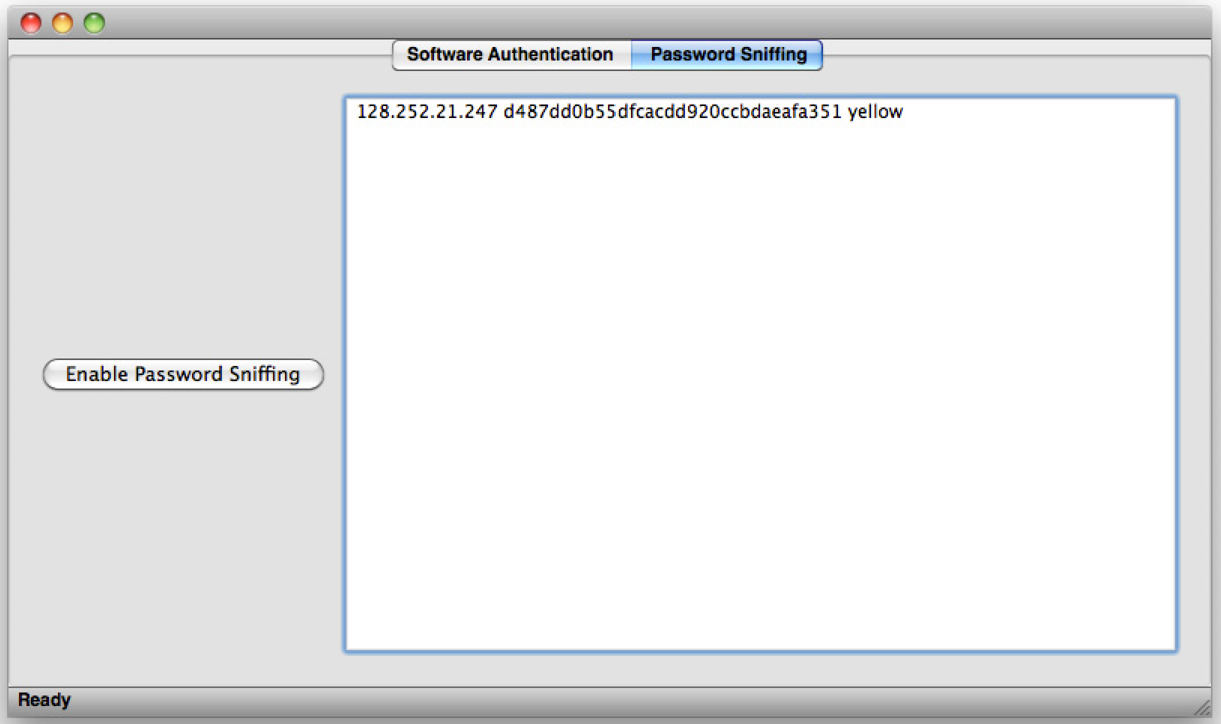
Enter your password, and if it is client-side MD5 encrypted, then your password will be detected and decrypted. Again, if you are using our website, you should do something similar to the image below.



Username: JasonZ123

Password: yellow

Once it is decrypted you will see the hash value and the decrypted value on your main window.



```
128.252.21.247 d487dd0b55dfcacdd920ccbdaeafa351 yellow
```

If you take a look at your terminal window, you should be able to get a better idea of how this works.

In the packet's payload is all of the html generated by the webpage. We scan all the payloads and if we find the key word inside any html, we listen to outgoing packets sent back to those webpages.

If we find the field in the outgoing packet back to the detected ip address we lookup the proceeding 32 bytes.

4. Conclusion

We hope you learned a lot from this guide and that you can find good uses for our project! Please feel free to develop extensions to the framework (that's what the word Open in OpenPacketPro is for).

5. Source Code

The code for this project can be downloaded from GitHub.

6. References

[1] "Programming with pcap". http://www.tcpdump.org/pcap.html
[2] Luis Martin Garcia. "Programming with libpcap â€" sniffing the network from our own application". Hackin9 Magazine, 3(2/2008), Feb 2008.
[3] "Packet Capture With libpcap and other Low Level Network Tricks". http://eecs.wsu.edu/~sshaikot/docs/lbpcap/libpcap-tutorial.pdf

Last Modified: December 6, 2011
This and other papers on latest advances in network security are available on line at http://www1.cse.wustl.edu/~jain/cse571-11/index.html
Back to Raj Jain's Home Page