Politecnico di Milano
Scuola di Ingegneria Industriale e dell'Informazione
Corso di laurea in Ingegneria Informatica

PROGETTO DEL CORSO PIATTAFORME SOFTWARE PER LA RETE: GRAPHICAL TOOL FOR P0F

Andrea Corna	Matr. 757184
Lorenzo Fontana	Matr. 758924

Anno Accademico 2012-2013

0 - INTRODUCTION

P0f-QT is a C++ graphical wrapper developed with QT's libraries of the C language tool p0f; p0f can identify the system on machines that connect to your box, machines you connect to, and even machines that merely go through or near your box, but simply prints all the informations in a disorganized way.
P0f-QT, on the other hand, organizes all the informations and shows a list of hosts, with all the informations associated.

Now we talk about some of the main aspects of the project.

1 - P0F FUNCTIONING

We will describe the p0f's functioning. The program is started by command line, specifying all the parameters the user wants. As first operation, p0f sets all the *u8* pointers* which correspond to informations insert by command line. According to theese, there are two different types of execution mode.

- *daemon mode*: p0f runs as a background process;
- *not daemon mode*: p0f runs as a foreground process.

While in daemon mode you can only listen to an interface, in the other mode you can also read a pcap file in order to see his information. There are other additional functions, like setting the maximun number of hosts and connections to listen to, setting the promiscous mode of the net interface or writing a file log, who contains all the collected informations. You can read the file *README* in the repository to obtain a complete description of the tool's fuctions.

2 - LIBPCAP USAGE

P0f needs to use the pcap libraries system in order to analyze the incoming packets; to achieve that it needs root privilegies. For this reason we have created a small script *make_p0f.sh*, which compiles the code and sets up the sticky bit, so that the user can run the program without being root.

3 - P0F-QT INFORMATION MANAGEMENT

In order to represent the informations we create a model with an unique class, *p0f_info*, representing the packet, in which there is an hash map that maps the p0f keys word with their current values.

p0f_info()

```

enum info_type{
    MTU_INFO,
    HTTP_REQUEST,
    HTTP_RESPONSE,
    SYN_INFO,
    UPTIME_INFO,
    HOST_CHANGE,
    IP_SHARING
};

class p0f_info
{
public:
    p0f_info(QString ip, info_type packet_type);
    QString get_value(QString field);
    QString get_address();
    QString get_info();
    info_type get_type();
    void set_info_field(QString key,QString value);


private:
   QHash<QString,QString> information;
   QString ip_address;
   info_type type;

};
```

4 - P0F WRAPPER

In order to collect all the informations coming from p0f, we have created *p0f_auditor.cpp*, which makes available some functions for p0f; these functions are:

- *create_packet*: it's called by p0f *start_observation()*, that passes the host ip and the keyword representing the type of packet. With theese informations it creates a new packet.

create_packet()

```

void create_packet(char* host, char *keyword){

QString qClient = QString::fromUtf8(host);
if(strcmp(keyword,"mtu") == 0){
     current_packet = new p0f_info(qClient,MTU_INFO);
    }
    else if(strcmp(keyword,"syn") == 0 ||
                        strcmp(keyword,"syn+ack") == 0){
     current_packet = new p0f_info(qClient,SYN_INFO);
    }
    else if(strcmp(keyword,"http␣response") == 0){
     current_packet = new p0f_info(qClient,HTTP_RESPONSE);
    }
    else if(strcmp(keyword,"http␣request") == 0){
     current_packet = new p0f_info(qClient,HTTP_REQUEST);
    }
    else if(strcmp(keyword,"uptime") == 0){
     current_packet = new p0f_info(qClient,UPTIME_INFO);
    }
    else if(strcmp(keyword,"host␣change") == 0){
     current_packet = new p0f_info(qClient,HOST_CHANGE);
    }
    else if(strcmp(keyword,"ip␣sharing") == 0){
      current_packet = new p0f_info(qClient,IP_SHARING);
    }
}
```

- *add_info*: adds to the current packet's hash map the key and value fields, which are passed by p0f in *add_info_field()* function.

add_info()

```
void add_info(char *key, char *value){

    QString qKey = QString::fromUtf8(key);
    QString qValue = QString::fromUtf8(value);
    current_packet->set_info_field(qKey,qValue);

}
```

- *end_packet*: sends the whole information to *network_db*. It's called by p0f *add_info_field()*.

end_packet()

```

void end_packet(){

        nt->add_info_network(current_packet);

}
```

5 - DATA MANAGEMENT

The packets created by *p0f_auditor* are managed by the class *network_db*; its main role is to handle the hosts which are created analyzing the packets and organize them in a *QVector<host*>*. An host is an object, belonging to the *host.cpp*, that has the following attributes:

- *ip_host*: it is the ip address of the host;
- *host_packet*: it is an hash map with an *info_type* key and a *p0f_info* value;

This object represents a machine revelated by p0f.

host.h

```

class host
{
public:
    host(QString ip);
    QString get_ip();
    QString print_packets();
    QHash<info_type,p0f_info*> get_packets();
    p0f_info* get_packet(info_type type);
    void set_packet(p0f_info* info);
    QString get_os();
    QString get_app();
    bool host_with_nat();

private:
    QString ip_host;
    QHash<info_type,p0f_info*> host_packets;


};
```

When a packet is complete, the *end_packet* function of *p0f_ auditor* calls the *network_ db* method named *add_ info_ network*. In this function, first of all it looks for the host to whom the packet belongs. Secondly, if the host matches with one already present in the database, it updates the host informations with the new packet;on the other hand, if it gives no matches, it adds the new host to the database.

network_db.h

```

class network_db
{
private:
    network_db();
    void add_packet_host(p0f_info *packet,
                                  host* new_host);
    host* find_host(QString addr);
    QVector<host*> network;
    static network_db* istance;

public:
     static network_db* get_istance();
     void  add_info_network(p0f_info *packet);
     QVector<host*> get_hosts(){
         return network;
     }
};
```

6 - GUI MANAGEMENT

The main program window is organized in four sections, that implement the chance to select the information source, to search specific hosts and to show informations.

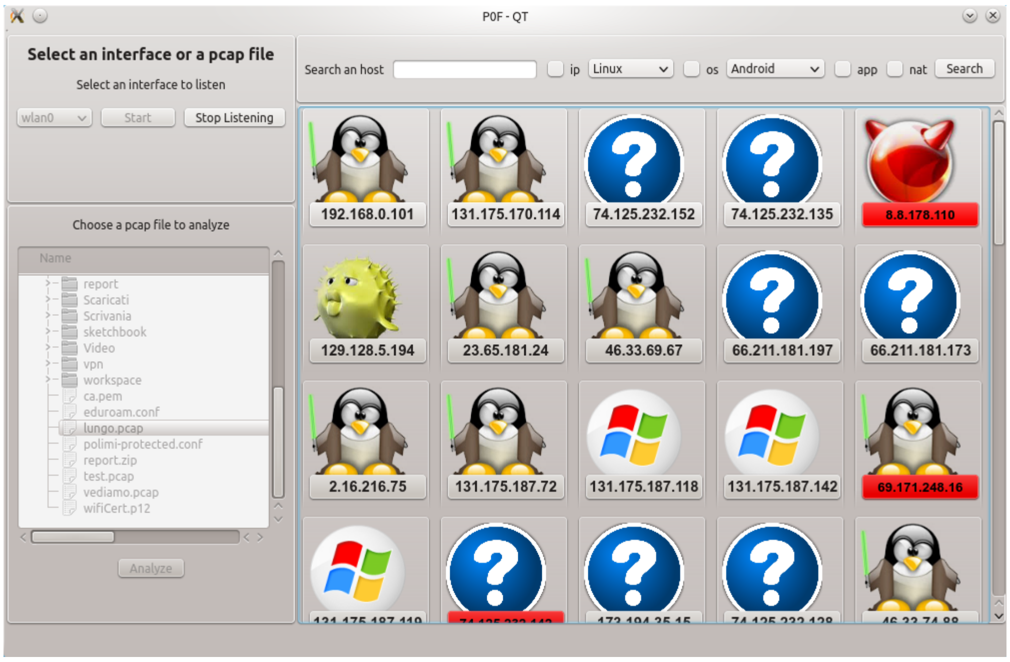
The first section allows to select one of the active network interfaces of the system and has two button:

- *start/restart* button: the first time is pressed it sets the name of the interface to listen to and starts or restarts a timer with 10 seconds timeout that permits to refresh gui informations;
- *stop listening* button: it stops the informations refresh until the next click on the start button;

The second section shows in a *tree list* the home directory, with only read access, in order to select the pcap file to read and analyze.

The third area implements a search function, made with some chech box and combo box in order to select which packet fields have to macht.

In the last section some group box are created run time, one foreach host which corresponds to search criteria. Every group box has an image representing the machine's os and a button: when it's clicked a popup with all host's informations is created. If the host is probably running behind nat, the button becomes red.



7 - TRADE OFF

During the development of the project we have changed the informations gathering method.

The old one creates a gerarchical structure, having *p0f_info* as a superclass. In this class there are attributes common to all the packets, like the host ip and the role it plays (client/server).
This superclass is extended by four other classes:

- MTU_INFO: link, raw_mtu;
- SYN_INFO: os, dist, params, raw_sig;
- UPTIME_INFO: uptime, raw_freq;
- HTTP_INFO: app, lang, params, raw_sig.

p0f_info()

```
class p0f_info
{
public:
    p0f_info(QString ip,int type);
    QString print();
    int isclient(){
        return is_client;
    }
    QString get_address(){
        return ip_address;
    }
protected:
   QString ip_address;
   int is_client;
};

class uptime_info : public p0f_info{
private:
    QString uptime;
    QString raw_freq;
public:
   uptime_info(QString server,int type) :
                                p0f_info(server,type){}
   QString print_info();
   QString get_uptime();
   QString get_raw_freq();
   void set_uptime(QString up_time);
   void set_raw_freq(QString rawfreq);
};

class mtu_info :public p0f_info{
private:
    QString link;
    QString raw_mtu;
public:
    QString print_info();
    mtu_info(QString server,int own) :
                                p0f_info(server,own){}
    QString get_link();
    QString get_raw_mtu();
    void set_link(QString mtu_link);
    void set_raw_mtu(QString mtu);
};

class syn_info :public p0f_info{
private:
    QString os;
    QString dist;
    QString params;
    QString raw_sig;
public:
    syn_info(QString server,int own) :
                                p0f_info(server,own){}
    QString print_info();
        QString get_os();
    QString get_dist();
    QString get_params();
    QString get_raw_sig();
    void set_os(QString SO);
    void set_dist(QString distance);
    void set_params(QString parameters);
    void set_raw_sig(QString sig);
};

class http_info :public p0f_info{
private:
    QString app;
    QString lang;
    QString param;
    QString raw_sig;
public:
    http_info(QString server,int own) :
                                 p0f_info(server,own){}
    QString print_info();
    QString get_app();
    QString get_lang();
    QString get_param();
    QString get_raw_sig();
    void set_app(QString application);
    void set_lang(QString language);
    void set_param(QString parameters);
    void set_raw_sig(QString sig);

};

enum info_type{
    HTTP_INFO,
    SYN,
    MTU,
    UPTIME
};
```

However this approch leads us to a messy and full of unuseful branch if-else code even if it allowed to entirely separate the p0f structure data from the gui implementation. But once we started to implement the gui code, we discovered that this approach was not worth the trouble because of the few informations we have to pull out. For this reason we have decided to adopt the method presented on *p0f-qt information management*, which produces a more adjustable and cleaner code. The biggest issue we haven't succeded to solve is that we can just run p0f on a single input. For this reason we can only listen to one interface or read only one file pcap.