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

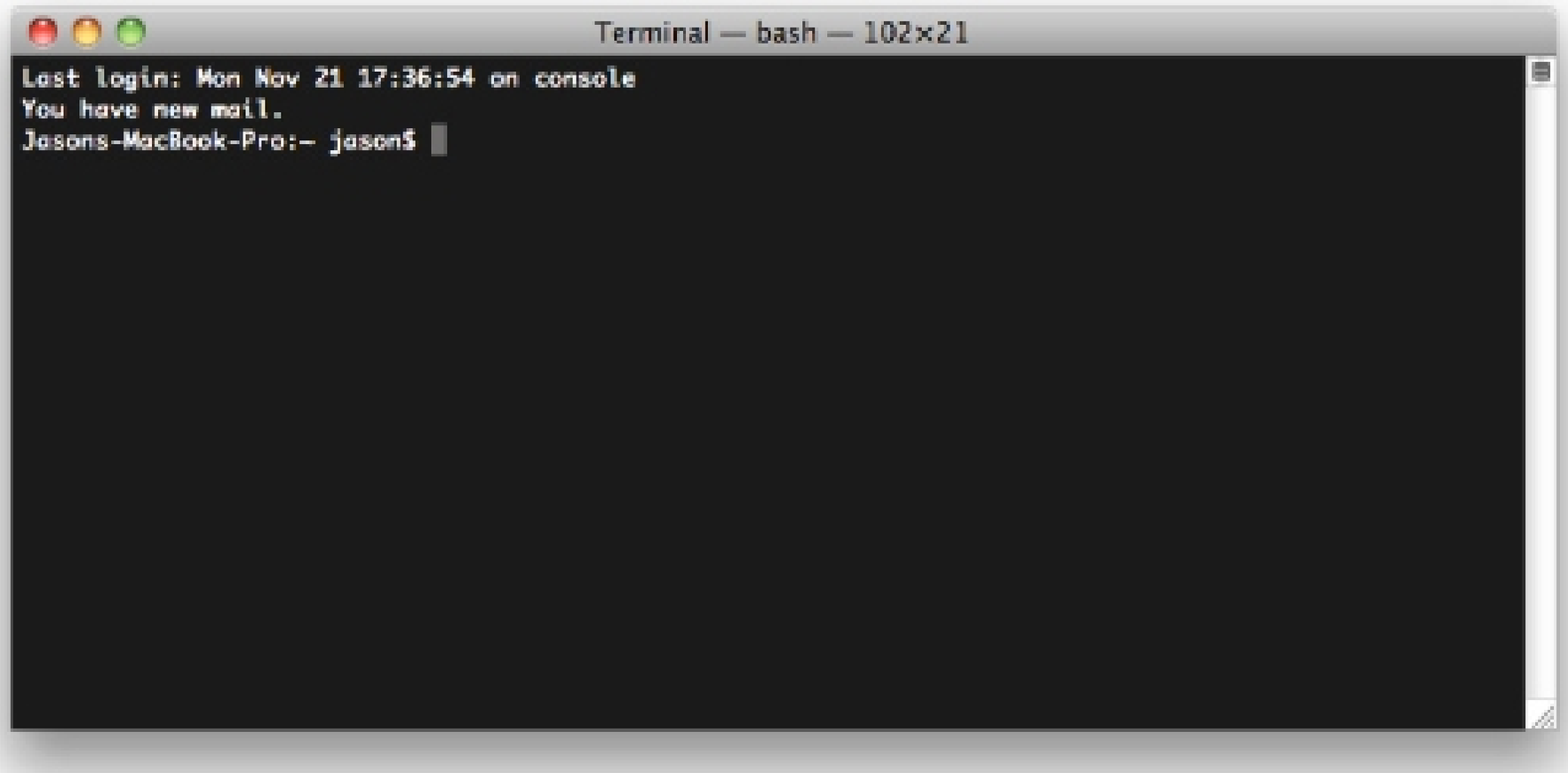
Download the source code

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



Go to the application executable